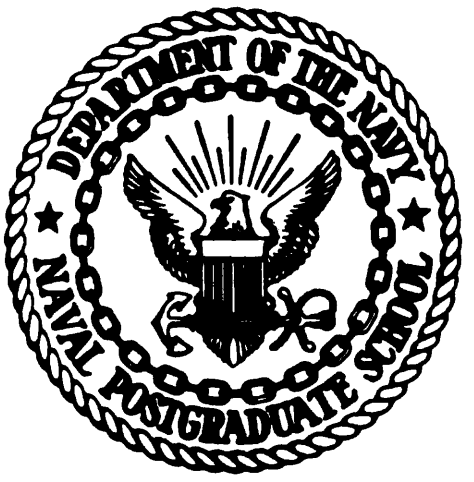
NPS-53ZZ73031A

United States
Naval Postgraduate School



ALL APPLICATION DIGITAL COMPUTER: COURSE NOTES

by

Gordon H. Syms

March 1973

Second Printing August 1973

Approved for public release; distribution unlimited

FEDDOCS
D 208.14/2:
NPS-53ZZ73031A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Mason Freeman, USN
Superintendent

M.U. Clauser
Provost

ALL APPLICATION DIGITAL COMPUTER: COURSE NOTES

ABSTRACT:

This report is a set of course notes, or text, on the proposed Navy All Application Digital Computer. The AADC, as it is called, is a programmer-oriented, general purpose, modular digital computer that was originally designed to meet all the 1975-1985 Naval airborne data processing requirements, but it has now had its role generalized to include "All Applications." Since the AADC combines many of the most advanced computer hardware concepts now under development in the United States, the study of AADC should be of general interest.

The all application role includes real-time and time-sharing computations, and special applications such as line concentrators, super modems, data channels and aircraft electric power controllers.

This report includes a chapter on each of the following: a general introduction and summary of all chapters, AADC architectures, all application role, hardware technology, Data Processor Element, Master Executive Control, Signal Processing Element, evaluating AADC developments, High Order Language, and AADC applications.

The report will be used for a 33-hour course for graduate students at the Naval Postgraduate School, but could be used for other audiences or for shorter courses.

This task was supported by Naval Air Systems Command under Work Request 2-6297, dated March 23, 1972.

NPS-53ZZ73031A
March 1973

Table of Contents (Overview)

Table of Contents (Overview) - Con'd.

Preface and Chapter Organization

This report* is intended as a study guide for the proposed Navy's Advanced Avionic Digital Computer or All Application Digital Computer. The AADC, as it is called, is a programmer-oriented, general purpose, modular digital computer with special features designed to meet all the 1975-1985 Naval airborne data processing requirements, as well as the normal scientific and business data processing requirements. The AADC combines many of the most advanced computer hardware and software concepts now under development in the United States; and, therefore, the study of AADC should be beneficial to anyone interested in the projected state-of-the-art in computer developments, as well as to Navy personnel.

The general interest in AADC has grown significantly in the last year, since the Navy decided to generalize the role of this powerful and inexpensive computer to include All Applications. Now specially designed features are being added to the original avionic computer to make it suited for normal batch and time sharing computations, without jeopardizing the original real-time avionic features. The AADC also appears suitable for such special applications as line concentrators, super modems, data channels and aircraft electric power controllers.

Although this report could be used as an independent study guide, it will also be used for an eleven week, 3 hours-per-week course for graduate students in computer science, computer systems management, avionics and other students at the Naval Postgraduate School. This study guide could also be used for a one, or possibly two, weeks concentrated course on AADC; or parts of it could be

---

* This report was produced under NAVAIRSYSCOM Work Request 2-6297 dated March 23, 1972.

used for a one or two day introduction to AADC for the Navy or Industry personnel.

This report is organized in a modular fashion - in keeping with the AADC concept - to allow the reader to concentrate on his area of interest without missing any essential background, or continually being diverted to other chapters. Chapter One is the Introduction and Summary. Since it provides the introduction and overview to the AADC development program, it should be reviewed before studying any other chapter. Chapter One also offers a fairly concise summary of all facets of the AADC development, which should be of interest to the more casual reader.

After reviewing Chapter One, any other chapter can be studied, and in any order, depending on the reader's interests. The contents of the Chapters include the AADC architecture, the "all applications" role, hardware technology - including LSI, memory and bussing technology - , the sequential Processor Element, the Master Executive Control, the parallel processor, evaluating AADC developments, and last - and probably the most important - the applications for AADC. The last chapter should be of special interest to non-computer specialists, especially anyone involved with avionics, because it asks the questions, "How could you use this powerful computer, and how many would you want at the very low predicted cost?"

Chapter 1

INTRODUCTION

AND

SUMMARY

Table of Contents for Introduction and Summary

List of Appendices

List of Figures

---

* Reference Number 1.1 located at the end of Chapter One. The first number refers to the chapter number, the second is the reference number.

Glossary for Introduction and Summary

A&C	- Arithmetic and Control Unit for sequential computations; often synomonous with PE.
Baseline	- The largest, or worse case, AADC architecture: contains several PEs, large BORAM, large RAMM, SPE, etc.
BORAM	- Block Oriented Random Access Memory: used to store program modules and permanent data.
CCD	- Charge Coupled Device Semiconductor: competitor for MOS for BORAM.
CFM	- Closed flux path thin film memories, a planar thin film analog of plated wire for RAMM and TM.
CMOS	- Complementary Metal Oxide Semiconductor used in memory arrays or LSI circuits.
Ferroacoustics	- A process of using coincident mechanical (acoustic strain wave) and electrical energy to write magnetic domains into semi-closed path permalloy film - used in BORAM.
HOL	- Higher Order Languages: like CMS-2, Fortran but particularily extensions to these languages.
ITACS	- Integrated Tactical Air Control System: a general aircraft control system scheduled for all 1980 Navy aircraft.
MEC	- Master Executive Control: supervises and controls all AADC modules.
MIPS	- Millions of Instructions Per Second: a measure of processor throughput.
MMM	- Multiple Memory Multiprocessor: an intermediate AADC architecture; see Chapter 2.
MNOS	- Metal N-channel Oxide Semiconductor (my guess) (used in Appendix 1.3).
MOS	- Metal Oxide Semiconductor: used in LSI circuits and semiconductor memories.
MPP	- Matrix Parallel Processor: early version of the parallel processor (Chapter 7).
msec	- Milliseconds = 10^{-3} seconds.

Glossary for Introduction and Summary (Cont'd)

MTBF	- Mean time between failures: a measure of reliability.
NDRO	- Non-distructive read out, i.e., no rewriting required after reading.
nsec	- Nanoseconds = 10^{-9} seconds.
OSP	- Optimize Simplex Processor: simpliest AADC architecture.
PE	- Processing Element for performing sequential computations: actually A&C plus TM; see Chapter 5.
RAMM	- Random Access Main Memory: used to store semi-permanent (mode independent) data and to buffer Input or Output (I/O).
SPE	- Signal Processing Element:- latest version of the parallel processor, like MPP (Chapter 7).
TDM BTM	- Time Division Multiplexed Block Transfer Multiprocessor: an intermediate AADC architecture.
TM	- Task Memory attached to PE and holds the currently executing program module and temporary variables.
TPP	- Three-Plus Processor: the ultra-variable AADC architecture with more than three PEs for extra reliability.
μsec	- Microseconds = 10^{-6} seconds.
DPE	- Data Processing Element: new name for the sequential Processing Element.

Chapter 1

INTRODUCTION AND SUMMARY

1.1 INTRODUCTION TO AADC

The All Application Digital Computer (AADC) is a programmer-oriented, general-purpose, modular digital computer with special features designed to meet 1975-85 Naval airborne data processing requirements, as well as, the normal batch and time sharing computational requirements. It combines many of the most advanced computer hardware and software concepts now under development in the United States.

1.1.1 AADC Design Philosophy

The AADC is a modular computer, designed to be inexpensively assembled from off-the-shelf large scale intregated (LSI) silicon wafer and advanced magnetic thin-film memory building blocks. It can be configured as a simple minicomputer, a super-multiprocessor, or anything in between. It is truly a fourth generation computer, employing hardware and software building blocks to construct the various computer systems. The cost should be one to two orders of magnitude less than today's state-of-the-art computers. The computers should also be one-tenth the size and weight, and should exhibit remarkable reliability.

Originally, AADC was the acronym for Advanced Avionic Digital Computer. The development of the AADC was the result of analyses into next-generation Naval aircraft computing requirements, as well as a serious attempt to find ways to reduce the enormous cost of computer procurement and support through the application of standardization and modularity. In the past, the designs of computers that were developed by private industry were frequently so different from one another that system evaluation by even the most qualified engineers was often extremely difficult.

To insure the availability of an adequate digital computer for the years between 1975 and 1985, the Naval Air Systems Command decided in the fall of 1968 to pursue an active computer development effort, originally named the Advanced Avionic Digital Computer Program. The ultimate success of this development will hinge on several basic engineering and management decisions made that year. First, equal emphasis would be placed on system hardware, software, and technology development. Second, no one company would be permitted to develop the computer; rather, jobs would be parceled out on the basis of vendor competance in each critical area, and only after open competition. Third, dependence upon proprietary designs and concepts would be minimized. [1.1]*.

In the last year the Navy has recognized the power of the AADC and its relatively low cost and have decided to generalize its role to "all applications". This involves several additions to the original design requirements.

To achieve its goal the AADC program requires the cooperation of Government and Industry personnel in a coordinated effort that will result in new capabilities in computer design, digital technology and microelectronics in general.

The AADC will provide a single family of hardware and software modules from which can be assembled computers of varying capacities that will satisfy the entire spectrum of Navy airborne and general purpose computing requirements. Exploitation of Large Scale Integration (LSI) digital logic circuitry and monolithic magnetically coupled thin film storage will allow use of powerful machine organizations and programming techniques within the weight and size constraints

---

* Denotes Reference 1 at the end of Chapter 1, i.e., chapter number followed by reference number.

of future aircraft. Replacing the present multiplicity of airborne computers with machines constructed of common modules will result in large savings in R&D, procurement, maintenance, training, and programming, and provide computers capable of adapting to, and growing with, evolving avionic system requirements.

Rather than present any more of the AADC philosophy here the reader is referred to Appendix 1.1, which is a September 1970 description of the AADC program by Ronald S. Entner, the System Architect and Project Manager of AADC at Naval Air Systems Command [1.1]. There are only two major changes that have been made since that article: first, the Data Processing Elements or DPEs are no longer organized as byte-functional modules because the current design by Raytheon is a word-oriented DPE [Chapter 5]; and second, the Matrix Parallel Processor has undergone several changes and is now called the Signal Processing Element or SPE [Chapter 7]. Another smaller change is that the DPE is not a microprogrammed computer, although it does have some microprogramming capabilities.

In a draft to a follow-on article, Mr. Ronald S. Entner describes the progress of the AADC program one year later in October 1971. Although this article was not published, it does give a very good overview of the complete AADC developmental project in its many facets, including DPE performance, cost and performance of various memories, a general purpose array processor (now called SPE), instruction utilizations in Navy aircraft, software development, appraisal of hardware technology and general progress of the AADC project [1.2]. Some of the latest developments, including the all application role, will be discussed in a later subsection.

Several articles have appeared in the popular press and excerpts from these will provide further background on the AADC program. An article in the August 3, 1970 issue of Electronics [1.3] contains a good overview of the AADC; it has several figures on costs and timetables in the first half of the paper, and an outline of the breakthroughs in LSI and memory technology necessary to make AADC realistic in the second half. The timetable is particularily interesting since it states that "So far [in August 1970], $12 million is expected to carry the AADC through the feasibility stage; $1.5 million is budgeted for fiscal 1971. The $1.4 million spent so far has been divided among 19 contractors. AADC feasibility must be established by the end of 1973; operating hardware is to be available for evaluation in 1974" [1.3]* Apparently the Defense Department has been paying up to $150 per word for program development (the standard industry figure is $10/word), and, by one count, the Pentagon was supporting as many as 287 different airborne computer efforts at one time.

In August 1970, the cost of producing a 2 million operations-per-second computer with 80,000 words of memory was estimated by Mr. Entner to be $30,000. "In quantity the cost should drop to about $13,000." The additional associative fast-Fourier elements and arithmetic units would create a machine comparable to the most powerful computer on the market today, and for only about $100,000.

One critic in the same article suggests that, "Entner's group talks primarily about the processing elements, but that's the least significant part of the unit. Input-output is the monstrous part of the system in engineering terms, and memory is the most expensive. Here's where they are going to have the problems." Another comment is, "Ron [Entner, AADC Program Manager] has done a fantastic job of interesting industry in the program at its own expense" [1.3].

*In January 1973 the AADC program is still on schedule with March 1974 as the predicted delivery date for the Advanced Development model.

Another article in the June 22, 1970 issue of Aviation Week and Space Technology gives a good view of several Navy avionic computer systems under current development, and provides the motivation for one set of computer modules like AADC. The last half of the article is a fairly general and vague description of the AADC concepts but the last few paragraphs on the LSI technology are interesting [1.4].

A June 1971 article in Aviation and Space Technology describes the application of AADC to the Integrated Tactical Air Control System (ITACS) that is scheduled for operation in new aircraft starting in the 1980s. It is anticipated that the AADC could be used as an integral part of the ITACS system controlling a variety of antenna elements, RF (Radio Frequency) heads and the modem with its programmable frequency synthesizer and matched filters, as well as, providing navigation and fire control computations [1.5].

The underlying motivation for the AADC is one of cost. The following is the estimated computer specifications for a conceptual advanced Naval Aircraft for 1980. The figures are the result of extensive analysis and therefore should be considered realistic [1.2]. The specifications are:

1) Throughput capacity: 3.6×10^6 ops/sec,
2) Random access storage: 8×10^5 words,
3) Bulk storage: 10^6 words,
4) Computer cycle time: 1.0 microseconds,
5) Weight: 30 lbs, and,
6) Volume: 870 cubic inches.

The life cycle cost of providing this with a conventional aircraft would be about $2 million, but with the AADC it is estimated at $100,000 or 1/20 of the cost. The relative portion of the cost for development, procurement, modification and maintenance are depicted in Figure 1.1. As can be seen, the largest percentage decrease with AADC over the conventional computer is in the maintenance. While the relative percentage of procurement costs goes up significantly, the relative cost of development and modification remain approximately constant. Thus the total cost of AADC is about 1/20 that for a conventional aircraft with the largest proportion being for procurment instead of maintenance.

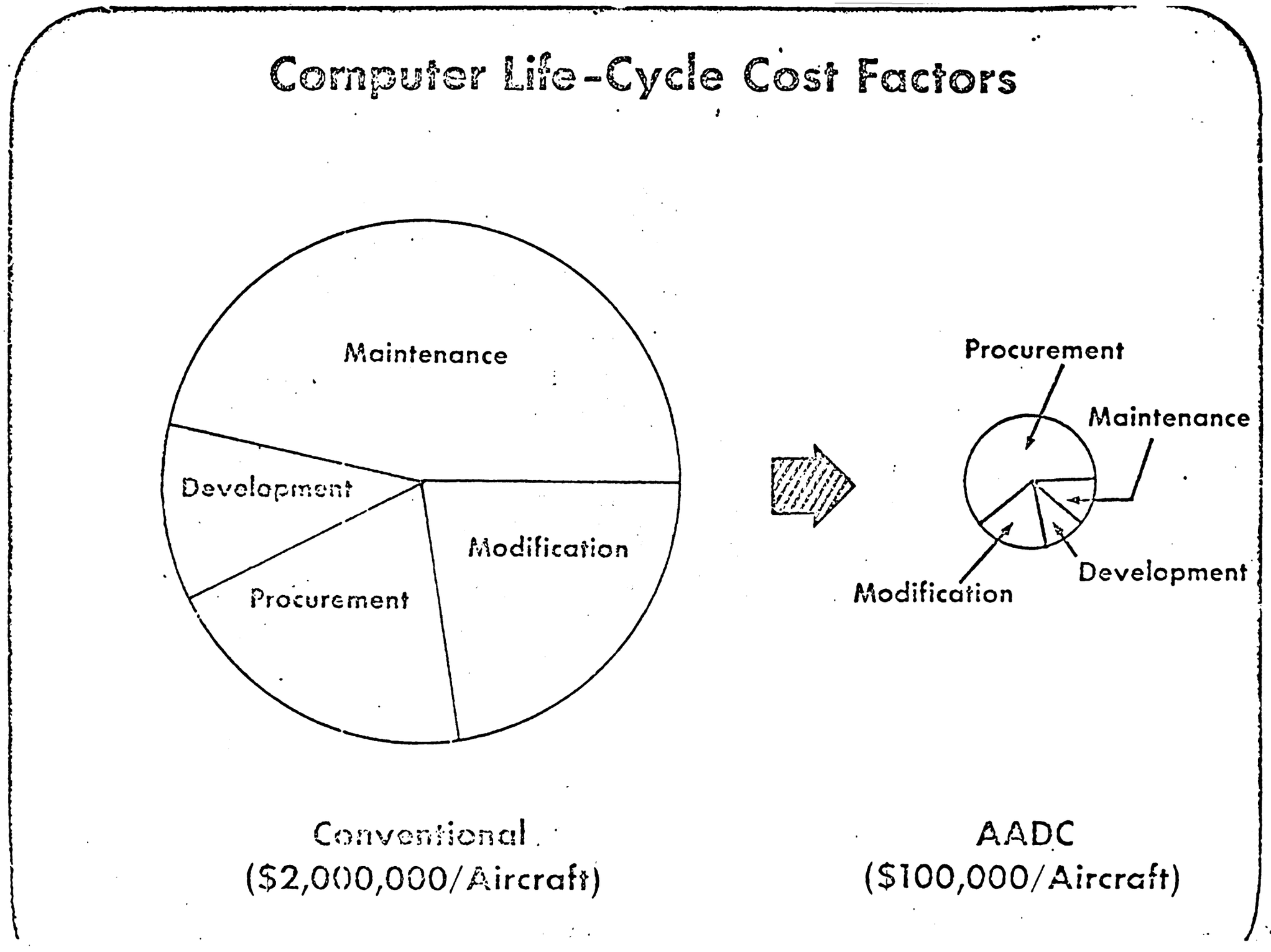


Figure 1.1. Relative Life Cycle Costs of AADC
1.7

1.2 OBJECTIVES OF REPORT AND COURSE

The AADC system should be of interest to all computer specialists, as well as Navy personnel, who are interested in new developments in computer systems, because the AADC incorporates many of the present and future state-of-the-art hardware and software technologies. This report is intended as the basis of a comprehensive AADC course.

The primary purpose of this report is to organize the AADC literature into "bit-sized chunks" so that it may be more readily "digested." Already the AADC literature represents thousands, and probably tens of thousands, of pages of description of AADC concepts, design philosophies, design alternatives, equipment specifications, operating characteristics and possible applications. This report is intended to organize all the AADC descriptive material so that a reader can easily locate the portions of interest and can obtain an overview of the pertinent sections. In keeping with the AADC philosophy, the material is organized in several modules or chapters that are each independent and self-contained (see Section 1.5). The material in this report is taken from the AADC literature and is referenced accordingly. In this way, the report represents a study guide for anyone wanting to learn more about the state-of-the-art in computer systems development and particularily that of the AADC system.

This report is the basis for a comprehensive course on the AADC suitable for personnel with some computer training and experience, who are interested in future computer technology. The course includes all aspects of AADC of interest to the computer specialist from design concepts to system compatibilities, as well as, the new applications that become practical with this powerful computer system. The course will be given to NPS graduate students as a three-credit

three-month course, but it could also be given as a one or two week short course for other Navy personnel or, in a shortened version, as a one to three day course to Industry. Two shortened versions of 12 lecture hours has already been tried at NPS, as special courses.

1.2.1 Justifications for an AADC Course

Some more specific justifications for developing and teaching an AADC course at this time are:

1. To inform the Navy of AADC. This course, when given to students and other Navy personnel, will produce a large group of informed Navy personnel who understand the AADC concepts and features and who are interested in AADC developments.
2. To inform the Naval students of future avionic computer technology.
3. To encourage NPS students in the conceptual design, development and applications of AADC. The students have the time (in terms of class and thesis projects), field experience and resources to make significant contributions to AADC, especially in the applications area.
4. To develop managerial guidelines for the use of AADC in Navy Systems.
5. To present seminars on AADC to industry, short courses to Navy personnel, as well as the regular quarter courses to Naval Postgraduate School students. A Comprehensive set of notes will make this task much easier.

In conclusion, when the Navy undertook the supervision of the development of the AADC system, it made a big step in controlling the design of computers for the Navy needs. In order to make this project truly effective, the Navy must have trained personnel ready to incorporate the AADC into existing or new applications in such a way as to maximize the usefulness of all its capabilities. Never before has the Navy known this far in advance what the future Navy computers will be, and now the Navy has a chance to develop applications for this computer while it is still being developed, instead of after it is in production. If the applications for AADC are ready when the equipment becomes available, the Navy will have made another major step in solving its computer oriented problems.

1.3 HISTORICAL DEVELOPMENTS OF AADC

The initial development of the AADC concept began in 1968 with two studies by Hughes Aircraft Company and Honeywell Inc. on future requirements for Naval avionics computers [1.6 and 1.7]. The first conference on AADC was held on February 27, 1969 to inform industry of the AADC concept and to ask for their cooperation in the AADC development [1.8].

The initial AADC Baseline definition was published by Ron Entner in three different but similar versions: first, in a Spartan book copyright 1970 but with the article dated March 16, 1969 [1.9]; second, in a NAVAIRSYSCOM report dated July 1969 [1.10]; and finally, at the second AADC conference in September 15, 1969 [1.12, 1.14]. The AADC organization is basically the same in all three articles and was essentially the same in 1969 as it is now at the end of 1972. For example, the AADC Baseline organization is shown in [1.9] and dated March 16, 1969 consists of the same processor element with memory, executive control, main memory, bulk memory, matrix parallel processor and I/O. The only differences at that time were: first that the PEs were organized as byte-functional modules; in other words, it took 4 PEs each operating on a byte to perform 32-bit word operations; second, the routing switch between the PEs and the memory has been eliminated by changing to word-functional modules; and finally the Matrix Parallel Processor has undergone several steps of evolution [Chapter 7].

1.3.1 The Second AADC Conference

The second conference, which unofficially marked the first birthday of the AADC program, was held on September 15, 1969 [1.12]. Four papers constitute the proceedings of that conference and describe the basic AADC concepts. The

first paper describes the motivation for AADC, the modular AADC philosophy and advantages and disadvantages of the AADC concept. The primary advantages are an expected 5 times reduction in size, 20 times reduction in cost and a 10 times reduction in mean time between failures (MTBT) [1.13].

The second paper reiterates the AADC goals, modularity concept and avionics computer tasks, and the present eight possible computer organizations. These are: 1) A unit or simplex processor consisting of processor, memory and I/O units; 2) A federated multiple processor consisting of two simplex processors with the I/O units interconnected; 3) A dedicated multiple processor - which is essentially the same as the federated multiple processor except that both memories and processors are connected to a single I/O unit; 4) A shared memory multiprocessor consisting of two processors with only a single memory and I/O unit; 5) A multiple memory multiprocessor with at least two processors, two I/O units and several memory modules all interconnected; 6) A pipeline multiprocessor with a commutator and several pipeline functional units (similar to CDC 6600 CPU); 7) A multiprocessor with dedicated task memories, as well as, a common memory; and finally, 8) The AADC baseline systems [1.14]. Out of these eight possible organizations, four were selected later for further study. They were: the Optimized Simplex Processor (OSP), the Multiple Memory Multiprocessor (MMM), the Time Division Multiplexed Block Transfer Multiprocessor (TDM BTM) - which is essentially a Baseline system without the hardware executive -, and the AADC Baseline system to handle the worse-case conditions. (More details on the organizations will be presented in Chapter 2.) Some of the concepts of software modularity and the Master Executive Control (MEC) are also presented in the second paper [1.14].

The third paper of the September 1969 Conference describes the AADC hardware considerations including LSI technology and possible optical computer memories (which have not yet materialized). Of special interest is a summary of digital logic gate characteristics [1.15, page 8] which is presented in Subsection 4.3.3.

The longest and certainly the most detailed presentations is the final paper on the Baseline associative processor [1.16]. Unfortunately, the parallel processor area has undergone the most evolution in the last three years and, therefore, the paper is the least reliable reference for the current status. See Chapter 7 for more up-to-date information.

1.3.2 Miscellaneous Historical Developments

The first report on the Master Executive Control was written by Honeywell in July 1969 [1.17] and the second by Ron Entner in December 1969 [1.18]. There were two other proposals for the AADC system by Grumman Aerospace Corporation in July 1969 [1.19] and by General Electric Company in August 1969 [1.20] but these proposals have not been accepted. Raytheon Company also produced two classified reports on the integration of AADC into operational systems [1.21 and 1.22]. A proposed technical approach report for AADC was written by Ron Entner in December 1969 but it was for "Official Use Only" [1.23]. The first simulation study was done by Univac Advanced System Group in November 1969 [1.24].

These documents are listed here for historical purposes and to give credit where due to the initial developers of AADC. The documents are not considered critical to the development of this report or to a course on the present AADC system and, therefore, have generally not be obtained or reviewed at NPS.

1.3.3 AADC Progress Reports

The AADC Development Program Progress Reports written by Ron Entner of NAVAIRSYSCOM are also a good means of following the progress of the AADC project. Project Reports numbered 1 to 4, written in November 1968, February 1969, May 1969 and December 1969, respectively, report the initial development of AADC [1.25 to 1.28]. Of these, Progress Report Four is the most significant since it reports on the September 1969 conference as unofficially marking the first birthday of the AADC effort and being an outstanding success. It reports on the introduction of the Block Oriented Random Access Memory (BORAM) as an important building block in AADC. It also contains as enclosures the preliminary statement of work for RFP for the MEC analysis design study and the AADC software considerations. Progress Report Four also discusses the Navy's MINCOMS (Multiple Interior Communication System) which is a means of simplifying the AADC I/O functions by standardizing the data formats and by providing AADC with control of the communication system between the computer and the outside world.

AADC Progress Reports Five through Eight present the AADC process from March 1970 to July 1971. Progress Report Five presents the effect of future avionic requirements on the AADC instruction repertoire, as well as the effects of the requirements of the AADC Baseline system on the AADC instruction repertoire [1.29]. Progress Report Six contains: 1) An AADC technology summary including cost information; 2) AADC associative processor interim report; 3) A memorandum entitled: "AADC workload characteristics requirements"; and, 4) an advanced memory technology progress note [1.30]. Progress Report Seven contains an AADC bibliography, a preliminary statement of work for a high level programming language

development, and a discussion of software modularity [1.31]. Progress Report Eight contains the following: 1) Preliminary statement of work for an analytical study to establish the feasibility of a tactical interactive programming facility; 2) Summary sheets of AADC program review; 3) A paper entitled: The programmer as a computer designer; 4) AADC status report; 5) Storage technology and AADC architecture; and 6) The agenda of the advanced digital technology conference, June 1971 [1.32].

Progress Reports Nine and Ten, dated November 1971 and May 1972 [1.33 and 1.34], present the current status of AADC and will be discussed later in Section 1.4.

Progress Reports Three through Eight are available through the Defence Documentation Center as referenced.

1.3.4 AADC Conferences

So far six conferences have been held on the AADC program*. The first held on February 27, 1969 was intended to inform industry of the AADC concept and to ask for their cooperation in the AADC developmental project [1.8]. The second conference was held on September 15, 1969 to describe the AADC philosophy, the possible computer organizations, the hardware development and the matrix parallel processor, see Section 1.3.1 above or [1.12].

The third conference was held on June 20-30, 1970 and discussed the HOL (Higher Level Language] requirements for aerospace computers [1.35]. Quoting from the introductory remarks:

*Actually seven counting the AADC 1973 Symposium.

> The languages discussed at the conference were Compiler Monitor System - 2, Space Programming Language, and Computer Language for Aeronautics and Space Programming. The purpose of the conference was to address the relative merits of each language with respect to avionic applications, as well as discuss high level aerospace programming language compatibility and computer hardware requirements (i.e., common instruction repertoires, standard word formats, etc.) which could lead to some measure of compiler standardization.

In particular, the conference discussed the characteristics needed for programming avionic applications and how the AADC instruction set could be matched to a suitable HOL.

The fourth conference is the Symposium on the Advanced Aircraft Electric Systems (SOSTEL) held April 20-22, 1971. The conference discussed the replacement of conventional electro-mechanical power-distribution devices with digital computer technology, multiplexed data transmission principles and solid state switching devices to improve the means of managing, controlling and distrubuting aircraft electrical power in the future [1.36].

The fifth conference on the Advanced Digital Technology was held June 8-10, 1971 and discussed the latest developments in LSI and memory technology. In total 27 papers were presented including papers on material growth and preparation, microelectronics processing, switching and memory devices and circuitry, LSI circuit interconnection technology, LSI test generation and array testing, LSI packaging technology, optical communications, and the implication of new computer architecture and memory technology on future computer systems [1.37].

The sixth conference was the AADC Software Conference on Command Control Software Technology for 1975-1985 held February 1972 and cosponsored by NAVAIR-SYSCOM and NELC. The purpose of the conference was to address the questions of

requirements that will be imposed on software systems and the methodologies that will be available to satisfy those requirements in 1975 to 1985. The conference also allowed an important segment of the software community to be introduced to the hardware and architectual comcepts embraced by AADC; and at the same time, provided an opportunity for open discussion of the AADC software goals and particularily the implication of using CMS-2 language as the basic AADC HOL [1.34]. Conference proceedings are not yet available.

One other conference, namely the National Aerospace Electronics Conference held May 17-19, 1971, is mentioned here because of its general applicability to the AADC problems and applications [1.38].

The last conference is the AADC 1973 Symposium held in Orlando, Florida, on January 23-25, 1973. Some results from this Symposium will be presented in Subsection 1.4.4.

1.4 CURRENT AADC DEVELOPMENTS

Since most of this report is based on reports that are about one year old and since the AADC System is in a continual developmental stage, this section will describe some of the latest developments.

1.4.1 All Applications Role

Certainly the most significant change in the AADC program in the last year is the change in emphasis from only avionic applications to all applications. This has caused significant changes in the AADC design by requiring many of the same features that produced so many problems in the present third generation computers. For example, rather than having a Processing Element (PE) executing a single program out of its own Task Memory, the PE must now have facilities for multiprogramming, virtual memory, demand paging and storage protection. Some of these features may even require the PE to have its own nucleus of an operating system, as well as relocation hardware to support the virtual memory. Also the traffic on the buses will increase significantly. The AADC designers will be required to solve many major operating system problems, such as thrashing (excessive paging until throughput is almost zero) and system deadlocks, that remain unsolved in present-day computers. In any case, the AADC supporters are convinced they can beat these problems with the very powerful AADC. The design changes for the All Application role is discussed in AADC Progress Reports Nine and Ten [1.33 and 1.34].

1.4.2 AADC Project Report Nine

Progress Report Nine presents the problems of 1) addressing a large virtual memory with only 12 bits in the PE address field; 2) multiprogramming

and demand paging on the OSP System where the MEC shares the PE; 3) adequate storage protection when several programs are concurrently resident in the TM; 4) binding at run time instead of compile time (this is usually an advantage except when time is critical in a real time application); 5) program maintenance in a more complex system; and finally, 6) the problem of using tag bits to protect data and programs [1.33, pages 1-10].

Progress Report Nine states that, "As a result of a recent appreciation for the processing power of the AADC/OSP, an interesting modification was made to existing MEC design goals." Because the unit processor provides the necessary throughput to meet the combined sequential processing needs of an integrated 1980 aircraft, the multiprocessing capability should be used for increased reliability rather than throughput. Thus four new classes of multiprocessors have been identified. These are a single PE, the dual PE capable of running MEC or application programs on either PE, the Triplex Processor using three PEs with majority voting, and finally the Three-Plus Processor which is capable of running as a Triplex Processor but has the added capabilities from extra PEs in case one fails.

Enclosures (1) and (2) to Progress Report Nine present recent Navy thinking on the subject of improving CMS-2 programming language to meet AADC needs; see Section 9.3 or [1.33, pages 13-58].

1.4.3 AADC Progress Report Ten

AADC Progress Report Ten reports recent thinking on several subject areas including: 1) the problems of the All Applications role; 2) BORAM developments; 3) Advanced Avionics Fault Isolation System (AAFIS); 4) Improvements

in CMS-2; 5) External I/O; and finally, 6) the Signal Processing Element (SPE) [1.34, pages 1-10]. Also listed are seven major tasks on which contractors and NAVAIRSYSCOM efforts are being concentrated. These include LSI packaging, BORAM and RAM memories, requirements for F-14 and A-7 aircraft, further development of MEC, internal bussing, further development on the PE (or A&C) design, and demand paging. Of particular interest is page 17 of Progress Report Ten because it contains a partial listing of the Plans for Fiscal Year 1973.

1.4.4 AADC 1973 Symposium

The latest development at the time of writing is the AADC 1973 Symposium held on January 23-25, 1973. The Symposium covered a wide variety of AADC subjects including a keynote address by RADM Rice, TADSO, the current status of AADC program by NAVAIR and NADC, AADC tradeoffs for NTDS, the Data Processing Element and 1/O controller by Raytheon and IBM, AADC simulations and the Signal Processing element by NRL, Master Executive Control preliminary design by Honeywell, revision to CMS-2 for use with AADC by Intermetrics, as well as, eighteen presentations on hardware developments. The most significant results from the conferences are:

1. RADM Rice's and TADSO's unquestable support of AADC. Other projects are being cancelled waiting for AADC. According to RADM Rice he has support of ADM Kidd in this project too.
2. The Advanced Development Models for the Data Processing Element (DPE - new name for the PE) and the Signal Processing Element (SPE) are scheduled for delivery in March 1974.

3. The instruction set for the DPE has been simulated so that DPE programs can be written and debugged.
4. The DPE now uses a 16-bit address field vice 12 bits.
5. A Microprogramming Language (AMIL) has been developed for the SPE so that its Microprogrammed Control Unit (MCU) can be programmed in a Fortran-like language rather by specifying bit patterns.
6. An AMIL translator has been developed to convert AMIL programs to bit patterns for the MCU.
7. A MCU simulator has been developed to run and test programs written in AMIL. This in the start of a complete SPE simulator.
8. A preliminary design for the Master Executive Control (MEC) has been completed.
9. Many new developments have been made in the hardware technology (LSI, RAM, BORAM and bussing) which indicate the AADC is technically feasible.
10. A new programming language - a revised version of CMS-2 - called CMS-2K has been proposed as the kernal AADC language. Other languages, such as CMS-2, Fortran COBOL, Jovial, APL, etc., - or variations of these - will be developed later as extensions to CMS-2K.
11. The last, and probably the most significant, development from the 1973 Symposium is the need to demonstrate the applicability and strategy of AADC to a wide variety of Navy problems. For example, according to Capt Roth,

FCDSSA, San Diego, it is not sufficient to show technical feasibility and low cost - because computer hardware (LSI) costs are only 0.3 to 0.5 percent of the total NTDS cost -, but it is necessary to demonstrate that the AADC program will result in a reduction in the complexity of the computer software and thus a significant improvement in the computer software maintainability and reliability. This demonstration must be for specific and realistic applications.

Since the 1973 Symposium covers almost all aspects of the AADC program and only a few of them have been covered briefly here, it is recommended that the reader obtain a copy of the Symposium proceeding as soon as they become available - hopefully by April 1973 [1.41].

1.5 BRIEF OUTLINE OF CHAPTERS

The chapters are organized in a modular fashion - in keeping with the basic AADC concept. Thus each chapter is largely independent and self contained and has its own tables of contents, figures and tables, its own glossory of terms, text material and a list of references. Appendices and problem sets are optional. Thus, each chapter can be studied with a minimum of reference to other chapters. Furthermore, other than Chapter 1, which is a general introduction, the chapters can be studied in any order depending on the reader's interest. It seems very appropriate for a computer system with modular hardware and software systems to also have a modular course.

This section will give a very brief outline of each course module. More detailed versions are given in the next section.

Chapter 2 (or module 2) describes the AADC architectures from the Optimized Simplex to the Baseline System and to the new Three-Plus Processor. It also describes each of the basic hardware modules.

Chapter 3 presents the design implication for the all application role, including multiprogramming, virtual memory, paging and storage protection.

Chapter 4 describes the developments in hardware technology, including: 1) developments in LSI technology that allows up to 5000 gates on a 3-inch diameter chip at very reasonable prices; 2) the developments in memory technology for the BORAM, RAM and TM, which provides memory access time from 70 to 150 nanoseconds for 0.1 to 5 cents per bit; 3) optical bussing technology with very high transfer rates; and, 4) new solid state electric power for increased realiability and lower weight.

Chapter 5 describes the very powerful, very small and very inexpensive Processor Element capable of executing 3.3 million instructions per second, occupying one-third of a cubic foot and costing as low as $600. Chapter 6 describes the three versions of the executive; the hardware MEC, the dedicated software MEC and the floating software MEC and compare the three on different AADC architectures. Chapter 7 describes the Parallel Processor which is probably the lease well defined and the most likely module to be redesigned. This module has been called the Matrix Parallel Processor (MPP), Bulk Parallel Processor (BPP), the General Purpose Array Processor (GPAP), or the Signal Processing Element (SPE). Chapter 8 discusses the means for evaluating AADC developments including simulations, breadboarding, and measuring systems in operation.

Chapter 9 is devoted to the AADC High Order Language developments and particularly what features should be added to the CMS-2 language to take advantage of the powerful AADC system to effectively handle the future applications. The most important problems are in reducing program developmental cost, reducing program complexity and improving reliability. All of these can be boiled down to improving software debugging techniques. The final and probably the most significant chapter is Chapter 10 which discusses the applications of AADC. How can this powerful computer system be used to effectively solve the Navy's operational problems?

Again, it should be emphasized that the chapters can be studied in any order after the first one. For example, a avionics specialist with a minimal computer background, who is interested in the operational aspects of AADC, can study the HOL and AADC applications in Chapters 9 and 10 by skipping over Chapters 2 to 8 completely. For the reader that is continuing on to other chapters, the next section should be skipped because it is basically the first section from each chapter.

1.6 SUMMARY SECTIONS OF CHAPTERS

1.6.1 Introduction

This section contains the introductory and summary sections of each chapter and is presented here to make this module self sufficient. This chapter can be used as a introductory one-day seminar suitable for informing Navy or Industry personnel on the AADC developments projects. Note the third digit in the subsection number corresponds to the chapter number.

1.6.2 Introduction and Summary to AADC Architectures

Chapter Two describes the AADC architectures from the simpliest processor - called the Optimized Simplex Processor (OSP) - to the most powerful multiprocessor - the AADC Baseline System - and to the new ultra-reliable Three-Plus Processor (TPP) system. This chapter also discusses the interconnections between AADC modules such as internal bussing and external I/O interconnections. Finally this chapter acts as a "catch all" for subjects which do not fit in any other chapter and pertain to the overall system organization or operation. This also includes some directly-executing High Order Language architectures which are interesting alternates to AADC.

The basic hardware building blocks of any AADC system are: 1) a Block Oriented Random Access Memory (BORAM) to hold program modules; 2) a Random Access Main Memory (RAMM or RAM) to hold semi-permanent data and to buffer I/O; 3) a small (4k word) Task Memory to hold the currently executing program module and temporary data; 4) Processor Elements (PEs)* to perform the sequential arithmetic computations; 5) an optional Matrix Parallel Processor (MPP) or Signal Processing Element (SPE) to process radar and video signals; 6) one or several

*The new name is DPE for Data Processing Element.

Input/Output Units; 7) the internal bussing to interconnect all the modules; and finally 8) a Master Executive Control to control all the modules and supervise the operation of the entire system.

The simpliest system is the Optimized Simplex Processor (OSP) with a single PE with its TM, a RAMM, a BORAM, an I/O unit, internal bussing and a floating software MEC.* The PE executes the MEC out of RAMM; this is the only case in which instructions are executed from RAMM. The PE also executes Program Modules out of the Task Memory. The most powerful system is the AADC Baseline system which contains several PEs with their TMs, a large RAMM, a large BORAM, several I/O units, a Signal Processing Element, four internal busses and a hardware MEC.

Between the two extremes, two architectures have been defined. There is a Time Division Multiplexed Block Transfer Multiprocessor (TDM BTM) which is essentially the same as a Baseline system except with a software MEC. There is also a Multiple Memory Multiprocessor (MMM) which has several RAMMs but no TMs. In this case the DPEs execute programs directly from the RAMMs.

Since the AADC PE is a very powerful processor capable of executing 3.3 MIPS and relatively inexpensive, it is deemed more important to increase the reliability rather than the throughput. Three extra reliable configurations have been defined. The Dual Processor has two OSP systems each capable of providing complete backup for the other. The Triplex Processor contains three OSP systems with majority gate decision logic sampling their output for added checking of random errors. The ultra-reliable configuration is the Three-Plus Processor which is the same as the Triplex Processor, except it has extra PEs that can be switched in automatically in case a PE fails.

---

* A floating software MEC is an operating system which runs on any available DPE on an as-required basis.

1.6.3 Introduction and Summary to All Applications Role

Although the AADC was originally intended for Naval avionic applications only, the powerful features and the low cost have caused the proponents to consider much wider applications. Although most of this report addresses the AADC design for the avionic applications, Chapter Three discusses some of the implications of the decision about a year ago to convert the AADC to an All Application Digital Computer. Although "all application" is undoubtedly too general, it was decided to retain the acronym AADC because it has been in existence for 3 years and because All Application Digital Computer sounds better than Almost All Application Digital Computer.

1.6.3.1 Implications of All Applications Role

Certainly the most significant change the AADC program in the last year is the change in emphasis from avionic applications only to all applications. This has caused significant changes in the AADC design by requiring many of the features that have caused so many problems in the present third generation computers. For example, rather than having a Processing Element (PE) executing a single program out of its own Task Memory, the PE must now have facilities for multiprogramming, virtual memory and demand paging. Thus, the PE must now have relocation hardware to support the virtual memory, and much faster busses to handle the increased bus traffic. Furthermore, the AADC designers must now solve many problems, such as thrashing (excessive paging until throughput drops to almost zero) and system deadlocks, that have remained unsolved in present day computers. In any case, the AADC supporters are convinced they can beat these problems with the very powerful AADC.

1.6.4 Introduction and Summary to AADC Hardware Technology

1.6.4.1 Scope of Chapter Four

Chapter Four discusses the new advances in hardware technology that are being developed for AADC. Although the development and production of modules using advanced hardware technology (at reasonable cost) is very important to AADC, the details of the technology and how it is implemented is of minimal interest in a course such as this one on the concepts and operations of AADC. In other words, the fact that the technology exists, has been proven, and can be mass produced at reasonable cost is certainly of interest, but the details of the technology and its implementation is considered beyond the scope of this report. Therefore, this chapter is an overview of the latest hardware technology emphasizing what has been implemented and proven, as well as, what will probably be in production by 1975.

Under the heading of hardware technology Chapter Four places all work which relates to the physical constituents of the AADC - the devices which will ultimately manifest itself in the physical computer. The hardware technology is divided into three major areas: Large Scale Integration (LSI) technology, memory technology and bussing technology.

1.6.4.2 Summary of LSI Technology

The basic AADC hardware building block module is an hermetically sealed (perfectly airtight) package capable of supporting either multi-chip arrays on a ceramic substrate, chip/wafer hybrids, or semiconductor monolithic three-inch diameter wafers - or any combination of these. ("Monolighic" means many circuits attached together to resemble one uniform pattern, i.e., a 5000 gate LSI wafer.) This year (1972) one of two AADC packaging modules has passed environmental testing at Naval Avionics Facility, Indianapolis. A complete

second level packaging system is presently under development at Singer-Kearfott, and will be similarily tested later this year [1.34, paragraph 23].

There is ample evidence that the technology will mass produce 5000 gates on a 3-inch diameter wafer by 1975. Texas Instruments are producing a Logic Slice-Type "P" which has the equivalent of 857 gates on a 1½-inch wafer. Intel Corp. has build an 8-bit parallel microcomputer the MCS-8 on a single chip. There are now examples of 1500 gate LSI chip available off-the-shelf and Honeywell has produced an 1800 gate LSI chip [1.41, Mr. A. Deerfield, Raytheon].

Many other articles on expected hardware developments can be found in the Proceeding of the Advanced Digital Technology Conference in June 1971 [1.37].

1.6.4.3 Memory Technology

Two promising magnetic storage technologies for AADC are the block oriented ferroacoustic memory for BORAM and the random access closed flux path thin-film memory (CFM) for RAMM and TM. The ferroacoustic technology employs the coincidence of mechanical and electrical energy to write magnetic domains into homogeneous, amorphous (non-crystaline), semi-closed flux path permalloy film. (Permalloy is a highly magnetic alloy of iron and nickle.) These domains are subsequently interrogated by way of an acoustic strain wave. A plated wire may be used for the ferroacoustic memory in place of the thin film. The ferro-acoustic memory is low cost (0.1 to 0.5¢/bit), high speed (150 nsec/wd read and 1 - 2 μsec/block access time), high density (5000 bits/in^3), low power (2 μwatts/bit), low weight (7.5 lbs for 64K 36-bit words, i.e., 2.3 magabits), non-volatile, and uses NDRO (non-destructive read out) techniques [1.34, page 13]. Blocks may be 128 to 512 64-bit words. For more details on the technology see [1.37].

Another magnetic technology, tentatively called Cross Tie Memory and similar to a Bubble memory, is also under investigation for possible use in BORAM [1.34, paragraph 26].

The CFM, a planar thin film analog of a plated wire, offers new capabilities for random access magnetic storage. It provides performance heretofore believed realizable only with semiconductors, but without the twin penalties of high power and data volatility. In comparison to previous magnetic memories, CFM is low cost (1¢ to 3¢/bit), high speed (80 nsec access time, 100 nsec read time with NDRO, and 150 nsec write time per word), high density (5000 to 11,000 bits/in^3), low power (100 μwatts/bit), low weight (3 lbs for 4K 36-bit words or 150K bits) non-volitile, and non destructive read out (NDRO).

In comparison to ferroacoustic memories, CFM is 2 to 30 times more expensive, about twice as fast, up to twice as dense, uses 50 times more power, and is 6 times heavier. Thus, a 64K word BORAM costs $2300 to $11,500; a 4K word TM costs $1440 to $4320.

It is believed that semiconductor memories are going to be very competitive by 1975. See [1.41] for more information.

1.6.4.4 Summary of Other Technologies

Because of AADC's very small geometry, modularity and need for wide bandwidth internal busses, optical communication is being considered seriously for AADC internal bussing. The optical bussing* has distinct advantages over all electronic alternatives in the area of noise immunity and ease of connection. See [1.2].

*Optical bussing is the transmitting of data via a modulated light wave transmitted via optical fibers.

The other improved technology is in the electric power distribution system. It is proposed to replace the conventional electro-mechanical relays with a Solid State Electric Logic (SOSTEL) power distribution system. SOSTEL will greatly reduce power consumption, wiring complexity and weight, as well as increasing the control over electrical power distribution. See Reference [1.36] or Chapter Four for further details.

1.6.5 Introduction and Summary to Data Processing Element

The AADC Processing Element (PE)* is a very fast, very powerful, very small and very inexpensive central processing unit (CPU) designed for large scale computing systems. It is one of the basic AADC modules and is designed to handle all the serial processing requirements of AADC. It is capable of executing 2.5 to 4 million instructions per second (MIPS), with effective processing rates of 8 to 10 MIPS. Its power is the result of the hardware implementation of a general deferral mechanism** and numerous powerful operations, especially the polynomial, matrix and vector operations. Most importantly, this fast powerful processor is packaged in an eight inch cube (0.5 cubic feet) and has an estimated production cost as low as $600. (As a comparison the CPU on the IBM 360 model 67 - a third generation large scale computer - executes about 0.3 to 0.5 MIPS, does not have the same powerful instructions, occupies about 125 cubic feet and costs $698,000.) This section will present an overview of the PE features, while later sections of Chapter Five will include a more detailed presentation.

In order to obtain the desired speed it was necessary to overlap the fetching of instructions and their executions. The instruction fetching operates

---

*Now called DPE for Data Processing Element.

**A general deferred mechanism is one that automatically defers the execution of an operator until its operands are available.

at 2.5 MIPS including an indexing operation and 3.3 MIPS without indexing. Since the PE is a Task Memory oriented element, the need for indexing is greatly reduced over previous computer designs, and the latter speed is more appropriate. These speeds are based on a memory cycle time of 150 nanoseconds (nsec). On the other hand, the instruction execution takes 100 nsec for short instructions (equivalent to Adds) and 800 nsec for fixed-point multiplications. With an assumed ratio of 7 short instructions to 3 multiplications, the instruction execution rate of 3.3 MIPS is also, possible. Since the proposed floating point multiplications are faster than the fixed point, the instruction execution rate with floating point operations is 4.0 MIPS.

The overlapping of instruction fetching and program execution is obtained by dividing the PE into a Program Management Instruction Handling Unit (PMU) and an Arithmetic Processing Execution Unit (AP). The two subsystems operate independently and asynchronously permitting the PMU to fetch instructions well ahead of their execution, and while the AP is processing previously fetched instructions. This is generally referred to as "look-ahead," where instructions are prefetched along the most probable branch path. If the results of a branch instruction are not along the expected path, then the stockpile of instructions is discarded and instruction fetching is initiated along the other path. To hold the stockpile of instructions, a sixteen-register queue connects the PMU with the AP.

The power of the AADC PE is demonstrated by the fact that it has many very powerful instructions, many of which are not even available in high level languages and certainly not implemented in hardware on a general purpose computer. For example, the PE has the following features implemented in hardware:

1. All 16 possible boolean functions,
2. A recursive subroutine call capability,
3. A general deferral mechanism that executes arithmetic, boolean and conditional expressions directly without reordering the operations or using excessive storing and fetching of intermediate results,
4. A rapid polynomial calculation capability for trigonometric, logorithmic, hyperbolic and exponential functions (all coefficients are loaded by a block transfer.),
5. Vector/matrix block handling mechanism for 256 component vectors and matrices.

The particular significance of these features to the programmer is that, (1) the general deferral mechanism allows the mixing of arithmetic, boolean and conditional expressions in a single statement - providing the accompanying high order language is upgraded -, and (2) the vector/matrix mechanism allows operations such as the vector dot product and the matrix product to be specified in two machine language statements. In both these cases the High Order Language will have to be upgraded beyond Fortran or CMS-2 before that language can use these powerful machine language (or hardware) features.

As well as being very fast and powerful, the PE is very small and inexpensive. A rough estimate of the PE logic is:

1.	The AP (arithmetic processor)	6,000 gates,
2.	Basic PMU (control unit)	1,000 gates,
3.	Queue between PMU and AP	1,000 gates,
4.	Parentheses control and vector/ matrix mechanism	1,000 gates,
5.	Instruction decoder and controller	1,000 gates,
	Total	10,000 gates.

These 10,000 gates are placed on two 3-inch diameter LSI chips and housed along with ten other chips in an 8-inch cube occupying 0.5 cubic feet. It is also estimated that the production cost of the PE will be about $600. Rather unbelievable?

If this design is achievable at this cost, or even at 100 times this cost, then it is going to be the biggest breakthrough in computer hardware development since the transistor. In order to achieve the maximum benefit from this new development, many of the programming aids, such as very powerful operators and extensive debugging features that were previously too expensive to implement will now have to be included in the design. Otherwise the AADC PE will be almost immediately replaced with another computer containing these extra programming aids.

This section would not be complete without some comment on the feasibility and current status of the PE. At present LSI 1-1/2-inch diameter chips with 1000 to 1500 gates are being produced at a cost of about $1000 each. The set-up costs, including drawing all the circuits, is about $50,000 for each different type of chip. (Ref. Dr. Ray N. Nilsen, University of California, Los Angeles). Also the CPU for the SUE computer - a small scale microprogrammed computer - is built on two LSI chips and costs less than $1000.

Although this section is written as though the PE actually exists, it must be realized that it is based on design specifications only and that even these are still under development. The information in this section is based almost exclusively on Raytheon's report [1.39].

1.6.6 Introduction and Summary for Master Executive Control

Chapter Six discusses the design of the executive system, or operating system, for the AADC. The Master Executive Control, or MEC as it is called, provides the control and supervision of all the AADC modules. The chapter includes design philosophy, design tradeoffs, MEC capabilities, operating characteristics, MEC evaluation criteria and methods of implementing MEC functions - including sample English language flowcharts. The chapter is based primarily on a design report by Honeywell [1.40]. The subsection is, in fact, a shortened version of the first section of Chapter Six.

Honeywell's report evaluates three possible MECs: a special purpose hardware MEC, a dedicated processor software MEC and a floating software MEC* - on each of four AADC architectures - including the AADC Baseline Architecture, the Time Division Multiplexed Block Transfer Multiprocessor, the Multiple Memory Multiprocessor and the Optimized Simplex Processor. As a result of flowcharting, timing and evaluating each MEC implementation on each applicable architecture, Honeywell recommended the hardware MEC for the Baseline and MMM architectures, the floating software MEC for the TDM Block Transfer Multiprocessor and the dedicated software MEC for the Optimized Simplex Processor. Actually the last recommendation is a violation of the OSP concept, since by definition the OSP contains only one Processing Element.

*The floating Software MEC is an operating system in software which runs on any available PE on a as-required basis, rather than on a dedicated PE.

The method of evaluating each MEC implementation - on each architecture is particularly interesting, for example from [1.40]:

> In order to effectively evaluate the MEC implementations studied, a list of attributes was formulated. Each attribute was assigned a weight corresponding to its assumed relative importance. For each system configuration, a table was constructed and the candidate implementations were scored for each attribute. From these tables a weighted sum for each implementation was obtained. This weighted sum is a measure of the efficiency of the implementation method when used in the particular system for which the table was constructed.

In the Baseline and Multiple Memory Multiprocessor systems the special purpose Hardware MEC is recommended, largely due to its speed advantage, a factor about four to one over the dedicated software, and eleven to one over the floating software in the baseline system. The speed advantage is obtained primarily from the use of an associative memory for very fast table look-up. Since the hardware MEC is specifically designed to accomplish MEC functions, its complexity is considerably less than a general purpose Processing Element. This infers that a special purpose executive should have cost, reliability, size, weight and power advantages over the use of an entire processor to accomplish the MEC functions. If a large enough quantity of special purpose hardware executives are built, they have the potential of being less expensive than a system processor dedicated as the executive. Finally, a special purpose executive can be made more reliable than the proposed system processors.

The floating software MEC implementation is recommended for the Time Division Multiplexed Block Transfer system primarily because of graceful degradation, cost and the other related attributes of size, weight and power. The

floating software is an ideal MEC implementation in a system which does not require a heavy executive load. The overhead time required for a floating software MEC is quite formidable and greatly affects the computation time of some executive functions. The required storage of a MEC kernel in one processor at all times also places a restriction on the size of some program modules.

The Dedicated software MEC implementation is recommended for the Optimized Simplex system due to its characteristics in almost every attribute, especially reliability, graceful degradation, speed and constraints on the rest of the system. Those appear to warrant the cost of the additional processor.

In general, a floating software executive has high overhead requirements and should only be used in a system with low executive function load. A 4096 word task memory should be sufficient for all software executive requirements. A software executive requires each Processing Element to contain a real time clock and a loop counter.

Chapter Six considers four combinations of MEC implementations and AADC architectures. The first is the hardware MEC for the AADC Baseline architecture. The second is the Floating Software for the Baseline system, which is the same as the floating software on the Time Division Multiplexed Block Transfer Multiprocessor. The third combination is the dedicated software MEC on a "Optimized Simplex" system, while the fourth is the floating software MEC on a true Optimized Simplex Processor. Each section contains a description of the applicable hardware, a list of the MEC functions, operation of the system under the MEC control. A description of the MEC and a summary flowchart of the MEC implementation. Also included in Chapter Six is an evaluation of each MEC

implementation on each architecture - including the author's critique of the evaluation method -, and some recommendations for further MEC studies and further development of this course material.

1.6.7 Introduction and Summary to Signal Processing Element

Whereas the PE described in Chapter 5 is designed to fulfill all the sequential processing requirements, the parallel processor is designed to handle all the parallel processing requirements for AADC. The avionic parallel processing requirements include signal processing, radar processing, multiple tracking, pattern recognition, table look-up, optimal filtering signal correlation, Fourier analysis and synthesis, analog test function generation, voice command interface, etc. Parallel processing requirements are for 70 to 133 MIPS and 32K to 100K words of memory.

Although the parallel processor was one of the first AADC areas of concern and it has undergone more changes in design concept than any other AADC module, it still is the module whose design is the least firm and may be subject to further change. Already the parallel processor has been referred to as the Bulk Parallel Processor (BPP), Matrix Parallel Processor (MPP), Associative Processor (AP), General Purpose Array Processor (GPAP), and the Signal Processing Element (SPE).

The feasibility of constructing a parallel processor capable of 150 MIPS throughput is not in doubt, but what will it cost, and how should it be designed to maximize the throughput, maximize the flexibility and minimize the cost? ILLIAC IV and PEPE are examples of very powerful parallel processors that are already in operation but have limited applications.

The major part of Chapter Seven is a description of the Signal Processing Element under development at NRL.

1.6.8 Introduction and Summary to Evaluating AADC Developments

Although a means of evaluating the development of AADC and accurately predicting the performance, cost and reliability is of the utmost importance, relatively little has been published on this specific subject. There are several means of evaluating the development, including:

1. Measuring the load on existing avionic computer and thereby projecting the future requirements,
2. Simulating the operation of individual AADC modules,
3. Simulating the module interaction or the overall AADC operation,
4. Simulating an application using the AADC system,
5. Modeling the operation of AADC modules,
6. Breadboarding at the PE, memory and bussing level (equivalent to CPU, memory and channel level in more common terminology),
7. Devising a test plan for the breadboard of the model including what to measure, how to measure and how to interpret the results, and finally,
8. Producing a prototype of individual modules for testing the complete AADC system.

According to the author's count, there is one completed study on measuring the load on existing avionic computers (but there must be others). (The AADC is currently sponsoring advanced analytical studies with Grumman Aerospace and LTV Corporations examining the computer requirements for the F-14 and A-7 class aircrafts.)

The author also counts three studies simulating AADC modules (case 2 above) and two reports on the simulation of module interaction (case 3 above), and two reports on simulating the AADC application to a particular problem area (Case 4 above). There are also three reports on other facets of evaluating the AADC. One of the current projects is to obtain an Optimized Simplex Processor breadboard or Advanced Development Model.

There are also plans in 1973 fiscal year for completing the PE and SPE register-level simulations, assembling a SPE breadboard, procuring verification hardware for PE and I/O, and procuring feasibility model for both the ferroacoustic and the semiconductor BORAM memories [1.34, page 17].

Therefore, the low number of reports in this area is probably not an indication of the lack of activity; but rather an indication that evaluation studies are being reported along with the particular subsystems.

1.6.9 Introduction and Summary to High Order Language

Chapter Nine presents the developments in defining and producing a very powerful High Order Language that can effectively and efficiently use the AADC System - one that can significantly reduce the development, documentation and maintenance costs of the AADC Software.

For the purpose of this report, a "High Order Language (HOL)" is defined as a language with many powerful extensions beyond those in the present high level languages, such as Fortran, Algol and PL/I. The HOL must be capable of generating efficient executive, I/O, test, display, file, data

manipulating programs. Also it must have powerful vector, matrix, list, character and bit manipulating features. (Although the equivalent of these features can be obtained in present languages they are not easily programmed and do not execute efficiently.) For example, CMS-2 (the Navy's Compiler Monitor System) is an attempt at defining a HOL. CMS-2 is designed especially for real time command and control applications and has the ability to define executive functions in Algol-like subroutines and reorganize data structures at run time.

Two conferences have been held on the HOL for AADC; one in June 1970 and the other in February 1972. The second conference was a good introduction to AADC for software specialists but did not present any concrete proposals for the design of a HOL for AADC. (The conference proceedings are not yet available.) Three papers have been written on the updating of CMS-2 to the AADC HOL, and one paper was written on how MTACCS (Marine Tactical Air Command and Control System) requirements should affect the CMS-3 (extended CMS-2) requirements. Currently there is a project to define the goals of the HOL more precisely.

This is one of the first times that the software specialist has had a chance to influence the design of the hardware. How about some suggestions?

1.6.10 Introduction and Summary to Applications for AADC

Although this is the most important chapter in the report, it is, unfortunately, one of the shortest. Never before has the Navy known so far in advance what the future Navy computers will be, and now the Navy has an opportunity to develop application programs while the computer is being developed, instead of after it is produced and delivered. Equally important, the Navy now has the

opportunity of allowing the applications to influence the software design, which in turn can influence the hardware design. If the Navy can develop an applications-oriented computer and have the application programs ready when the hardware is delievered, the Navy will have made another major step in solving its computer oriented problems.

Chapter Ten presents references to an E-2B aircraft simulation study, the requirements for MINCOMS (Multiple Interior Communication Systems for aircraft), and the On-board, checkout and system interface requirements for the F-14C. Also presented is the proposed Automated Design Facility (ADF) which is designed to provide automatic configuration and checkout of AADC for a new application.

This section has presented an overview of the AADC System by presenting the introductory and summary subsection to each chapter. It has not included the latest developments as reported at the AADC 1973 Symposium, although the major results from the symposium are presented in Subsection 1.4.4. For more details the reader is referred to the section on current status in each chapter, or to the conference proceeding when they become available [1.41].

1.7 CONCLUSIONS

While completing this report the following possible research projects or thesis topics were identified (many others undoubtedly exist):

1. Expand the design of AADC to include multiplatform and ground based systems. This implies virtual memory, multiprogramming, security of storage and interfaces to commercial input/output equipment (disks, CR, LP, etc.). Some of this has already been done (see Chapter 3) but there is still a lot more to do.

2. Simulate several parallel processor configurations and compare their operation on various applications. (Some of this has already been done at NRL, see Chapter 7.)

3. Prepare a concise list of PE features and their implications on the HOL and POL (Problem Oriented Languages).

4. Simulate PE features in such a way to assist in HOL development (coordinate with Bruce Wald at NRL).

5. Evaluate the proposals from industry on defining HOL primitives for AADC. What criticism or improvements can be suggested?

6. Define the HOL constructs that would simplify the writing, debugging, documenting and updating of real-time, scientific and data processing application programs. Repeat this for executive, I/O, test, display and data organizational programs, and then determine which can be implemented effectively on the AADC.

7. Develop a manual on User Characteristics of AADC. This would be a preliminary step to developing applications for AADC. (In some ways this report is a start in this direction, but it is too long and too technical.)

8. Develop new airborne computer applications using AADC features.

9. Develop managerial guidelines for the use of AADC in Navy systems.
   - What are its features?
   - What applications take advantage of these features?
   - How to use AADC to maximize its benefits.

The following conclusions are taken from [1.2]:

> The Advanced Avionic Digital Computer represents the collected effort of an audacious segment of the American computer, technology and aerospace community. More than twenty companies and universities, as well as many Navy laboratories, have held contracts on AADC; many as a result of rigorous competition. As such, it is doubtful that the expertise required to bring AADC to fruition exists under a single roof, except for one that extends from coast to coast.
>
> In a sense, the AADC program will serve to test a new management and procurement philosophy. The idea of competetive bidding on a major development effort, and the subsequent aware of multiple contracts is, of course, not new. What is different, is that these methods have proven <u>necessary</u> for a program involving exploration development and basic research. What must also be appreciated is the willingness of organizations to coordinate and exchange ideas even before these ideas are fully protected. In this manner, the customary delay which precedes the introduction of new inventions is eliminated, allowing a two to five year acceleration of system integration and application. This is especially crucial when these delays may very well approach the life-cycle profitability of such inventions.
>
> In order to avoid the twin dilemma of suboptimization and rapid obsolescence, AADC has been conceived as a system which can, when the time arises, be readily translated into newer technology with minimal impact on its physical, electrical and functional characteristics. By building the computer in this manner, system design experience gained over a longer period of useful years will allow highly refined applications of AADC to evolve. These considerations, along with everything else this report has addressed, make AADC a major and truly revolutionary development.

References to AADC Introduction and Summary

1.1 The Advanced Avionic Digital Computer System; R. S. Entner; Computer Design; Vol. 9, No. 9; September 1970; pp 73-76; (49, NPS)*.

1.2 The Advanced Avionics Digital Computer Revisited; R. S. Entner; Naval Air Systems Command; October 12, 1971; Unpublished paper; Unclassified; (NPS).

1.3 Navy Engineers Break the Rules with Radical Airborne EDP Concept; Electronics; August 3, 1970; pp 89-90; Unclassified; (35, NPS).

1.4 New Airborne Computer Concepts Evolve; Aviation Week & Space Technology; June 22, 1970; pp 217-219; Unclassified; (33, NPS).

1.5 Navy to Unveil Integrated Avionics Plan; Phillips J. Klass; Aviation Week and Space Technology; June 28, 1971; pp 51-53; (NPS).

1.6 Study of Future Requirements for Naval Airborne Computers (U); Hughes Aircraft Company Report No. B5939 ASD 85169R; June 1968; NAVAIRSYSCOM Contract AIR-5333B5-67-2; Confidential-Proprietary; (1).

1.7 Airborne Computer Study - Final Report; Honeywell Inc., No. SRM80; November 4, 1968; NAVAIRSYSCOM Contract AIR-5333B5-68-2; Unclassified; (2).

---

* The number in parantheses at the end of each reference refers to the sequential reference number assigned in the AADC Bibliography in Enclosure 1 of the tenth AADC Progress Report [1.34]. NPS indicates the report is available at the Naval Postgraduate School.

1.8 Minutes of Qualified Sources Conference on Advanced Microelectronic Packaging Concepts and the Advanced Avionic Digital Computer Development Program; Naval Air Systems Command, Code AIR-52022; February 27, 1969; Unclassified; (5).

1.9 The Advanced Avionic Digital Computer; R. S. Entner; Parallel Processor Systems, Technologies and Applications; Spartan Books; Copyright 1970; pp 203-214; Unclassified; (46, NPS).

1.10 AADC Baseline Definition; R. S. Entner; NAVAIRSYSCOM Code 5333F4; July 23, 1969; Unclassified; (11, NPS).

~~1.11 AADC Baseline Definition; R. S. Entner; NAVAIRSYSCOM Code 5333F4; July 23, 1969; Unclassified; (11, NPS).~~

1.12 Proceeding of AADC Conference, September 15, 1969; (include the 4 papers referenced as [1.13 to 1.16]); Unclassified; (NPS).

1.13 Presentation on Advanced Avionics Digital Computer Concepts; Francis J. Leuking, NAVAIRSYSCOM, Code 360F; September 15, 1969; Unclassified; (NPS).

1.14 Presentation of Advanced Avionic Digital Computer Baseline Definition; Ronald S. Entner; NAVAIRSYSCOM; September 15, 1969; Unclassified; (14, NPS).

1.15 Advanced Avionic Digital Computer Hardware Considerations; A. David Klein; NAVAIRSYSCOM; September 1969; Unclassified; (13, NPS).

1.16 Baseline Associative Processor; John E. Shore and Frank A. Polkinghorn, NRL; March 14, 1969; Unclassified; (6, NPS).

1.17 Master Executive Control Techniques for AADC System Final Report; Honeywell Inc., No. 14206-FR; July 1969; NAVAIRSYSCOM Contract AIR-5333F4-69-1; Unclassified; (10).

1.18 AADC Master Executive Control, Baseline Definition; R. S. Entner, NAVAIR-SYSCOM and J. Stepenosky, NAVAIRDEVCEN; December 22, 1969; Unclassified-NOFORN; (19).

1.19 Advanced Airborne Digital Computer Research and Development Program (U); Grumman Aerospace Corp.; July 1969; for NAVAIRSYSCOM, Code AIR-5333F4; Confidential-Proprietary; (9).

1.20 AADC Computer Study (U); General Electric Company Report No. 224-2709; August 22, 1969; NAVAIRSYSCOM Contract AIR-5333F4-69-1; Confidential-Proprietary; (12).

1.21 Advanced Avionics Integrated Digital System Study (U); Raytheon Company Report No. BR5535-1; October 1969; NAVAIRSYSCOM Contract N00019-69-C-0444; Confidential; AD-511-223L; (15).

1.22 Advanced Avionics Integrated Digital System Study (U); Raytheon Company Report No. S-BR-5535-2; October 1969; NAVAIRSYSCOM Contract N00019-69-C-0444; Secret; AD-511-166L; (16).

1.23 Proposed Technical Approach/Advanced Avionic Digital Computer (U); NAVAIR-SYSCOM; January 1970; Unclassified - Four Official Use Only; (20).

1.24 Advanced Avionics Digital Computer Simulation Model, Preliminary Report (U); Univac Advanced Systems Group; November 12, 1969; Unclassified-Proprietary; (17).

1.25 AADC Development Program Progress Report No. 1; R. S. Entner; NAVAIRSYSCOM Code AIR-5333B55; November 12, 1968; Unclassified; (3, NPS).

1.26 AADC Development Program Progress Report No. 2; R. S. Entner; NAVAIRSYSCOM Code 5333B55; February 14, 1969; Unclassified; (4, NPS).

1.27 AADC Development Program Progress Report No. 3; R. S. Entner; NAVAIRSYSCOM Code 5333F4; May 19, 1969; Unclassified; AD-729-666; (7, NPS).

1.28 AADC Development Program Progress Report No. 4; R. S. Entner; NAVAIRSYSCOM Code 5333F4; December 1, 1969; Unclassified; AD-727-603; (18, NPS).

1.29 AADC Development Program Progress Report No. 5; R. S. Entner; NAVAIRSYSCOM Code 5333F4; March 16, 1970; Unclassified; AD-729-667; (28, NPS).

1.30 AADC Development Program Progress Report No. 6; R. S. Entner; NAVAIRSYSCOM Code 5333F4; August 31, 1970; Unclassified; AD-729-668; (37, NPS).

1.31 AADC Development Program Progress Report No. 7; R. S. Entner; NAVAIRSYSCOM; February 4, 1971; AD-727-605; (57, NPS).

1.32 AADC Development Program Progress Report No. 8; R. S. Entner; NAVAIRSYSCOM; July 1, 1971; AD-727-607; (58, NPS).

1.33 AADC Development Program Progress Report No. 9; R. S. Entner; NAVAIRSYSCOM; November 1, 1971; (67, NPS).

1.34 AADC Development Program Progress Report No. 10; R. S. Entner; NAVAIR-SYSCOM; May 31, 1972; (78, NPS).

1.35 High Level Aerospace Computer Programming Language Conference Proceeding; R. S. Entner; NAVAIRSYSCOM; June 29-30, 1970; Unclassified; AD-733-454; (31, NPS).

1.36 Symposium on Advanced Aircraft Electric Systems (SOSTEL) Proceedings; Leonard W. Wendling, NAVAIRSYSCOM; April 20-22, 1971; (NPS).

1.37 Advanced Digital Technology Conference Proceeding; Vol. 1 and 2; NAVORDLAB; June 8-10, 1971; (NPS).

1.38 National Aerospace Electronics Electronics Conference - NAECON 71; May 17-19, 1971; Available from IEEE Transactions on Aerospace and Electronic Systems, reference 71-C-34 AES; (NPS).

1.39 AADC Arithmetic and Control, Functional Block Diagram, Design, Analytical Study; Raytheon Company, Report No. BR6154; December 1970; NAVAIRDEVCEN Contract N62269-70-C-0210; Unclassified-NOFORN, AD-880-510; (44, NPS).

1.40 AADC Master Executive Control, System Analysis Design Study, Final Report; Honeywell Inc., Report No. 12234-FR; December 1970; NAVAIRDEVCEN Contract N62269-70-C-0314; Unclassified-NOFORN; AD-800-635; Vol. 1: Basic Document; (43, NPS).

1.41 All Application Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceeding not yet available.

Representing a fourth-generation computer in the fullest sense, AADC is a modular computer concept, employing hardware and software building blocks to construct various computer architectures

Appendix 1.1

AADC Initial Developments

The Advanced Avionic Digital Computer System

Ronald S. Entner

Naval Air Systems Command
Washington, D.C.

The Advanced Avionic Digital Computer (AADC) is a programmer-oriented, general-purpose, modular digital computer with special features designed to meet 1975-85 Naval airborne data processing requirements. It will combine many of the most advanced computer hardware and software concepts now under development in the United States.

The AADC is a modular computer, designed to be inexpensively assembled from off-the-shelf large scale integrated (LSI) silicon wafer and advanced magnetic thin-film building blocks. It can be configured as a simple minicomputer, a super-multiprocessor, or anything in between. The cost should be one to two orders of magnitude less than today's state-of-the-art computers. The computers should also be one-tenth the size and weight, and should exhibit remarkable reliability.

Development of the AADC is the result of analyses into next-generation Naval aircraft computing requirements, as well as a serious attempt to find ways to reduce the enormous cost of computer procurement and support through the application of standardization. In the past, the designs of computers that were developed by private industry were frequently so different from one another that even system evaluation by qualified engineers was often extremely difficult. Ironically, when some measure of design commonality was found, it was usually attributable to state-of-the-art constraints rather than to a singleness of mind.

To some extent, the objective of commonality could be achieved by bringing the Navy into each contractor's development loop. This would be accomplished by providing Industry with access to pertinent Navy planning documents, as well as to advanced subsystem specifications. While this policy is currently being pursued to the greatest extent possible, security considerations, as well as the proprietary nature of most advanced subsystem development work, place severe restrictions on the procedure. Furthermore, the fact that a vendor is aware of a projected Navy need is no guarantee that he will attempt to satisfy that need unless a respectable profit is in the offing.

To insure the availability of an adequate digital computer for the years between 1975 and -85, the Naval Air Systems Command decided in the fall of 1968 to pursue an active computer development effort, namely the Advanced Avionic Digital Computer Program. The ultimate success of this development will hinge on several basic engineering and management decisions made that year. First, equal emphasis would be placed on system hardware, software, and technology development. Second, no one company would be permitted to develop the computer; rather, jobs would be parceled out on the basis of vendor competance in each critical area, and only after open competition. Third, dependence upon proprietary designs and concepts would be minimized.

Ronald S. Entner is system architect and project manager for the Advanced Avionic Digital Computer Program of the Naval Air Systems Command. Since joining NASC in 1966 he has worked on radar system development, weapon systems analysis, and computer system analysis and design. He received a BSEE degree from the Polytechnic Institute of Brooklyn.

Primary Goals

The AADC program contains several basic objectives.

• **Building Block Construction:** *Develop a family of functional modules* which will take maximum advantage of rapidly improving LSI semiconductor technology. The availability of off-the-shelf building block modules will greatly reduce the time and cost for custom computer design, fabrication, and support when compared with current practices.

• **Modular Organization:** *Develop a general-purpose digital computer architecture* employing a minimum number of unique building block modules, which may then be fabricated in large quantities. Development costs can, in this manner, be amortized over several computer procurements. The alternative to this approach is the development of unique circuits and LSI modules for each new computer requirement. However, since the cost of design and development of each new module may greatly exceed fabrication costs, little or no savings may be realized.

• **Bulk Parallel Processing:** *Include the capability of operating on extremely large quantities of data in real time.* This capability results in a machine with an effective processing rate of billions of operations per second and allows the AADC to function in both the time and frequency domains.

• **Microprogram Application:** *Permit dynamic reconfiguration of each computer's control structure*, thereby providing a better match between problem and machine. It is further anticipated that some measure of inventory computer emulation will be feasible in the microprogrammed processor.

• **Program Modularity:** *Enable the use of large macroroutines and standard program packages (pages)*, thereby reducing the severity of problems associated with the preparation and maintenance of object code. Ultimately, the idea of program modularity will become an integral element of an automated design facility (ADF) which will have the capacity to turn an operational requirement into operating hardware (and software) in a matter of days and weeks as opposed to the traditional months and years.

• **Graceful Degradation:** *Provide the highest level of system reliability concommitant with cost-effective operation.* In large AADC systems, this will amount to failure-tolerant architecture.

The AADC program is, in part, the outgrowth of attempts to establish guidelines for the cost-effective application of LSI technology. To this end, a family of functional and byte-functional modules, or building blocks, is being developed. These modules are general-purpose in nature and flexible enough to meet the challenge of new requirements created by new technology over the AADC's

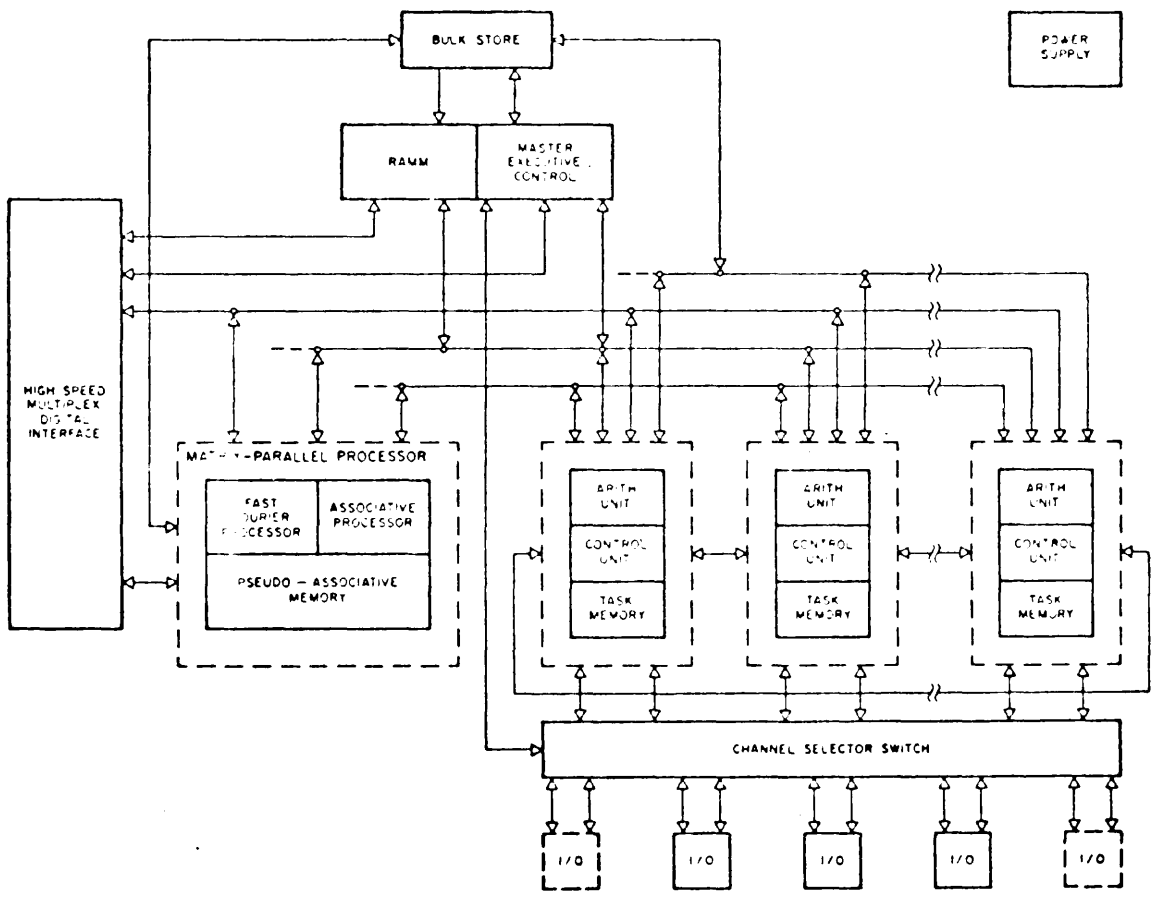


Fig. 1 Baseline organization embodies all hardware elements found in the AADC concept. This architecture combines a sequential time-division multiplexed (TDM) multiprocessor, a bulk-parallel processor, combined hardware and software executive control, a complement of memory hierarchies, and multiplexed and dedicated I/O channels. The baseline system is designed to meet worst-case processing requirements

approximately 10-year life cycle. Functional modularity is used to provide maximum design flexibility by permitting incremental configuration to suit a specific operational requirement and is, therefore, not organization-limited. The same building blocks can be used to construct a general- or special-purpose, central or federated processor or multiprocessor. This wide range of component application is expected to meet most, if not all, airborne data processing requirements for next-generation Naval aircraft, including fighter, intercept, attack, reconnaissance, electronic countermeasure, airborne early warning, antisubmarine warfare, electronic intelligence, transport, and surface-to-air rescue. Among the avionic functions which may be mechanized in one of the various forms of AADC are air-to-air and air-to-ground weapon delivery; inertial, radar, radio, and satellite navigation; radar and acoustic signal processing; target signature recognition; aircraft flight control; sensor monitoring and control; and electronic countermeasure monitoring and control.

The computer organization shown in Figs. 1 and 2 represents the baseline AADC organization. Embodying all hardware elements of the AADC concept, this organization is believed to embrace those qualities needed to meet 1975-85 Naval airborne computer system requirements. The effectiveness of the baseline organization is founded on a functional distinction between sequentially organized problems, such as weapon delivery, navigation, and system test, and parallel organized problems, such as multiple target tracking, sensor correlation, and data compression. The sequential problems are assigned to the processing elements (PEs) shown in the diagrams. Each PE contains sufficient memory (approximately 2K to 4K words) to effectively store and process large routines, or program modules (PMs). This application of paging methodology reduces by several orders of magnitude main memory access conflicts usually associated with multiprocessing. Parallel problems are assigned to the programmable matrix-parallel processor (MPP). This device consists of a combination of fast Fourier processor (FFP), associative or array processor (AP), and an associative or pseudo-associative memory (AM or PAM). These three elements are interconnected by switching logic and are controlled by a PE or an internal microprogrammed controller. Among the tasks that can be assigned to the MPP are radar signal processing, radar beam steering, multisensor correlation, multiple target tracking, optimal filtering, video preprocessing, table lookup, pattern recognition, data correlation, radar and acoustic spectral analysis, analog test signal generation and analysis, and basic voice interface functions.

In addition to the processors, the baseline organization illustrates the application of multiplexed as well as dedicated interface channels. The dedicated channels are coupled, in this case, into the sequential processor (PEs) via a low-frequency crossbar switch. This switch is set at the moment a PE is dedicated to a specific task, or PM—a situation that occurs whenever extremely high PM iteration rates might create undesirable communications traffic jams through the multiplexed input output bus. Fig. 3 illustrates the class of multiplexed communication system with which the AADC is expected to interface. Within this system, data and command transfers occur within allocated time intervals or frequency slots. These allocations are guaranteed against worst-case communications

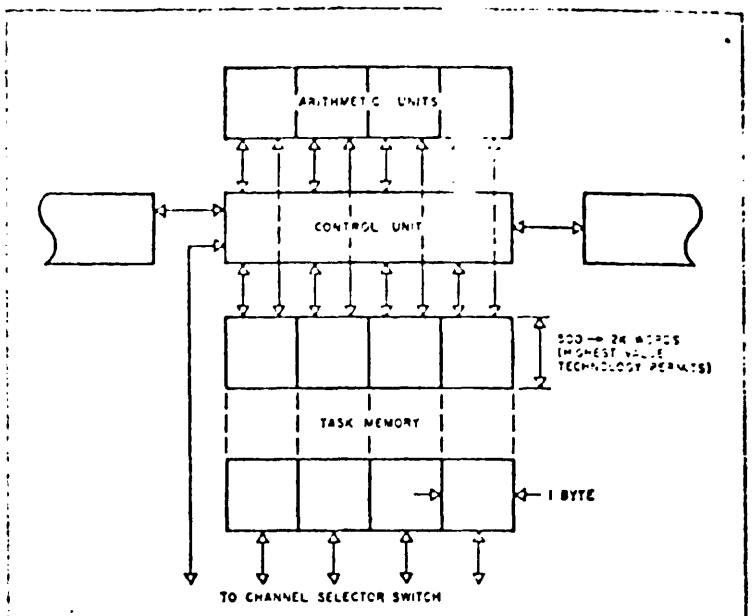


Fig. 2 Processing element (PE) detail illustrates the building block modularity feature of the AADC. The PE arithmetic unit and task memory are byte-functionally modular and are configured after an optimal word length has been determined. The number of words of task memory can also be expanded vertically. The control unit is not partitioned by byte, since the control feature requires full word length to operate

requirements, permitting peripheral equipment to be designed against standard interface specifications. Performance can be optimized, however, by allowing unused time or frequency slots to be passed along to other potential users. Allocation operations are monitored and controlled by the master executive control (MEC).

The functions of the MEC are to provide dynamic control of system resources, perform task queuing in order to optimize resource utilization, initiate and supervise I/O operations, and initiate and supervise system hardware and software reconfiguration in the event of failure. In the baseline organization, the MEC may consist of an area of control logic, an arithmetic unit, a program memory, and an associative status memory. In other AADC architectures, the MEC may be implemented entirely with software, or with some varying combination of hardware and software such as a floating executive with common associative status file.

Also shown in Fig. 1 are the two highest members of AADC on-line memory hierarchy. These are the bulk store and random access main-store memories. As can be seen from the diagram, all processors have access to both memories. For the PEs, this permits the direct access of invariant data and routines from bulk storage in order to reduce the quantity of expensive random access memory required in the system. The 0.1¢/bit ferro-acoustic bulk memory under development for AADC exhibits a 70-ns/word cycle time on a block, or page organized basis, thereby providing approximately a 10:1 cost reduction over an all RAMM implementation with no sacrifice to system throughput. Direct PE to bulk store access also provides graceful degradation in the event of RAMM failure. The MPP requires direct bulk store access due, in part, to the large quantities of data that must be stored and operated on within the MPP.

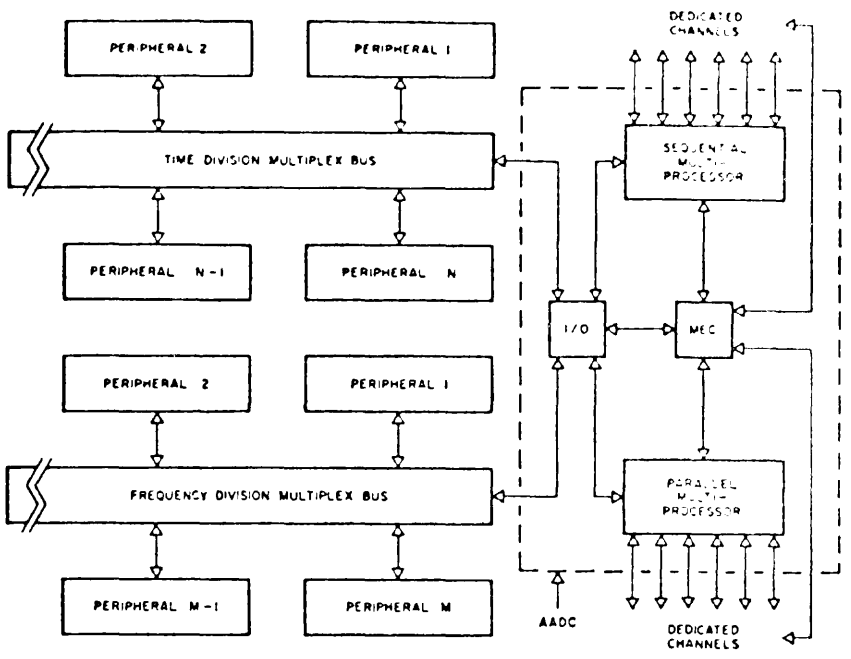


Fig. 3 AADC is expected to operate with a sophisticated multiplexed communications network. This feature will extend executive control into system peripherals and permit tighter regulation of real-time communication resources

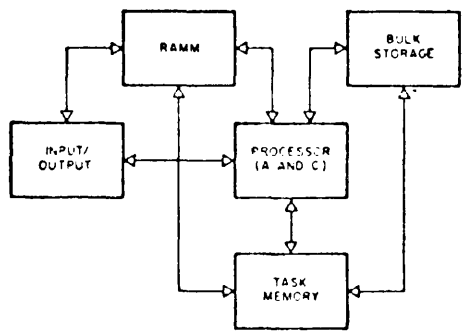


Fig. 4 Optimized simplex processor (OSP) represents the least complex of all AADC architectures. This structure employs three memory systems, each specially suited to provide a paged program capability with no loss of system throughput. This approach is expected to offer nearly an order-of-magnitude cost reduction for avionic computers requiring extensive program storage

Fig. 4 illustrates a particularly interesting variation of the AADC baseline architecture. Termed the optimized simplex processor (OSP), the organization takes advantage of relatively large program-to-data storage ratios (typically 5:1 to 10:1) found in aerospace program listings. This design differs from conventional simplex architecture in that three memories are used in lieu of a single random access main store memory. In operation, invariant program pages are stored and transferred from the block organized bulk store to task memory, followed by pertinent data from the RAMM before process initiation. The task memory then performs the functions of both local storage and scratchpad for the job at hand. Because of special situations, such as large matrix computations, the arithmetic and control structure should have the capability of working directly out of the RAMM as well as task memory. This would minimize inefficiencies created by insufficient workspace within task memory and, hence, excessive data transfers between the RAMM and task memory.

Although not as sophisticated as the baseline organization, the OSP provides a logical first step in an orderly progression toward that complex organization. For this and other reasons, the OSP will likely be the first goal for AADC prototype development.

Summary

The AADC concept has been developed in response to projected Naval airborne digital computer system requirements for the 1975-85 timeframe and beyond. The approach utilizes old and new technologies and methodologies to create a cost-effective, integrated digital computer system capability based on the concepts of functional and byte-functional modularity. This approach will permit the design and fabrication of optimally configured computer systems for each unique operational requirement likely to develop within the addressed timeframe. The time and cost to develop each new member of the AADC family will, on the basis of this discussion, be significantly less than for alternative approaches. The cost and effort required to support AADC equipment, too, will prove to be much less than for any alternative approach currently known to the author. Most significantly, the availability of an "off-the-shelf" computer capability will free essential Naval and Industry systems engineers from the arduous and noninventive task of tracking the development of a major avionic system component, i.e., digital computer, for each and every new aircraft system procured. These persons will then be free to place greater emphasis on the problem of defining dynamic wartime and peacetime total system requirements. The AADC will, in this manner, permit rapid response in an era of sophisticated technological innovation.

Chapter 2

A A D C

A R C H I T E C T U R E S

Table of Contents for AADC Architectures

Appendix

List of Figures for AADC Architecture

List of Tables

Glossory to AADC Architectures

AAGIS - Advanced Avionics Fault Isolation System.

Baseline - The most powerful AADC Architecture - several DPEs and a SPE.

BORAM - Block Oriented Random Access Memory: used to store Program segments.

MEC - Master Executive Control (Chapter 6).

MINCOMS - Multiple Interior Communications Systems: standard I/O interface to the aircraft that is used by the AADC system.

MIPS - Millions of instructions per second: a measure of computer throughput.

MMM - Multiple Memory Multiprocessor: similar to Baseline architecture but no TMs and several RAMMs.

nsec - Nanoseconds equals 10^{-9} seconds.

OSP - Optimized Simplex Processor: simpliest AADC architecture.

PE - Old name for the Data Processing Element (Chapter 5).

PM - Program Module: a portion of a program that contains less than 4K words and execute as a unit.

RAMM - Random Access Main Memory: used to store mode-independent data and buffer I/O.

TDM BDM - Time Divison Multiplexed Block Transfer Multiprocessor: similar to AADC Baseline but uses a software MEC.

TM - Task Memory: a random access memory dedicated to a PE for temporary data and the currently executing PM.

μsec - Microseconds equals 10^{-6} seconds.

DPE - Data Processing Element - new name for the sequential Processing Element (PE).

Chapter 2

AADC ARCHITECTURES

2.1 INTRODUCTION AND SUMMARY

2.1.1 Introduction

Chapter Two describes the AADC architectures from the simpliest processor - called the Optimized Simplex Processor (OSP) - to the most powerful multiprocessor - the AADC Baseline System - and to the new ultra-reliable Three-Plus Processor (TPP) system. This chapter also discusses the interconnections between AADC modules such as internal bussing and external I/O interconnections. Finally this chapter acts as a "catch all" for subjects which do not fit in any other chapter and pertain to the overall system organization or operation. This also includes some directly-executing Higher Order Language architectures which are interesting alternates to AADC.

2.1.2 Summary of Architectures

The basic hardware building blocks of any AADC system are: 1) a Block Oriented Random Access Memory (BORAM) to hold program modules; 2) a Random Access Main Memory (RAMM or RAM) to hold semi-permanent data and to buffer I/O; 3) a small (4K word) Task Memory to hold the currently executing program module and temporary data; 4) Processor Elements (PEs)* to perform the sequential arithmetic computations; 5) an optional Matrix Parallel Processor (MPP) or Signal Processing Element (SPE) to process radar and video signals; 6) one or several Input/Output Units; 7) the internal bussing to interconnect all the modules; and finally, 8) a Master Executive Control to control all the modules and supervise the operation of the entire system.

*PE and the new name DPE - for Data Processing Element - are used interchangably.

The simpliest system is the Optimized Simplex Processor (OSP) with a single PE* with its TM, a RAMM, a BORAM, an I/O unit, internal bussing and a floating software MEC. The PE executes the MEC out of RAMM; this is the only case in which instructions are executed from RAMM. The PE also executes Program Modules out of the Task Memory. The most powerful system is the AADC Baseline system which contains several PEs with their TMs, a large RAMM, a large BORAM, several I/O units, a Signal Processing Element, four internal busses and a hardware MEC.

Between the two extremes, two architectures have been defined. There is a Time Division Multiplexed Block Transfer Multiprocessor (TDM BTM) which is essentially the same as a Baseline system except with a software MEC. There is also a Multiple Memory Multiprocessor (MMM) which has several RAMMs but no TMs. In this case the PEs execute programs directly from the RAMMs.

Since the AADC PE is a very powerful computer capable of executing 3.3 MIPS and relatively inexpensive, it is deemed more important to increase the reliability rather than the throughput. Three extra reliable configurations have been defined. The Dual Processor has two OSP systems each capable of providing complete backup for the other. The Triplex Processor contains three OSP systems with majority gate decision logic sampling their output for added checking of random errors. The ultra-reliable configuration is the Three-Plus Processor which is the same as the TP above, except it has extra PEs that can be switched in automatically in case a PE fails.

2.2 AADC ARCHITECTURES

2.2.1 Optimized Simplex Processor

This section will present the simpliest AADC architecture called the Optimized Simplex Processor (OSP). The Optimized Simples Processor has a single PE (including Arithmetic and Control Unit and a Task Memory), a RAMM, a BORAM and four internal busses as shown in Figure 2.1. The MEC is a floating software MEC that is executed from RAMM.

The basic building modules of the OSP are described as follows:

1. BORAM - A Block Oriented Random Access Memory with a 2 μsec per block access time and a 70 to 150 nsec per word transfer rate. It is non-volatile. It stores all Program Modules (PMs); all MEC software segments, special MEC identification words. A Program Module may be stored in several consecutive BORAM blocks. The BORAM will probably use a ferroacoustic recording technique (Chapter 4).*

2. RAMM - A Random Access Main Memory which is non-volatile and has a 150 nsec per word access time. It is used to hold all mode-independent (or permanent) data and it provides I/O buffering area. In the OSP, the RAMM also holds the MEC program segments while they are being executed. It probably uses a Closed Flux Memory, or CFM, technology (Chapter 4).

3. TM - Task Memory which is part of the PE. It is probably a 4K 36-bit word RAM with 150 nsec per word access time and may be volitile. It is used to store the currently executing PM. It probably uses the same recording technique as the RAMM. In all applications role the TM contains segments or pages of several PMs.

4. PE - Processing Element (or more accurately a A&C for Arithmetic and Control Unit) a general purpose sequential processor capable of executing arithmetic and logical operations at a rate of 3.3 MIPS and having a very powerful instruction set. The PE will run either a PM from TM or MEC segments from RAMM (Chapter 5).

5. I/O Unit - a general purpose interface unit between RAMM and the environment.

*BORAM is still under development.

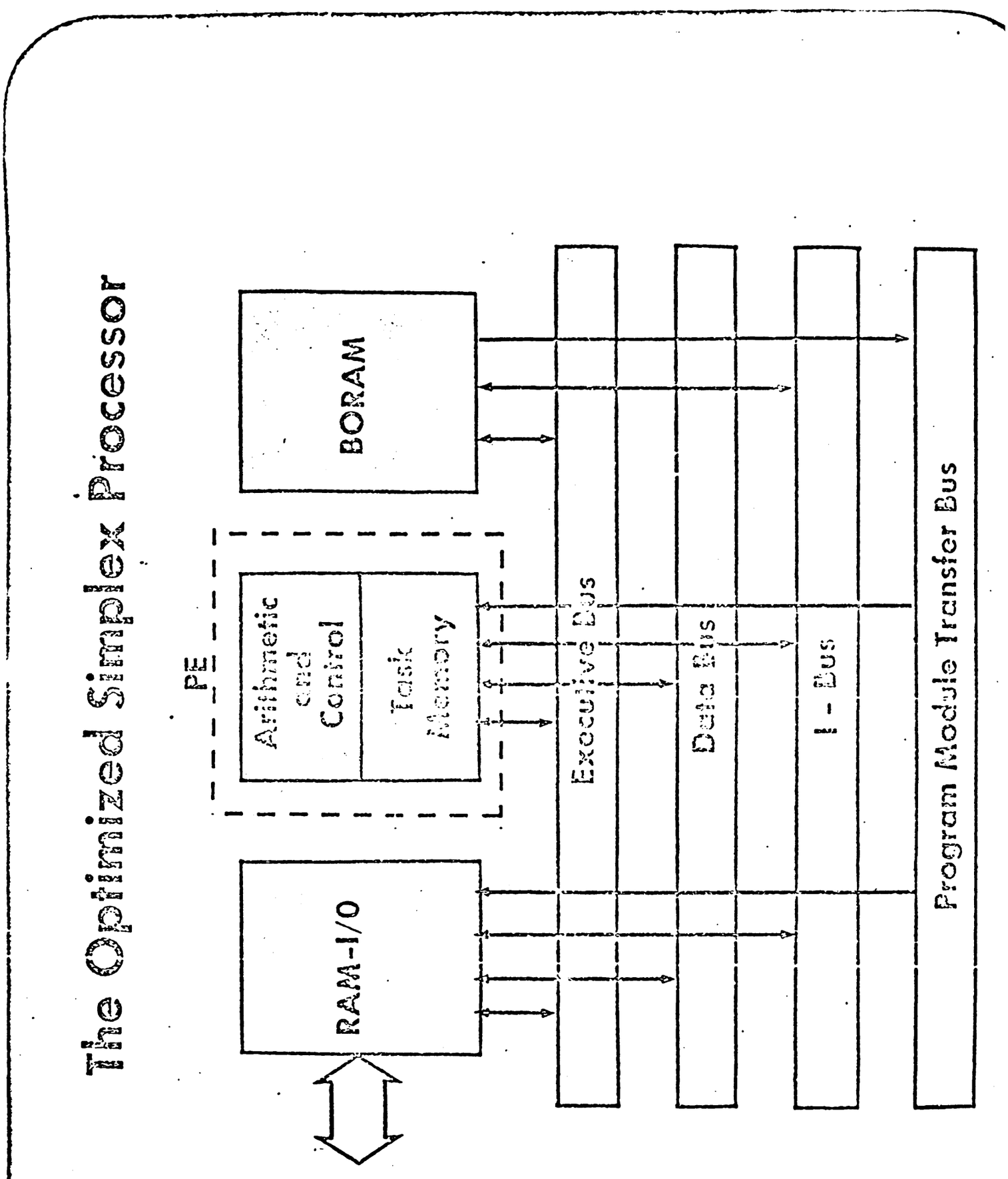


Figure 2.1. Optimized Simplex Processor (OSP)

6. Four internal busses interconnect the modules. The executive bus allows the MEC to control the other modules. The data bus allows the transfer of data and I/O between the PE, TM and the RAMM. The interrupt bus allows an external I/O unit to signal the MEC or allows a PM to request a MEC function. The interrupt bus also provides a means for any module to signal the MEC in case of failure. The program module transfer bus allows the transfer of PMs to TM and MEC segments to RAMM. This unidirection bus allows transfer out of BORAM only.

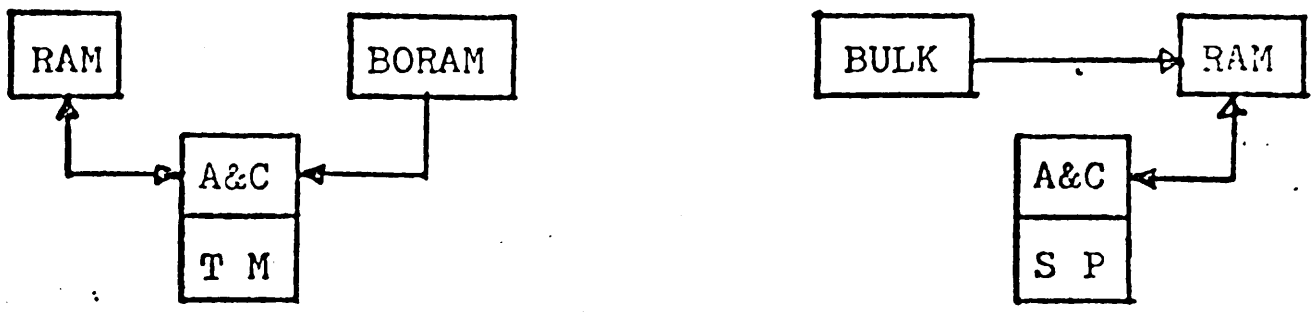
7. MEC - Master Executive Control is a software program that provides the supervision and control of the other modules. It is executed by the PE out of RAMM in the OSP case and thus is often referred to as a floating software MEC (Chapter 6).

The Signal Processing Element is not normally included in the OSP.

A comparison of the OSP with a conventional architecture is shown in Table 2.1. This table also provides information concerning the storage costs and performance of a similar system using conventional architecture and components. The advantage of partitioning procedure and data between RAMM and BORAM, as well as the use of BORAM for primary storage is obvious from the relative cost and speed.

The operation of the OSP is fairly standard with the BORAM providing backup storage for all PMs, MEC segments, and permanent data or descriptor words. A PM is moved from BORAM to TM for execution. A Program Module can issue an instruction to call another PM or to overlay part of itself. MEC segments are moved from BORAM to RAMM for execution with a MEC kernel always residend in RAMM. Output of data is performed by placing the data in RAMM and signaling an external I/O device to remove it. Input of data is recognized by an external interrupt on the interrupt bus and it is then removed from the RAMM buffer area.

TABLE 2.1.COMPARING THE AADC/OSP TO CONVENTIONAL ARCHITECTURE



OSP	CONVENTIONAL
RAM	
6K words, 32 bits, 250 nsec/wd cycle time, - $9,600	64K words, 32 bits, 250 nsec/wd cycle time, 160 nsec/wd access time, - $102,000
BORAM/ BULK	
124K words, 32 bits, 150 nsec/wd cycle time, 2 usec/block access time, 250 - 500 words/block, - $12,000	80K words, 32 bits, 5 usec/wd cycle time, 4 msec/block access time, 3K words/block, - $7,800
TASK MEMORY/ SCRATCH PAD	
4K words, 32 bits, 150 nsec/wd cycle time, 80 nsec/wd access time, - $6,400	128 words, 32 bits, 150 nsec/wd cycle time, 80 nsec/wd access time, - $200
TOTAL STORAGE COST	
- $28,000	- $110,000
THROUGHPUT*	
3.3 MIPS	2 MIPS

* Using Raytheon A&C for AADC with 5 nsec logic

2.2.2 AADC Baseline Architecture

The AADC Baseline Architecture is the most powerful AADC System containing several PEs and a Signal Processing Element as shown in Figure 2.2. Another major difference is that the Baseline Architecture will probably have a hardware MEC, as well as a floating software MEC for backup in case the hardware MEC fails.

The hardware modules of the Baseline System can be described as follows (Figure 2.2):

1. The BORAM, TMs and PEs are the same as for the OSP except there can be several PEs and TMs.

2. The RAMM is the same except it holds only the mode-independent data and the buffered I/O data, and is not used to store MEC segments.

3. Dedicated I/O units are included which can be dedicated to a particular PE having excessive I/O requirements.

4. A Channel Selector Switch has been added which is a programmable digital interconnection network capable of connecting any PE to any dedicated I/O unit.

5. High-Speed Multiplexed Digital Interface has been added which is a programmable sampling network switch (see Figure 2.3) capable of interfacing into the aircraft's MINCOMS (Multiple Interior Communication System). Input data to (and output data from) the AADC system is stored in (read from) the RAMM by this unit. This unit with the RAMM is the I/O for most of the systems communication to the external sensors and actuators.

6. Hardware MEC has been added which is an expanded version of the OSP MEC but implemented in hardware. The MEC has the following functions:

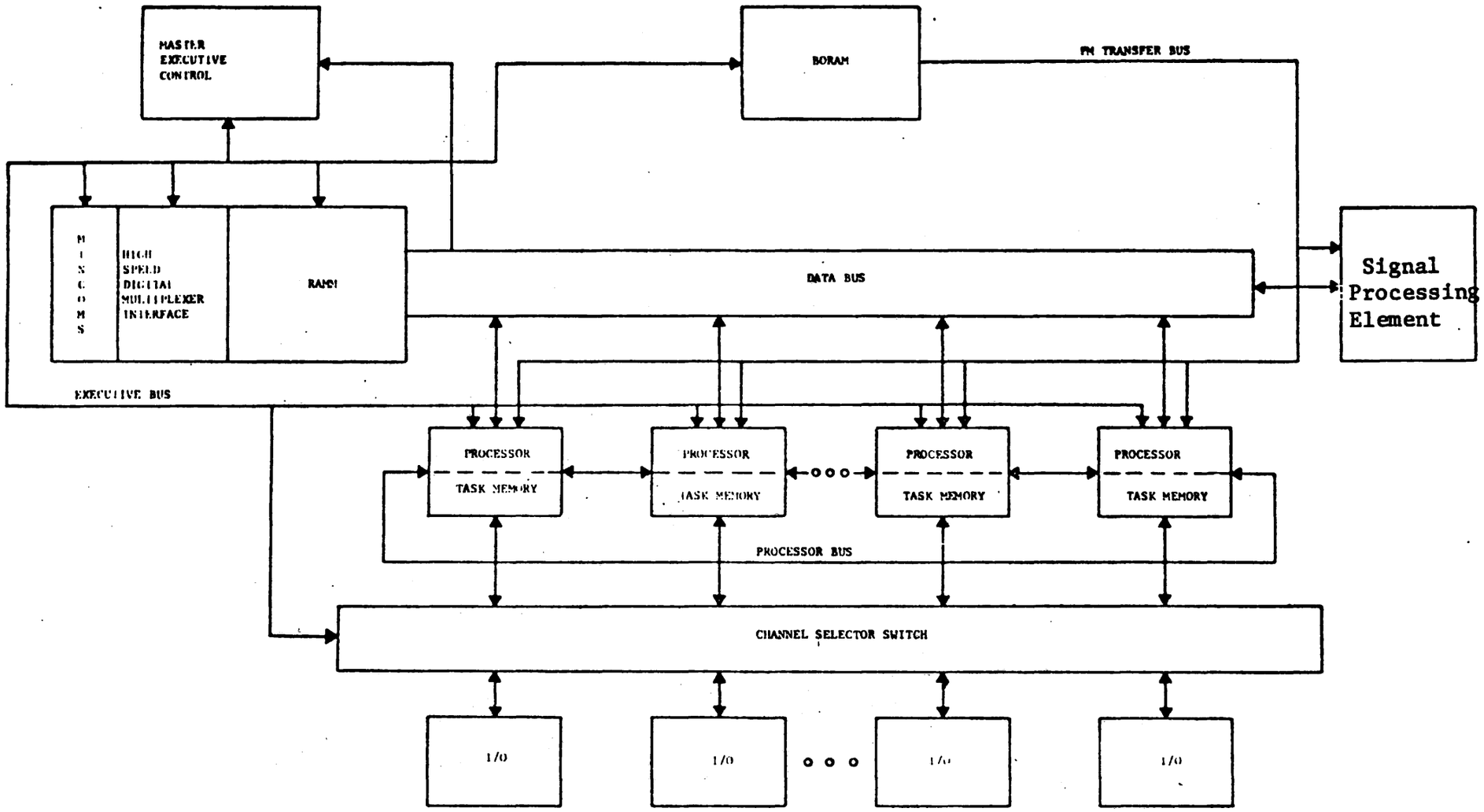


Figure 2.2. Block Diagram of the AADC Baseline System

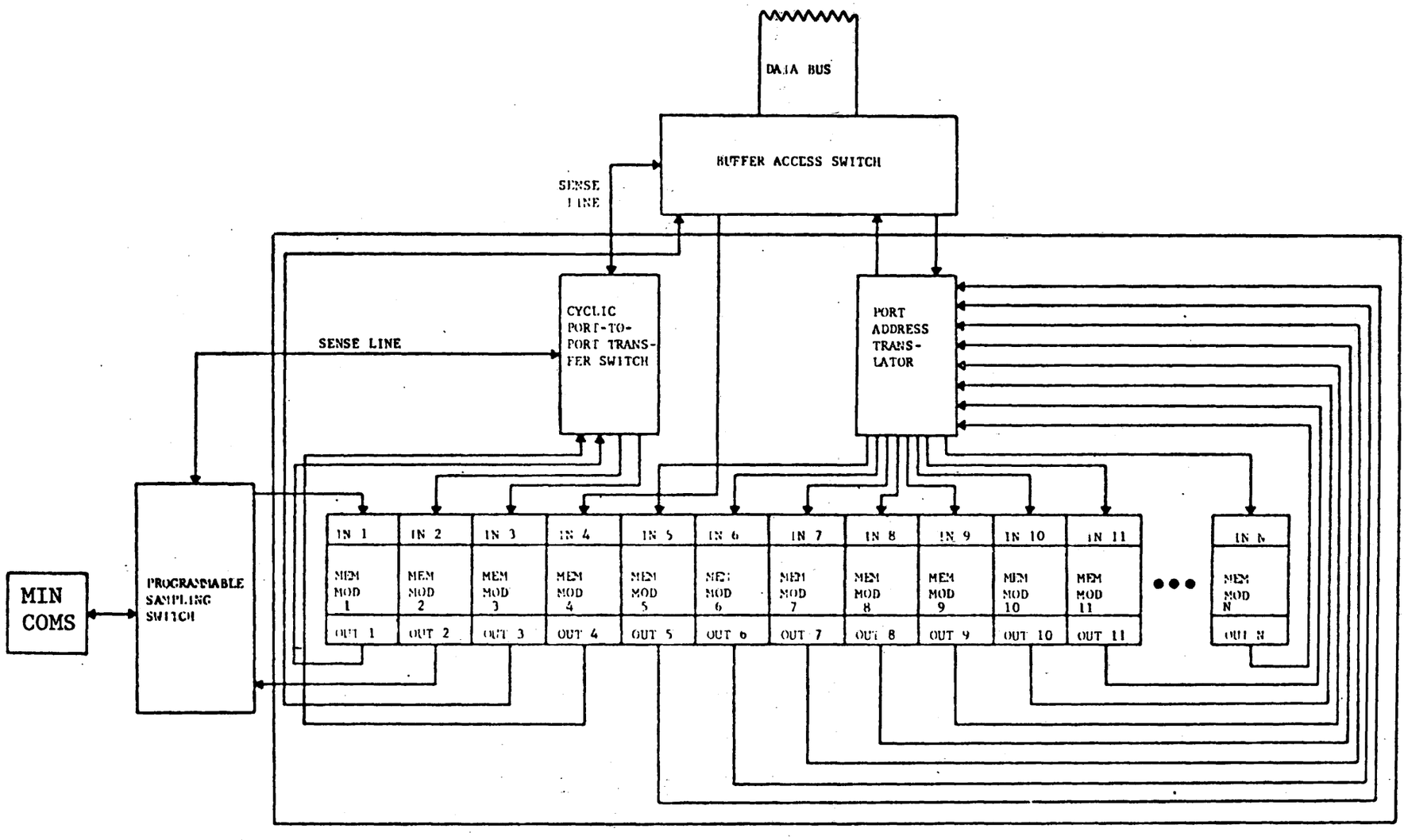


Figure 2.3. Block Diagram of RAMM and External Interfaces

- Monitor the various processing elements in the system to meet the requirements of all (externally-requested) modes of operation of the aircraft.

- Assign operational programs to the various processing units.

- Supervise data transfers between units within the AADC.

- Supervise the overall System operation, for such items as processor failures, interrupt requests, etc.

A floating software MEC will probably also be provided as backup in case of a failure in the hareware MEC in the Baseline System. Further description of the hardware MEC is available in Chapter 6.

The operation of the Baseline System is quite similar to the OSP except a lot more activity can be occuring simultaneously. Several PMs can be executing on different PEs, other PMs can be transferring from BORAM to TMs, and data can be read and written simultaneously. One difference is that the MEC segments are used only for backup and then they are transferred to and executed from the TM of a PE. A more detailed description of the operation will be given in the following subsection.

2.2.3 Time Division Multiplexed Block Transfer Multiprocessor.

The Time Division Multiplexed Block Transfer Multiprocessor (TDM BTM) represents an intermediate architecture between the OSP and the Baseline and is one of the first attempts to solve the classical problem of memory access conflicts. A scheme was devised which combined a small random access memory (or TM) with an Arithmetic and Control (A&C) unit to form a Processing Element (PE). Several PEs are then arrayed on a Time Division Multiplex (TDM) bus and serviced by a

conventional Random Access Mainstore Memory (RAMM). The RAMM, in turn, is tied to a Block Oriented Random Access Memory (BORAM), which provided off line program storage. This relationship is illustrated in Figure 2.4. Note there is no direct connection from BORAM to the TMs in this first version of TDM BTM.

In theory, because each PE had 512 to 4K words of local storage, programs can be block transfered from RAMM to a local Task Memory (TM), thereby reducing the number of PE to RAMM accesses made during program execution by two to three orders of magnitude. This reduction permits various PEs to access RAMM sequentially on a non-interfering, or nearly non-interfering basis, thus eliminating the need for an elaborate crossbar switch between PEs and RAMM for access conflict resolution. The TM provides local storage for data and programs and thus permitted one RAMM to service several PEs. For similar reasons, this organization also eliminates the need for careful partitioning of object code in RAMM, since access conflicts are now resolved in time, not space. This last factor means drastically reduced RAMM size, for RAMM can now be readily reloaded from BORAM on a mode-to-mode basis. In this way, the BORAM inherited the role of primary program storage. The success of this first architecture hinges on:

- the ability to structure aerospace programs from modules (Program Modules or PMs)
- the ability to assign PMs to PEs in a timely and optimal manner
- PM run times which are long when compared with RAMM to PE transfer times
- a substatical cost differential between RAMM and BORAM technologies.

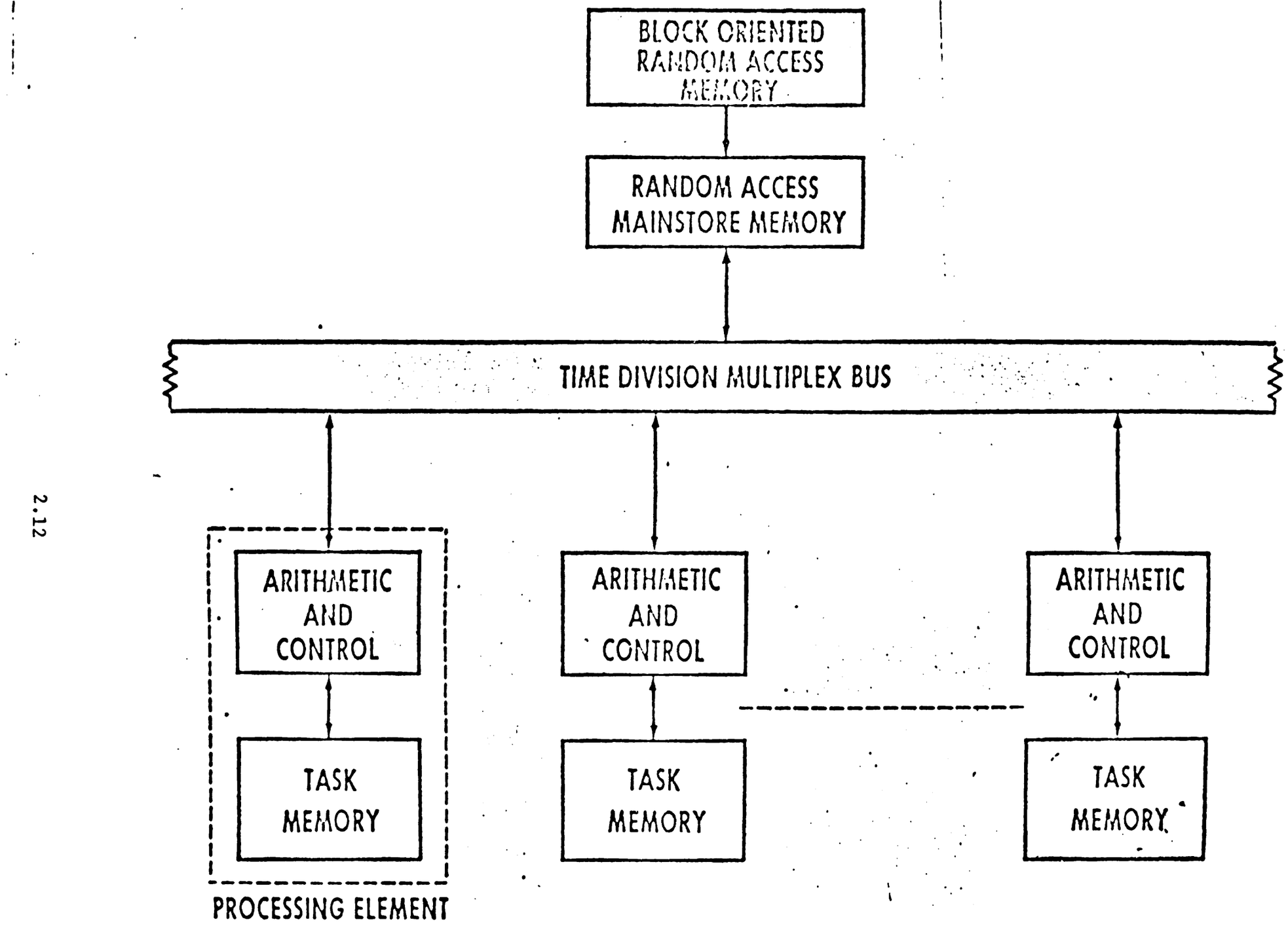


Figure 2.4. Preliminary Multiprocessor Design Concept (Version 1 of TDM BTM)

As a result of early development efforts, various relationships and technologies have evolved. Among the most important are:

- the fact that aerospace programs can, indeed, be modularized, and that these modules can be further partitioned into pages which exhibit useful replacement properties
- critical path analysis techniques can be successfully applied to the PM to PE assignment problem
- ferroacoustic memory technology would allow the fabrication of mass memories which are at least an order of magnitude less expensive, and at the same time an order of magnitude faster than militarized RAM technologies
- 80% to 90% of typical aerospace programs consist of invarient procedure and constants.

As a consequence of these, as well as other findings, an extremely important design change has been made to the original architecture. The BORAM was disconnected from the RAMM and joined directly to each PE through a Program Module Transfer Bus (see Figure 2.5). This alteration permitted further reduction of RAMM size, reduced PM transfer times, and provided immediate processing resources to all PMs without recourse to time consuming roll in procedures.

In the newer architecture and the final version of the TDM Block Transfer Multiprocessor, PMs were stored as pages in the BORAM and transfered to the TM

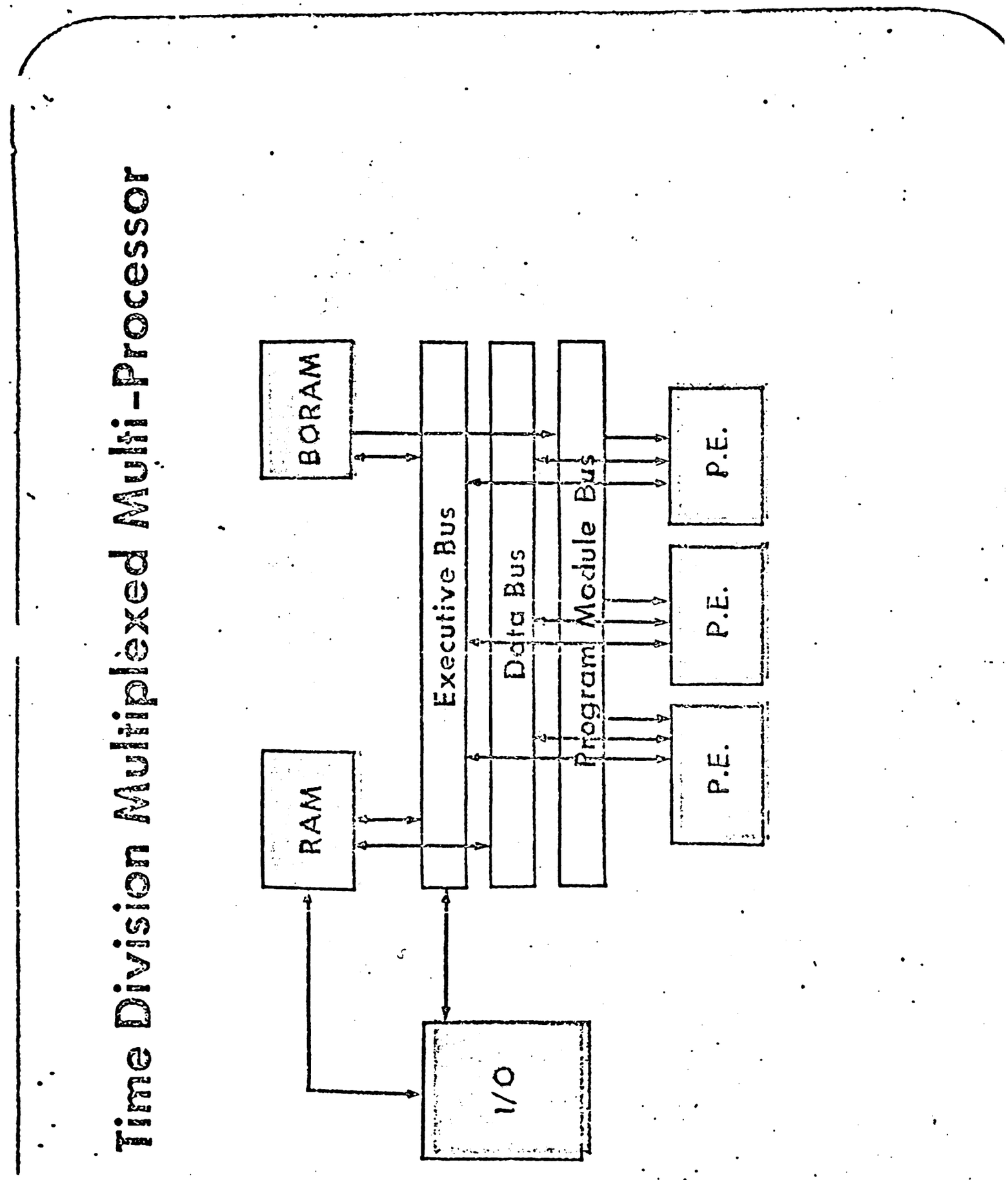


Figure 2.5. Time Division Multiplexed Block Transfer Multiprocessor (TDM BTM)

upon initiation by the AADC operating system, the Master Executive Control (MEC). In practice, these PM pages could vary anywhere from a few score to several thousand words. While loading such pages into PEs on an exclusive basis will result in satisfactory performance, recent computer simulations have shown that a substantial increase in processor and memory utilization can be achieved by multiprogramming each TM. For this reason, present AADC system design calls for one or more PMs to share a PE, with only a portion of the PM resident in TM at any one time. Consequently, each PM, consisting of one or more pages of 128 or 256 words, is stored in BORAM until called by the MEC or called by a page fault within an active PM. The executive bus is added to prevent control conflicts with program and data transfers.

The TDM BTM gets its name from the fact that the busses are time division multiplexed, which means that only one PE has a given bus at a given time, and the fact that programs are transferred as block between BORAM and TM. Note the TDM BTM is somewhat similar to the Baseline architecture when the hardware MEC has failed. (The material in this subsection is taken from Reference [2.1].)

2.2.4 Multiple Memory Multiprocessor

The Multiple Memory Multiprocessor shown in Figure 2.6 is similar to the Baseline except that the PEs have no Task Memories and there are several RAMMs. Thus, the PEs share the RAMMs and execute programs directly from it via a second channel selector switch (which is also programmable similar to channel selector switch on the OSP). The MEC may be hardware or software.

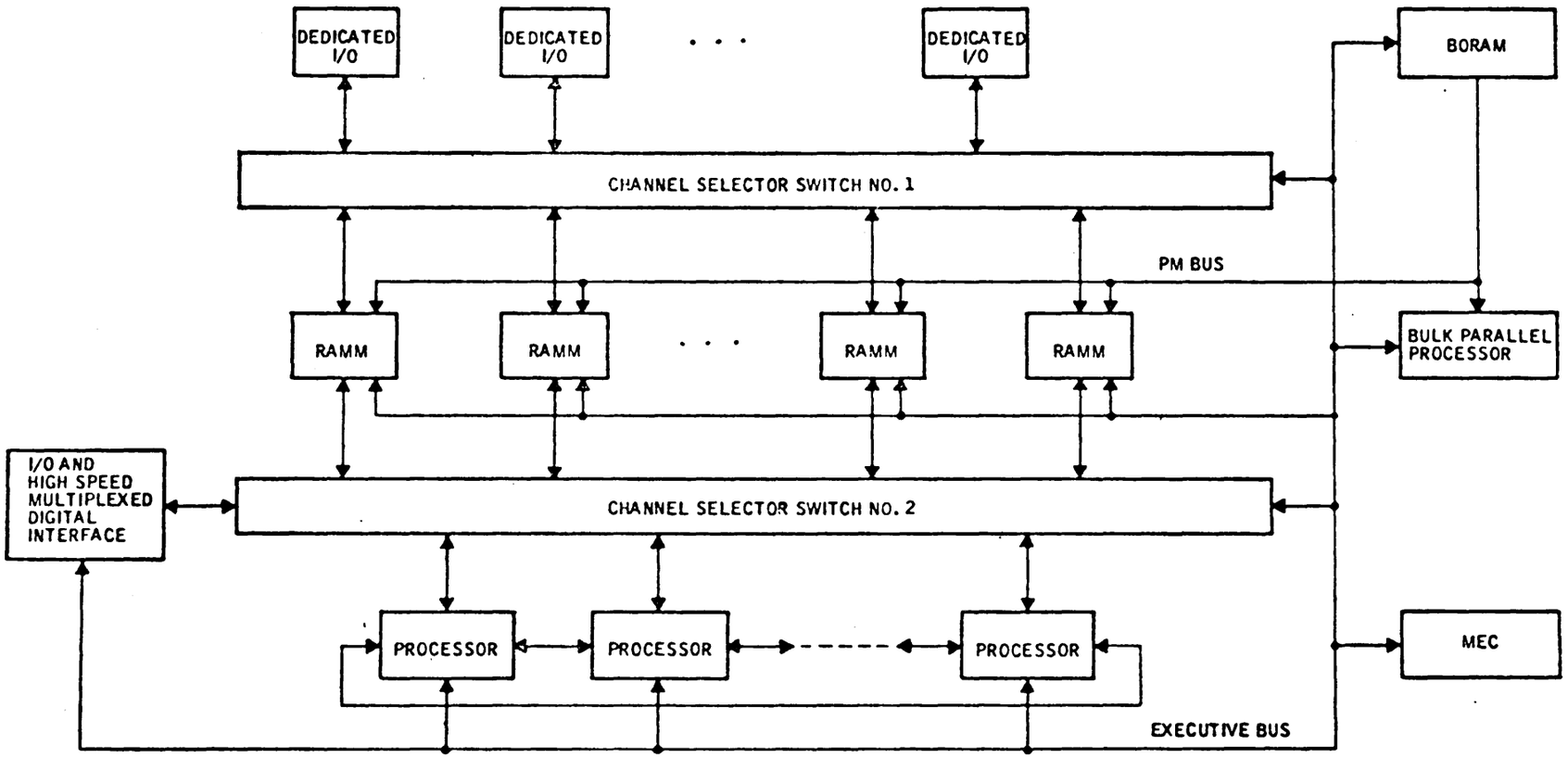


Figure 2.6. Multiple Memory Multiprocessor

This configuration has the advantage of allowing the PEs to execute another program while the next PM is being loaded into RAMM. The disadvantage of this system is that it takes more memory and, since memory modules are probably more expensive than the PEs, the MMM is probably more expensive than the Baseline architecture for the same throughput and reliability.

2.2.5 Ultra-Reliable Architectures

As a result of a recent appreciation for the processing power of the AADC/OSP, an interesting modification was made to existing MEC design goals. Because the unit processor provides the necessary throughput to meet the combined sequential processing requirements of an integrated 1980 aircraft, the need for multiprocessing should arise, when and if it does, from a desire for improved reliability rather than increased computer throughput. Toward this end, three extra classes of multiprocessors have been identified based on the OSP. These are:

1. The Duplex Processor (two PEs): used whenever some processing backup is desired after a PE failure - the MEC kernel stored is stored in the PE performing the applications programs - improved throughput, while provided, is not a fundamental goal;

2. The Triplex Processor (three PEs with/without a hardware MEC): used when solution confidence is all important - provisions for majority voting - executive fallback provided in the event of an MEC hardware failure;

3. The Three-Plus Processor (more than three PEs with/without hardware MEC): provides all the above capability with the addition of power switching for improved, long term, maintenance free reliability.

The present AADC Architectures and MEC design are being updated to accomodate this new applications philosophy [2.2, paragraph 2C].

2.3 INTERFACING AADC MODULES

2.3.1 Internal Bussing

The following excerpt is taken from AADC Progress Report No. 10 dated May 31, 1972 [2.3, paragraph 30, 31] which discusses the need for a very fast and reliable internal bussing scheme for AADC. The busses are described in Subsection 2.2.1 above.

> 30. Internal Bussing: It is a long held belief that communications among AADC components will prove a particularly difficult problem to resolve. This is because of a) the requirement for modular extensibility, b) the high data rates between and among hardware modules, c) the extremely small size of AADC components, even at the second packaging level, d) the need for EMI/EMP immunity, e) the requirement for high reliability, and f) the desireability of low cost. As a result of these concerns, study efforts were initiated to investigate the options afforded by various bus implementation schemes. If anything, these studies confirm the original sense of concern. Fortunately, the problems which still exist do not seem beyond resolution.
>
> 31. In addition to functional analyses, study efforts are proceeding to determine a reasonable technology with which to implement an internal bus subsystem. Feasibility hardware for a low power, multiplexed, optical communication system is presently being assembled for the Navy by IBM.

Reference [2.4] presents the results of IBM's investigation on optical data links. Some details of a possible implementation are presented in Chapter 3 - AADC Hardware Technology.

2.3.2 External I/O

The following excerpt from AADC Progress Report No. 10 [2.3, paragraph 13] describes the current status of the external Input/Output interface developments:

13. External I/O: To be truly useful, the AADC has to interface with other equipments or peripherals in the larger context of an information processing system. A contracted effort to address the design requirements for a general purpose AADC interface is planned. An RFP for this effort has been released by the Naval Air Development Center, Warminster, Pa. Enclosures (2) and (3) detail the goals of this effort.

Enclosure (2): Preliminary AADC RAM-I/O Statement of Work is attached as Appendix 2.1. Enclosure (3) considers the AADC I/O Baseline [2.3, pages 39-47 and 2.5].

At the 1973 AADC Symposium Raytheon reported some research on using a PMU (Program Management Unit of the Data Processing Element) as the RAMM/IO Controller. The PMU would have a Task Memory for instruction and temporary data. The advantages of this approach are improved modularity, expandability, graceful degradation and memory protection (via the PMU), as well as, a cost savings since 95 percent of the controller is already designed [2.22]. Also see [2.23].

IBM also reported on development of a microprogrammed I/O controller. The controller consisted of a 4k 64-bit words of ROM (Read Only Memory), local store of 16 x 16 bits and a 16 bit minicomputer. In many ways it was similar to the PMU. See [2.24] for the final report on this development.

2.3.3 Interface to Aircraft

Reference [2.6] is a Grumman Aerospace Corporation report on the AADC interfact requirements for a representative F-14C aircraft weapon system. The primary goal of the report was to provide detailed definition of the interface of future aircraft systems to AADC. The report concludes that the 13 subsystems require an interface to AADC of 200,000 bits/second, and the AADC configuration for the F-14C should have two DPE, one RAMM for data, one BORAM, and another RAMM for buffering I/O, a hardware MEC and a Data Handling System for interface to the aircraft subsystem. The Data Handling system includes a Bus Control Unit, several Subsystem Controllers, and standard Interface Units. For further information see Chapter 10 or [2.6].

2.4 MISCELLANEOUS SUBJECTS

2.4.1 Transient Radiation Effects

Grumman Aerospace Corporation has completed two studies on the transient radiation affects on the AADC. The first was completed in July 1969 [2.7]; the second in July 1971 [2.8]. These reports are classified proprietary and secret, respectively.

2.4.2 Advanced Avionics Fault Isolation System (AAFIS)

The following excerpt is taken from AADC Progress Report No. 10 [2.3, paragraph 7 to 10]:

> 7. Advanced Avionics Fault Isolation System (AAFIS): "The AAFIS program is planned for the development of automatic test equipment for the support of Naval avionics in the post 1980 era. It will be phased into fleet use subsequent to the presently deployed VAST system. The prime objective of the AAFIS program is to reduce the cost of ownership of avionics support equipment.
>
> 8. "The AAFIS program is presently investigating, by an industry contract with RCA, a technique which appears promising for automated testing of analog devices. The technique has, however, proven unapplicable to digital systems." [2.9].
>
> 9. Among the more salient goals of the AAFIS program, from the point of view of AADC, are: a) compatibility with Large Scale Integrated (LSI) semiconductor technology, to the extent that it may provide inputs to the LSI design process itself and b) the utilization of test procedures which may be computer controlled and monitored, and which result in data which may be evaluated by the same computers - namely AADC.
>
> 10. Industry response to the NADC RFI was due on 7 January 1972. An RFP for AAFIS studies has been released.

Also see [2.22] for the latest developments on AAFIS.

2.4.3 AADC Building Block Module

Reference [2.10] is a report by Westinghouse Defense and Space Center entitled the "Building Block Module for Advanced Avionics Digital Computer". (This report has not been reviewed and it may even pretain to the basic LSI package for AADC, in which case it should be in Chapter 3.)

2.5 OTHER NON-AADC ARCHITECTURES

2.5.1 Directly Executing HOL Architectures

This section is included to reference some of the other Higher Order Language architectures that are alternate designs to the AADC. These alternate designs should be reviewed to ensure that the features they offer or the implementation techniques they use are not superior to AADC. If AADC is going to be the All Application Computer for 1975 to 1985, it must offer a flexible efficient High Order Language that can be effectively implemented.

An original 1968 proposal for an "Integral Hardware/Software Design" is given in [2.11]. Two other very early alternates to the AADC design were presented in Subsection 1.3.2 [1.19 and 1.20].

Three more recent HOL Architecture designs include: An Aerospace HOL Computer by Honeywell in February 1971 [2.12], another by Burroughs Corporation in April 1971 [2.13], and the third by Hughes Aircraft Company in April 1971 [2.14].

The U. S. Air Force is also investigating the design of an HOL architecture. The conception, feasibility and initial design are described in [2.15]. A follow-on study was carried out under contract [2.16] and should be completed by now. A study on SPL Architecture Study is given in [2.17]. (SPL, or Space Programming Languages, is the USAF's competitor for CMS-2, and is described in [2.18].)

A final study by the Corporation for Informations Systems looks like it should be in Chapter 9 on Higher Order Languages rather than here, but it does pertain to the Air Force effort [2.19].

Reference [2.20] discusses another possible architecture using a distributed fetch computer, not specifically designed for a HOL. Reference [2.21] suggest a universal function unit for avionics and missile systems.

References to AADC Architectures

2.1 The Advanced Avionics Digital Computer Revisited; R. S. Entner; NAVAIR-SYSCOM; October 12, 1971; Unpublished paper; Unclassified; (NPS).

2.2 AADC Development Program Progress Report No. 9; R. S. Entner; NAVAIRSYSCOM; November 1, 1971; (67, NPS)*.

2.3 AADC Development Program Progress Report No. 10; R. S. Entner; NAVAIRSYSCOM; May 31, 1972; (78, NPS).

2.4 High-Speed Optical Data Links; S. R. Parsons, D. J. Stigliani; R. C. Clapper; D. W. Hanna; IBM, Federal Systems Division; NAVAIRSYSCOM Contract No. N00019-71-C-0345; April 1972; (87).

2.5 AADC I/O Baseline; Carl Mattes; NAVAIRDEVCEN; AEDC April 10, 1972; Also available as Enclosure 3 to AADC Progress Report No. 10 [2.3]; (NPS).

2.6 Future Naval Aircraft Subsystems/AADC Interface Definition for Operational and OBC Requirements - Final Report; W. I. Butler and G. A. Kohler; Grumman Aerospace Corp.; NADC Contract No. N62269-72-C-0065; April 17, 1972; (89, NPS).

2.7 A Generalized Study on Transient Radiation Effects of Advanced Aircraft Computer Systems (U); Grumman Aerospace Corp.; July 1969; for NAVAIR AIR-5333F4; Unclassified - Proprietary; (8).

---

* This number in parentheses refers to the sequential reference number in the AADC Bibliography [2.3, Enclosure 1]. NPS indicated the report is available at the Naval Postgraduate School.

2.8 Transient Radiation Effect Study Pertaining to Development of Modular Processor Aircraft Computer System (U) Final Report; Grumman Aerospace Corp.; July 1, 1971; NADC Contract N62269-70-C-0340; Secret; (59).

2.9 Request for Information for an Advanced Avionics Fault Isolation System; NADC; October 26, 1971.

2.10 Building Block Module for Advanced Avionics Digital Computer; Westinghouse Defense and Space Center; October 1971; NAVAIRSYSCOM Contract N00019-70-C-0505; (64).

2.11 Integral Hardware/Software Design; Larry L. Constantine; Part 2, Modern Data Systems, May 1968, pp 22-30.

2.12 Aerospace Higher Order Languages (HOL) Computer; Honeywell; February 24, 1971.

2.13 Aerospace HOL Computer; Burroughs Corp.; Proposal #B-2301-D Revised; April 16, 1971.

2.14 Addendum to Technical Proposal for Aerospace Higher Order Language Computer Study; Hughes Aircraft Co.; Report #TP71-50; HAC Ref. #C3602-002; April 1971; (In response to RFP #33615-71-Q-1775.)

2.15 Development of a Higher Order Language Architecture; James R. Foster, Jr.; Air Force Avionics Laboratory; in NAECON '71 Record; May 17-19, 1971; pp 201-205; Available from IEEE Transactions on Aerospace and Electronic Systems, Reference 71-C-24-AES; (NPS).

2.16 Aerospace HOL Computer; Air Force Avionics Laboratory (AFAL/NVE), WPAFB, Ohio; Contract FY33615-71-C-1775.

2.17 Space Programming Language Machine Architecture Study; Space and Missile Systems Organization (SAMSO/SYGN), Norton AFB, California; Volume 1; Final Report; Contract F04701-71-C-0200; May 15, 1972; (NPS).

2.18 SPL/Mark IV Reference Manual; SAMSO-TR-70-349; (I assume this is available from Space and Missile Systems Organization (SAMSO/SYGN), Norton AFB, California).

2.19 Software Technology Study for Advanced Guidance Computer Architectures; Final Report; Corporation for Information Systems; Contract F04701-71-C-0183; May 1, 1972; (NPS).

2.20 Architectural Study of a Distributed Fetch Computer; Alan J. Deerfield, Raytheon; NAECON '71; May 17-19, 1971; pp 214-217; (See [2.15]); (NPS).

2.21 A Universal Function Unit for Avionics and Missile Systems; Frank J. Langley, Raytheon; NAECON '71; May 17-19, 1971; pp 178-187; (See [2.15]); (NPS).

2.22 All Application Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceedings not yet available.

2.23 Final Report for the AADC Ramm-I/O Functional Black Diogram Design Study; Raytheon Report BR-7516; April 1973; NAVAIRDEVCEN Contract N62269-73-C-0051; Unclassified; (NPS).

2.24 Study of AADC-I/O; Final Report; IBM Report 73-559-0003; April 1973; NAVAIRDEVCEN Contract N62269-72-C-0831; Unclassified; (NPS).

ENCLOSURE (2)

NAVAL AIR DEVELOPMENT CENTER
AERO ELECTRONIC TECHNOLOGY DEPARTMENT
WARMINSTER, PENNSYLVANIA 18974

AEDC
17 Apr 1972

PRELIMINARY
AADC RAM-I/O
STATEMENT OF WORK

1. OBJECTIVE/SCOPE

The objective of this initial study concerning the RAM-I/O (Random Access Memory-Input/Output) portion of the AADC (Advanced Avionics/All Applications Digital Computer) is to establish basic concepts and designs which will fully utilize the advanced capabilities and modular flexibility of the AADC. The I/O should optimize speed and flexibility of communication between the internal AADC processing elements and the external subsystem data processors, sensors, and controlled peripherals. This study shall result in three types of highly flexible I/O structures which can be configured to meet the data processing requirements foreseen for the 1978-1990 time frame. The study shall provide a detailed functional (register level) design of the RAM-I/O portion of the AADC and provide the basis for a subsequent contract to generate a detailed logic design.

The I/O types are as follows;

a. A standard AADC interface
b. A conventional dedicated multichannel interface
c. Flexible serial interface for multiplexed airborne applications

2. SUBJECT

The subject of this study shall be the RAM-Input/Output portion of the AADC.

3. DETAILED STUDY PROGRAM

a. Introduction

The contractor will perform a three (3) part analysis/study/design of an I/O System for the Navy's AADC which is now under development. The three technical areas to be covered are:

(1) the design of the RAM-I/O Architecture and Standard AADC I/O Bus (see reference (a)),

(2) the design of a conventional dedicated multichannel interface as described in reference (b), and

(3) the design of a serial multiplexed I/O interface.

This contract is not intended to be a study of Navy I/O requirements. Where necessary, requirements information relevant to the designs will be provided to the contractor by the Navy.

In all the designs there will be included a complete reliability/maintainability philosophy and a rigorous analysis of the failure modes and fail safe capabilities of the design. Trade off studies, parametric analysis, hardware/software considerations, specifications and justifications will also be included.

The designs discussed below shall incorporate a modular approach for easy expandability of I/O channels and/or expandability of the number of Random Access Memories.

In addition, the designs should reflect the fact that computer to computer communications will be handled by any of the I/O interfaces described below without any modifications to the designs.

b. Discussion - RAM-I/O and Standard AADC I/O Bus

A trade off analysis of potential programmable IOC (Input Output Controller) and RAM configurations shall be performed. A register level design and timing diagrams shall be generated for the architecture(s) recommended to satisfy the Navy's needs. The contractor shall give prime consideration to the possibility of implementing the IOC function with the Program Management Unit portion of the Processing Element (see references (a) and (c)) or the MCU (Microprogrammed Control Unit) currently under development within the Navy. The MCU is being developed to provide control of arithmetic or logic units for signal processing, emulation, and I/O control. Detailed information about both designs will be supplied to the contractor early in the contract period.

The above design(s) shall provide and reflect all control functions necessary for:

(1) Operation of the I/O Bus
(2) Operation of RAM including:

(a) Multiporting/multipartioning consi rations
(b) Buffering considerations, e.g.

Types of buffers
Buffering techniques
Buffer contiguity
Buffer Controls
buffer lengths
buffer status
buffer acquisition
buffer control words

In addition, the contractor will also be required to recommend a design for the "standard AADC I/O" bus system to meet the requirements of the "AADC I/O Baseline." This bus will be essentially a parallel version of the serial multiplexed I/O bus discussed in a following section. It is the goal of the study that the number of peripheral devices connecting to the bus be limited only by the word rate capability of the bus.

Effort will be expended in (but not necessarily limited to) the following technical areas:

(1) Detailed investigation of I/O-RAM interrelationships with specific recommendations and justifications.

(2) Interrupt Notification/Handling and priority communications.

(3) MEC (Master Executive Control) impact.

(4) Modularity/growth considerations.

c. Discussion - Conventional Dedicated Multichannel Interface

The Conventional Dedicated Multichannel Interface to be designed shall comply with the requirements of reference (b). This work shall be in general conformity to that described in the preceeding section.

d. Discussion - Serial Multiplexed I/O Interface

During the study/design several multiplexed alternative I/O bussing systems shall be considered. The bussing system(s) recommended shall be detailed to a functional block diagram (register) level and timing diagrams shall be provided from which a logic design can be generated under a future contract.

As part of the designs, an interrupt and bus acquisition scheme or schemes shall be developed. It is a desired goal of this study that the number of peripheral devices connected to the bus be limited only by the

bit rate of the bus. For example, if the bus has a 5 illion data bit per second capability, any number of devices can be co nected until their combined information transfer rates just equal that capability i.e., five (5) peripherals operating at 1 megabit per second each or 100 peripherals operating at 50 kilobits per second.

REFERENCES

(a) NAVAIRDEVCEN AEDC AADC I/O Baseline Document of 10 Apr 1972
(b) NAVAIRDEVCEN AEDC Functional Specification for the Conventional Dedicated Multichannel Interface of 10 Apr 1972
(c) Raytheon Co. Uncl-NoForn AADC Arithmetic and Control Functional Block Diagram Design Analytical Study of Dec 1970, DDC No. AD880544

Chapter 3

A L L

A P P L I C A T I O N

R O L E

Table of Contents for All Application Role

Chapter 3

ALL APPLICATION ROLE

3.1 INTRODUCTION AND SUMMARY

Although the AADC was originally intended for Naval avionic applications only, the high speed, powerful instruction repertoire and low cost of AADC have caused the proponents to consider much wider applications. Although most of this report addresses the AADC design for the avionic application, this chapter will discuss some of the implications of the decision in 1971 to convert the AADC to an All Application Digital Computer. Although "all application" is undoubtedly too general, the acronym AADC was retained because of the wide variety of intended applications and because it has been in use for 3 years. Also All Application Digital Computer sounds better than Almost All Application Digital Computer.

3.1.1 Implications of All Application Role

Certainly the most significant change in the AADC program in the last year is the change in emphasis from avionic applications only to the All Application Digital Computer. This has caused significant changes in the AADC design by requiring many of the features that have caused problems in the present third-generation computers. For example, rather than having a Data Processing Element (DPE) executing a single program out of its own Task Memory, the DPE must now have facilities for multiprogramming, virtual memory and demand paging. Thus, the DPE must now have special hardware to support the virtual memory, and much faster busses to handle the increased bus traffic. Furthermore, the AADC will now have to solve problems, such as thrashing (excessive paging until throughput drops to almost zero) and system deadlocks, that have remained unsolved in present day computers. In any case, the AADC supporters are convinced they can overcome these problems with AADC.

3.2 DESIGN IMPLICATIONS OF ALL APPLICATION ROLE

3.2.1 General Problem Areas

This section is intended to provide background on the general problems of operating a computer in a multiprogramming and paging environment. Although the problems and solutions discussed here are not unique to AADC, they are presented to provide background for discussing the implementation of multiprogramming and paging on AADC in the next section.

An All Application Digital Computer will be defined here as a computer capable of operating effectively on normal batch processing, time sharing, data processing and real time applications. An example of each type of application is processing jobs in a batch at a programming center, supporting several terminals for interactive computation, maintaining inventories at a supply center and data analysis on control of an aircraft. In order to operate effectively in all these application areas, a computer must have the following features:

- virtual memory (or paging)
- multiprogramming

Multiprogramming means more than one job in the "active status" in the main memory (i.e. TM) at one time. The virtual memory feature means that a large (virtual) memory can be addressed as if it were actually main memory. Thus the user assumes he has a large virtual memory for his programs and data, and the system ensures that the required segments of the virtual memory are in the main memory when required. These two features could be provided separately but are generally provided together.

Incorporating these two features into a (serially processing) computer introduces the following problem areas:

- selection of an optimal page (segment) size,
- page fetching strategy,
- page placement strategy,
- page replacement strategy,
- address binding at run time,
- task switching, and
- storage protection.

Many possible solutions has been used in existing general purpose computer system to handle these problems. Page sizes range from 64 word to 4096 words, with the smaller page sizes usually causing the least load on the channels (busses) to the backup storage (disk or drums) but also causing the largest amount of CPU overhead. The two common page fetching strategies are one page on-demand (or as required) fetching and one or more page pre-fetching. The common page placement strategies are selecting the first available space or the smallest available space. The first-space strategy is common with fixed sized pages while the smallest space is common for variable sized pages. The page replacement strategies are many and varied. They include random (the simpliest), first-in first-out (FIFO), last-in first-out (LIFO), least recently used (LRU), optimal, various combination of these, etc. The optimal replacement policy is defined, a-posteriori, as the one that minimizes the number of pages that must be transferred to the main memory and thus can only be determined after the program has executed. Since the optimal is determinable after-the-fact, it has no predictive powers and can not be implemented. The problem of address binding* at run time results from the fact that the pages for any Program Module (PM) are randomly distributed throughout the main memory (i.e. TM), because of the placement and replacement strategies. Thus all addresses must be converted (bound) to the absolute

*"Address binding" as used here does not mean binding that occurs only once; "Address translation" or "Address mapping" may be more appropriate terms.

TM addresses at execution time. Common methods of achieving run time binding include using base registers and using associative memories. One of the disadvantages of run time binding is that each branch operation now implies an obligatory indexing operation.

If more than one program is active in the Task Memory at one time then some method must be provided for protecting one program's storage from being destroyed by another. Some solutions include using protection keys on each page and checking for addresses out of bounds. The last problem area in implementing virtual memory and multiprogramming is task switching. Generally this involves maintaining lists or queues of tasks ready to use the DPE (data processing element), waiting for a page to be brought into TM and waiting for I/O. Further elaboration on the specific strategies to be used by AADC will be presented in the next sections.

3.2.2 AADC Strategies for Paging Program Modules

The justification for adding a virtual memory feature to AADC was obtained during simulation studies when it was found that only about one third of the segments of a program module were active at any one time [3.1, paragraph 2]. Thus it was concluded that three or more program modules could have their "active" pages in TM simultaneously, which would decrease the task switching time and improve the performance.

The general strategy for implementing virtual memory on the AADC is described in the following excerpt taken from AADC Progress Report Ten [3.2, paragraph 33]:

> 33. Demand Paging: In order to reduce processor inefficiency produced by the transfer of unnecessary procedure from BORAM to Task Memory, a demand paging scheme is being developed for AADC. In this manner, only a kernel page is loaded into TM at the outset of

> a Program Module (PM) execution cycle, after which only those pages containing procedure actually requested by the running program are transfered into TM. Because the total number of pages required to execute a particular PM without excessive requests for new pages during some interval of time may exceed the number of pages which constitute the available TM storage (< 4K words, allowing for data and scratch pad), some means must be provided to intelligently replace unneeded pages with new ones. Because a page which is not required at one moment may be required the next, care must be taken to not arbitrarily toss out "unnecessary" pages. Enclosures (5), (6) and (7) address the issues, philosophy, alternatives, design tradeoffs and simulated results of paging and page replacement algorithms for AADC. [3.3 to 3.5].

For AADC the virtual memory feature will be implemented by assigning fixed-sized segments, called pages, to all procedures and data. Pages of Program Modules will be moved from BORAM to the Task Memory while data pages will be moved between RAMM and TM in both directions. The paging of Program Modules is described in this subsection, while the paging of data is described in the next subsection.

The selected page size for Program Modules is 256 words, which is a convenient size for BORAM. This page size allows 16 pages in the 4K word Task Memory*. With 2 microsecond (usec) block access time to BORAM and 150 nsec per word transfer rate, it takes 40.4 usec (10^{-6} sec) to load a page into TM compared to 646 usec to load the entire TM. Thus task switching can be substantially improved with a paged memory. The page fetching strategy for AADC is demand paging. The alternate strategy of prefetching pages is not reasonable because the TM is being accessed continually during the page transfer and therefore programs cannot be executed on the DPE during the transfer. The page placement policy for AADC is to select the first empty space if one exists. If there are no empty spaces then a page is selected for removal from TM by the page replacement strategy.

*There has been some discussion on making the TM extendable to 16K words.

The page replacement strategy for AADC has been left very flexible. In fact, according to Raytheon at the January 1973 conference [3.6], 16 possible replacement strategies are to be implemented including random, first-in first-out, least recently used, Raytheon's load forward reverse grain (similar to last-in first-out), and user specified. This appears to be a very complex solution when the flexibility is not justified. According to A. W. Cerillo and C. F. Mattes, NADC, the performance of all replacements strategies is almost the same (within 5 percent) with Raytheon's load forward reverse grain (LIFO) algorithm being the most appropriate page replacement algorithm. It is more efficient than first-in first-out (FIFO) and easier to implement with about the same efficiency as the least recently used (LRU). They also conclude that in cases where there is only one process whose pages cannot all fit into the main memory, for example in AADC, the most appropriate page replacement strategy is LRU not the working set [3.4]. This last recommendation is taken from Denning's paper [3.7].

Mr. William R. Smith at NRL also found very little variation in performance of the various replacement algorithms for AADC on an avionic (E2B) work load. Based on the simulation of possible AADC replacement algorithms, the NRL's recommended replacement schemes, in decreasing order of preference, are:

1. FIFO/LRU by pairs
2. RANDOM/LRU by pairs
3. FIFO.

where FIFO/LRU by pairs means a pair is selected on the first-in first-out basis and then the least recently used one of the pair is selected for replacement [3.5].

The problem of address binding at run time has not been very well specified yet. The DPE address field has been increased from 12 to 16 bits with the first 8 bits being the page address and the last 8 bits being the address within a page [3.6]. A sixteen bit address field means that the largest Program Module or data array is 64k words. Programs may contain several PMs. (The 16-bit address field is a change from the original proposal of using 32-bit addresses [3.1].)

The next problem area introduced by the all application role and the need for multiprogramming is the problem of task switching. To facilitate task switching on the AADC, each program module (PM) is assigned a kernal page which must be in the TM whenever that PM is active. The kernal page contains the BORAM address of all other pages. Furthermore each page has a kernal word which is used for task switching and for storage protection. A description of the use of the kernal word for task switching is not yet available at NPS. For storage protection, bits 32 to 36 of the kernal word are used for read protection, write protection, command protection and parity, respectively. Thus it is possible to specify that a page can not be read from or written into, can only be read, or contains program instructions or data [3.1, paragraph 14-19].

Honeywell has also completed a demand paging analysis in which they recommend:

1. Using a 256-word BORAM page and virtual memory addressing technique. Programming cost can be cut by 25 to 45 percent by dynamic overlay management rather than user specified overlay scheme.
2. Using demand paging scheme and either least recently used or working set as the page replacement algorithm. Later they say there is very little advantage of working set strategy

when using a multiprocessor system rather than a single processor multiprogramming system. Thus LRU would be simplier and better.

Volume I summarizes the results of the demand paging analysis while Section 2 of Volume II provides details of the analysis, advantages and disadvantages of paging, addressing methods, simulations models used and the effects of paging on the system, on the MEC and on the Data Processing Element [3.8].

3.2.3 AADC Strategies for Paging Data

Mr. William R. Smith at NRL has suggested that AADC's two level memory hierarchy between RAMM-BORAM and Task Memory is similar to the IBM 360/85 cache-memory system. This fact alone can give insight to the operation of AADC in a paging environment. Reference [3.3] attempts to summarize those portions of the cache memory literature that pretain to the AADC addressing and data management. This literature suggests that data pages should be no larger than 32 words and preferably 16 words. According to Smith, "A natural utilization of both BORAM and RAMM features would involve having 32 relocatable sectors [pages] of 128 words each in Task Memory. A sector of a procedure would be transferred in its entirety from BORAM to Task Memory but a data sectors would be transferred from RAMM one block (16 or 32 words) at a time as referenced." (It now appears that the 4k-word TM will be divided into sixteen 256-word pages not 128-word pages as recommended by Smith [3.6].) This two page size seems necessary in order to keep the bus traffic within reasonable limits. Moving data, even with blocks as small as 16 words, can be expected to cause one data word transfer per instruction executed - thus burdening the RAMM/TM memory interface channel. Program pages smaller than

128 words are incompatible with BORAM and would cause an excessive number of accesses to BORAM.

A cache technique that would work quite well in AADC is the "store through" in which data store operations are carried through to secondary storage (RAMM) in parallel with local storage (TM). As well as improving the processing speed the "store through" technique ensures the residence in RAMM of "fresh" data for system output without the necessity of moving data from TM to RAMM at crucial points in a program. Smith also presents evidence that direct access of data from RAMM would be superior to moving 128 word blocks of data to TM in most cases. [3.3, pg. 74].

3.2.4 Other Implications

Thus far, it appears that very little investigation has been undertaken into determining what special features would be useful for manipulating large files such as required in supply inventory applications and in management information systems. Some of the preliminary investigations on the external Input/Output controller are described in Chapter 2, but apparently no one has yet addressed the problems of file maintenance on large disk or tape files. This area will undoubtedly be investigated further in the near future as AADC continues toward an All Application Role.

Although this chapter is relatively short at the present time, it is expected to expand rapidly as further implications from the All Application Role of AADC are investigated.

References for All Application Role

3.1 AADC Development Program Progress Report No. 9; R. S. Entner, NAVAIRSYSCOM; November 1, 1971; (67, NPS).

3.2 AADC Development Program Progress Report No. 10; R. S. Entner, NAVAIRSYSCOM; May 31, 1972; (78, NPS).

3.3 Implication of Published Program Behavior Statistics on the AADC Addressing and Data Management Question; William R. Smith, NRL; Ref 5030-8:WS:mib; Jan 7, 1972; Available as Enclosure 5 to AADC Progress Report No. 10 [3.2]; (NPS).

3.4 Replacement Algorithms for AADC Analyzed; A. W. Cerillo and C. F. Mattes, NAVAIRDEVCEN; AERD, May 17, 1972; Available as Enclosure 6 to AADC Progress Report No. 10 [3.2]; (NPS).

3.5 Simulation of AADC Page Replacement Algorithms; William R. Smith; NRL Memorandum Report 2464; July 1972; (NPS). Preliminary version available as enclosure 7 to AADC Progress Report No. 10 [3.2].

3.6 All Application Digital Computer 1973 Symposium; Orlando, Florida; Jan. 23-25, 1973; Proceedings not yet available.

3.7 The Working Set Model for Program Behavior; P. J. Denning; Communications of the ACM; May 1968; pp 323-333. **(NPS).**

3.8 Master Executive Control for the Advanced Avionic Digital Computer; Interium Report; Honeywell Report Z9508-3018; June 1972; NAVAIRDEVCEN Contract N62269-72-C-0051; Unclassified; Volume I and II; **(NPS).**

Chapter 4

A A D C

H A R D W A R E

T E C H N O L O G Y

Table of Contents for Hardware Technology

List of Figures for Hardware Technology

List of Tables for Hardware Technology

Glossary for AADC Hardware Technology

BORAM - Block Oriented Random Access Memory: organized in 128 to 512 word blocks for program modules and used to store Program Modules and permanent data.

CCD - Charge Coupled Device: competitor for MOS for BORAM.

CCSL - Compatible Current Sinking Logic.

CFM - Closed Flux Thin Film Memory: used for RAMM and TM.

CMOS - Complementary Metal Oxide Semiconductor.

CMTL - Current Mode Threshold Logic - current mode.

ECL - Emitter Coupled Logic.

IC - Integrated Circuits: technology used in third generation computers where transistors, resistors and capacitors are built together as different layers of conductor, insulator and semiconductor materials.

LSI - Large Scale Integration

MNOS - Metal N-channel Oxide Semiconductor: as contrasted to MOS which usually refers to P-channel MOS.

MOS - Metal Oxide Semiconductor circuits or memory.

MSI - Medium Scale Integration.

NDRO - Nondestructive read-out: memory does not have to be written after reading.

PE - Processor Element: sequential processing unit (see Chapter 5).

RAM - Random Access Memory: any word is addressable.

RAMM - Random Access Main Memory: used to store mode-independent data and buffer I/O.

SSI - Small Scale Integration: same as IC.

TM - Task Memory: a RAM attached to PE to hold currently executing program module.

TTL - Transistor-Transistor Logic, probably the most common semiconductor logic technology. Also called T^2L.

Chapter 4

AADC HARDWARE TECHNOLOGY

4.1 INTRODUCTION AND SUMMARY

4.1.1 Scope of Chapter

This chapter will discuss the new advances in hardware technology that are being developed for AADC. Although the development and production of modules using advanced hardware technology (at reasonable cost) is very important to AADC, the details of the technology and how it is implemented is of minimal interest in a course such as this one on the concepts and operations of AADC. In other words, the fact that the technology exists, has been proven, and can be mass produced at reasonable cost is certainly of interest, but the details of the technology and its implementation is considered beyond the scope of this report. Therefore, this chapter is an overview of the latest hardware technology emphasizing what has been implemented and proven, as well as, what will probably be in production by 1975.

Under the heading of hardware technology is placed all work which relates to the physical constituents of the AADC - the devices which will ultimately become the PEs, the RAMs, the BORAMs, the buses, etc.. In other words, all that which will ultimately manifest itself in the physical computer. The hardware technology is divided into three major areas: Large Scale Integration (LSI) technology, memory technology and bussing technology.

4.1.2 Summary of LSI Technology

The basic AADC hardware building block module is an hermetically sealed (perfectly airtight) package capable of supporting either multi-chip arrays on a

ceramic substrate, chip/wafer hybrids, or semiconductor monolithic three-inch diameter wafers - or any combination of these. ("Monolithic" means many circuits attached together to resemble one uniform pattern, i.e., a 5000 gate LSI wafer.) This year (1972) one of two AADC packaging modules has passed environmental testing at Naval Avionics Facility, Indianapolis. A complete second level packaging system is presently under development at Singer-Kearfott, and will be similarily tested later this year [4.1, paragraph 23].

There is ample evidence that the technology will mass produce 5000 gates on a 3-inch diameter wafer by 1975. Texas Instruments is producing a Logic Slice Type "P" which has the equivalent of 857 gates on a 1½-inch wafer. Intel Corp. has built the CPU of an 8-bit parallel microcomputer the MCS-8 on a single chip [4.2]. There are now examples of 1500 gate LSI chips available off-the-shelf but the author does not have exact references.*

More details on the developments in LSI technology for AADC will be presented in Section 4.3. Many other articles on expected hardware developments can be found in the Proceeding of the Advanced Digital Technology Conference in June 1971 [4.3].

4.1.3 Summary of Memory Technology

Two promising magnetic storage technologies for AADC are the block oriented ferroacoustic memory for BORAM and the random access closed flux path thin-film memory (CFM) for RAMM and TM. The ferroacoustic technology employs the coincidence of mechanical and electrical energy to write magnetic domains into homogeneous, amorphous (non-crystaline), semi-closed flux path permalloy film. (Permalloy is a highly magnetic alloy of iron and nickle.) These domains are subsequently interrogated by way of an acoustic strain wave. A plated wire

---

* Current work at Hughes Aircraft is on 2000 gate/chip on a 2 inch diameter substrate [4.27].

may be used for the ferroacoustic memory in place of the thin film. The ferroacoustic memory is low cost (0.1 to 0.5¢/bit), high speed (150 nsec/wd read and 1 - 2 μsec/block access time), high density (5000 bits/in^3), low power (2 μwatts/bit), low weight (7.5 lbs for 64K 36-bit words, i.e., 2.3 megabits), non-volatile, and uses NDRO (non-destructive read out) techniques [4.1, page 13]. Blocks may be 128 to 512 64-bit words. For more details on the technology see [4.3].

Another magnetic technology, tentatively called Cross Tie Memory and similar to a bubble memory, is also under investigation for possible use in BORAM [4.1, paragraph 26].

The CFM, a planar thin film analog of a plated wire, offers new capabilities for random access magnetic storage. It provides performance heretofore believed realizable only with semiconductors, but without the twin penalties of high power and data volatility. In comparison to previous magnetic memories, CFM is low cost (1¢ to 3¢/bit), high speed (80 nsec access time, 100 nsec read time with NDRO, and 150 nsec write time per word), high density (5000 to 11,000 bits/in^3), low power (100 μwatts/bit), low weight (3 lbs for 4K 36-bit words or 150K bits) non-volatile, and uses NDRO technique.

In comparison to ferroacoustic memories, CFM is 2 to 30 times more expensive, about twice as fast, up to twice as dense, uses 50 times more power, and is 6 times heavier. Thus a 64K word BORAM costs $2300 to $11,500; a 4K word TM costs $1440 to $4320.

It is believed that semiconductor memories will be very competitive by 1975. The 1973 AADC symposium presented several possible semiconductor memories as candidates for the AADC memories [4.27].

More details and references to the developments in memory technology will be presented in Section 4.4.

4.1.4 Summary of Other Technologies

Because of AADC's very small geometry, modularity and need for wide bandwidth internal busses, optical communication is being considered seriously for AADC internal bussing. The optical bussing has distinct advantages over all electronic alternatives in the area of noise immunity and ease of connection. See Section 4.5 and [4.5].

The other improved technology is in the electric power distribution system. It is proposed to replace the conventional electro-mechanical relays with a Solid State Electric Logic (SOSTEL) power distribution system. SOSTEL will greatly reduce power consumption, wiring complexity and weight, as well as increasing the control over electrical power distribution. See Section 4.6 and [4.24 and 4.25].

4.2 AADC TECHNOLOGY PHILOSOPHY

The following statement of AADC technology philosophy and current status of LSI packaging is taken from AADC Progress Report No. 10 [4.1, paragraph 22 - 23]:

> AADC technology philosophy calls for the use of 1975 state-of-the-art technology in 1975, followed by gradual technology improvements through the system's life time. These improvements should, however, remain transparent to the user and, in turn, the procuring agency. AADC building blocks will be specified in terms of function, form and interface. Legal improvements to these building blocks will, therefore, affect cost, reliability, and availability only.
>
> For this philosophy to be meaningful, it is important that the packaging system developed for AADC be compatible with present and projected component technologies. The AADC basic building block module - an hermetically sealed package capable of supporting hybrid, multi-layered ceramic and semiconductor substrates up through monolithic 3" diameter silicon wafers - is just such a package. This year has seen at least one of two AADC package designs pass full MIL-E-5400 Class 4X spec testing at NAFI.

From the inception of AADC, its hardware technology aspect has always attracted the greatest measure of skepticism. Conversely, AADC Program Management has consistently said that AADC does not depend on advanced technology for its feasibility. There is agreement on one point, however - that a state-of-the-art AADC will not be the same revolutionary machine advanced hardware technology will make it. For the present, based on the success of initial development efforts, as well as independent Industry performance, no modification of earlier projections appears warranted.

4.3 LSI TECHNOLOGY

4.3.1 Packaging

In February 1969, when the Naval Air Systems Command announced their intention of procuring a first level packaging system capable of supporting discretes, ICs, MSIs and wafer technology out to three inches in diameter, the immediate and predicted majority response was that of incredulity. Today, hermtically sealed ceramic and metal-ceramic packages meeting the original requirement have passed environmental tests (MIL-E-5400 Class 4X spec.) at the Naval Avionic Facility, Indianapolis [4.1, paragraph 23]. A photograph of one such package is shown in Figure 4.1. A schematic diagram AADC first level package containing multiple IC or MSI chip, chip/wafer hybrid or whole wafer (LSI) is shown in Figure 4.2.

One or two of these LSI modules (packages) will contain the entire Processing Element (Chapter 5) of 10,000 to 12,000 gates. This seems quite realistic since 1000 to 1500 gates are presently being put on a chip. See Figure 4.3. The LSI modules will be housed in a high level LSI package shown in Figure 4.4. The results of a study on high level packaging by Singer Aerospace and Marine System is reported in [4.4].

The space required to package 10,000 gates has decreased two orders of magnitude in the past 20 years and is expected to drop another 10 fold in the next 3 years. At the same time, the cost is expected to drop by a factor of 250 times as shown in Figure 4.5,

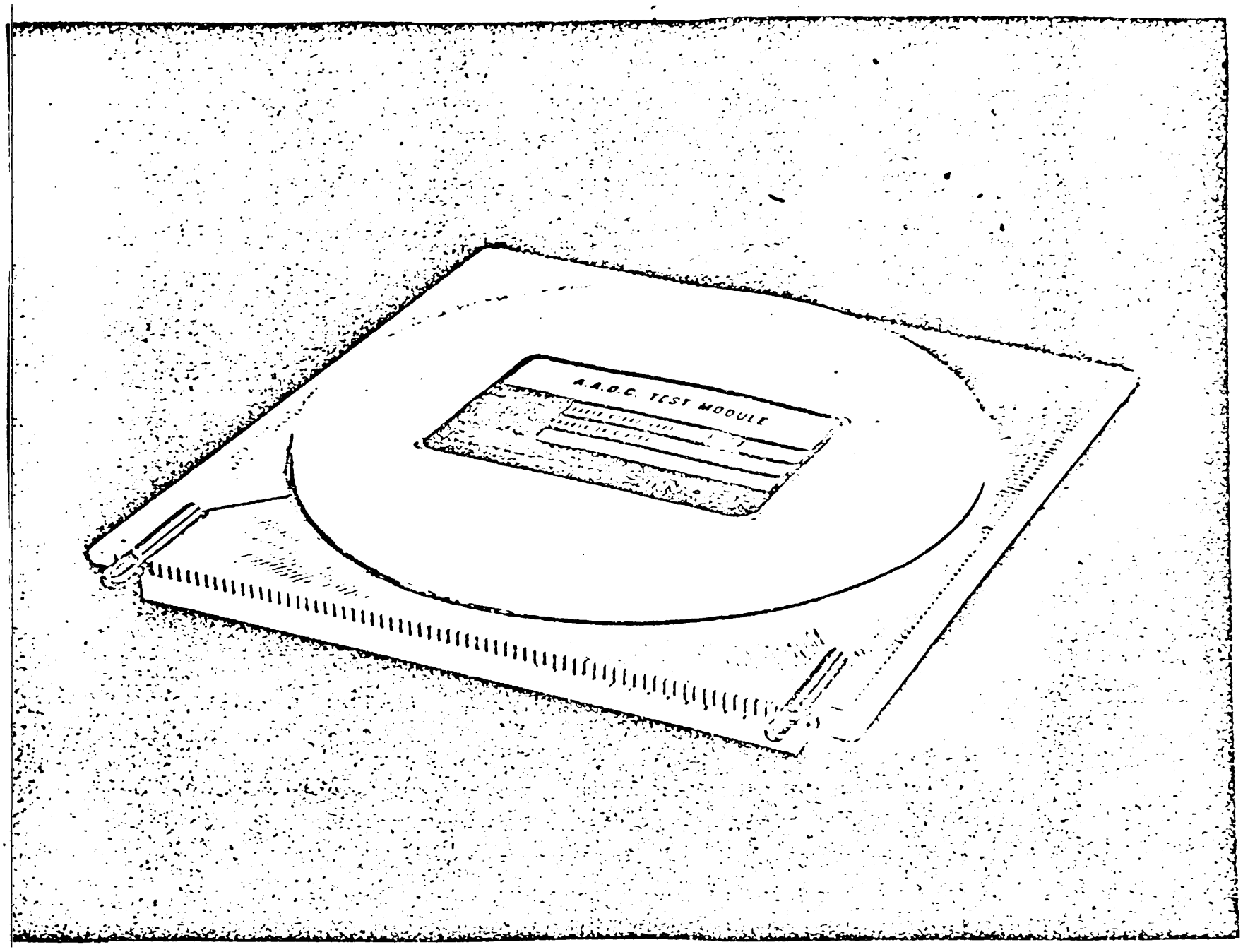


Figure 4.1. One Type of AADC Super LSI Package

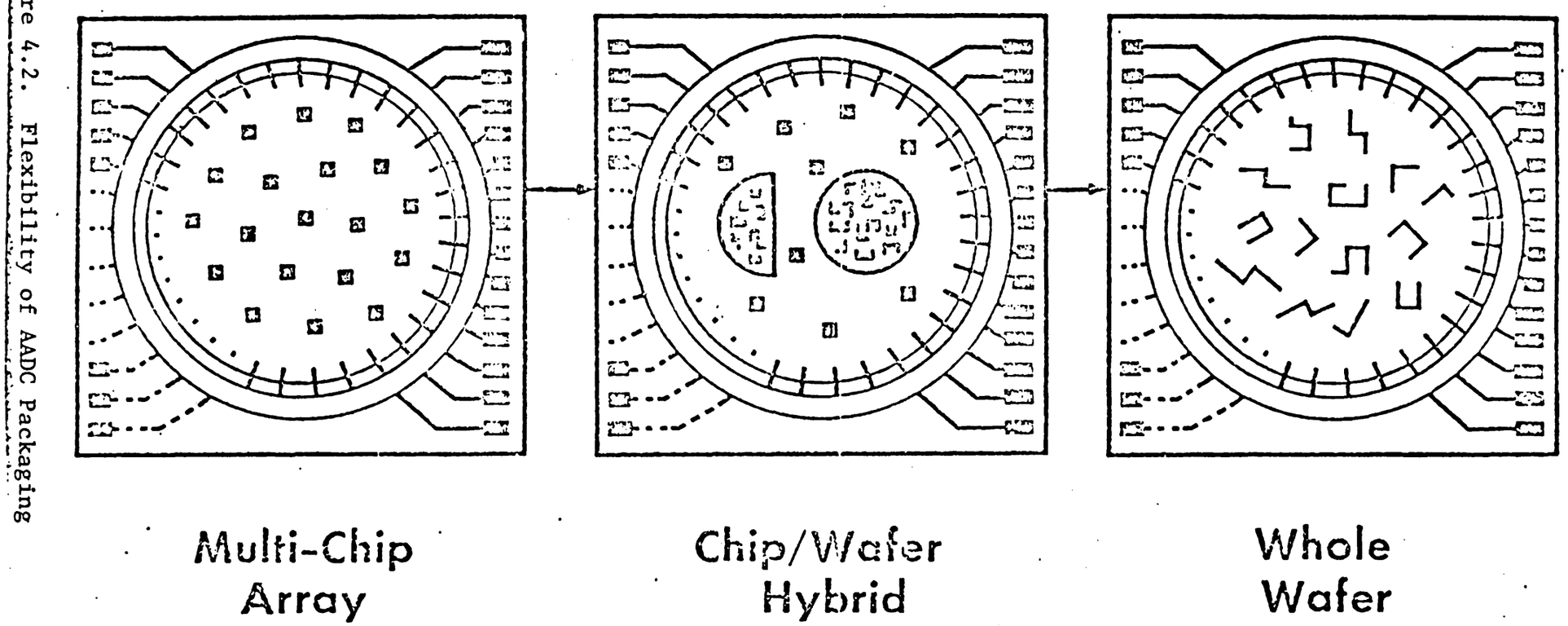
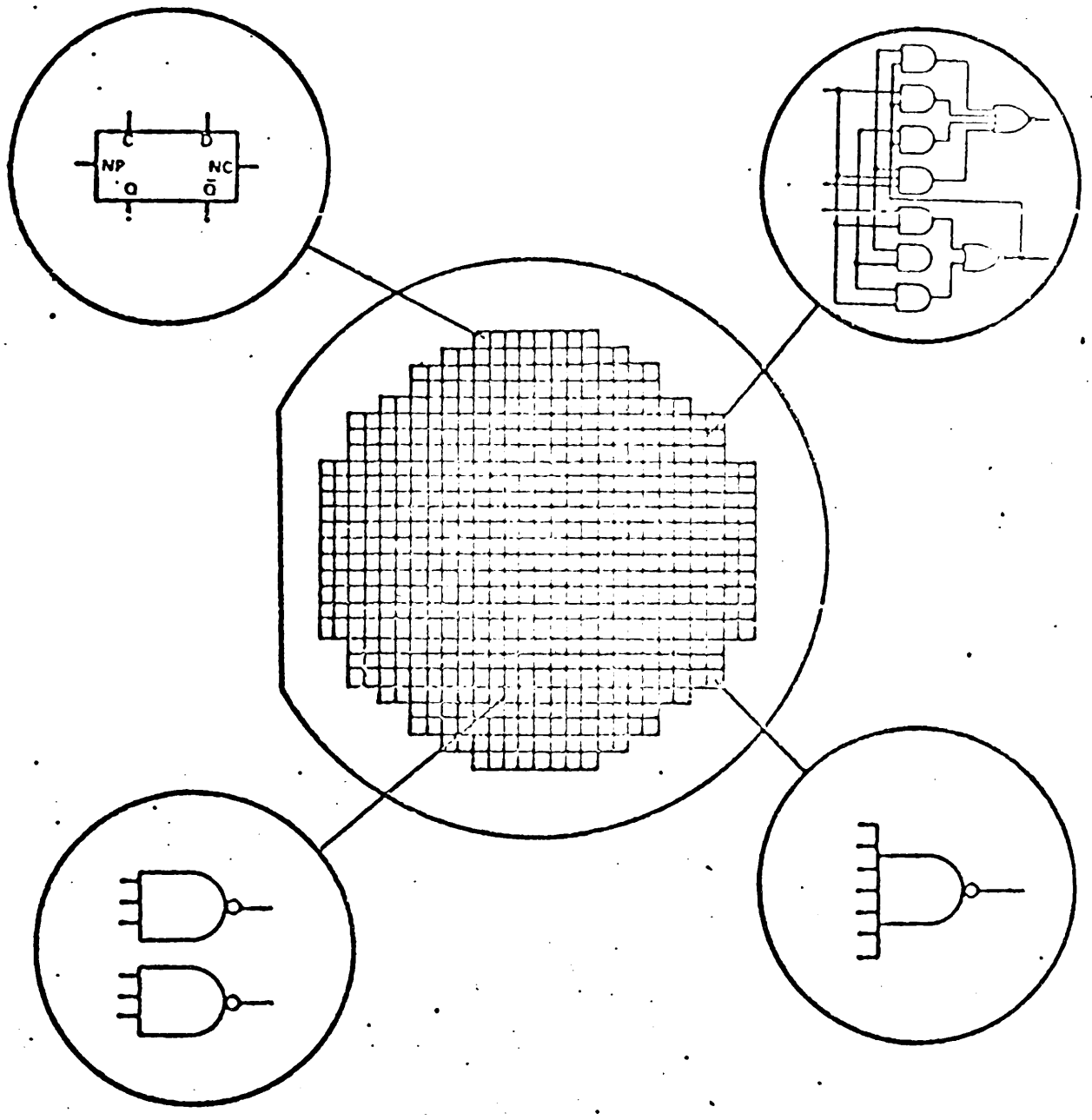


Figure 4.2. Flexibility of AADC Packaging

TEXAS INSTRUMENTS

LOGIC SLICE - TYPE "P"



CIRCUIT TYPE	TOTAL NUMBER	EQUIVALENT GATES
D FLIP-FLOP	116	210
FULL ADDER	182	495
3-INPUT GATE	420	126
7-INPUT GATE	88	26
TOTAL		857

Figure 4.3. A Sample of Present LSI Packaging in a Single Chip

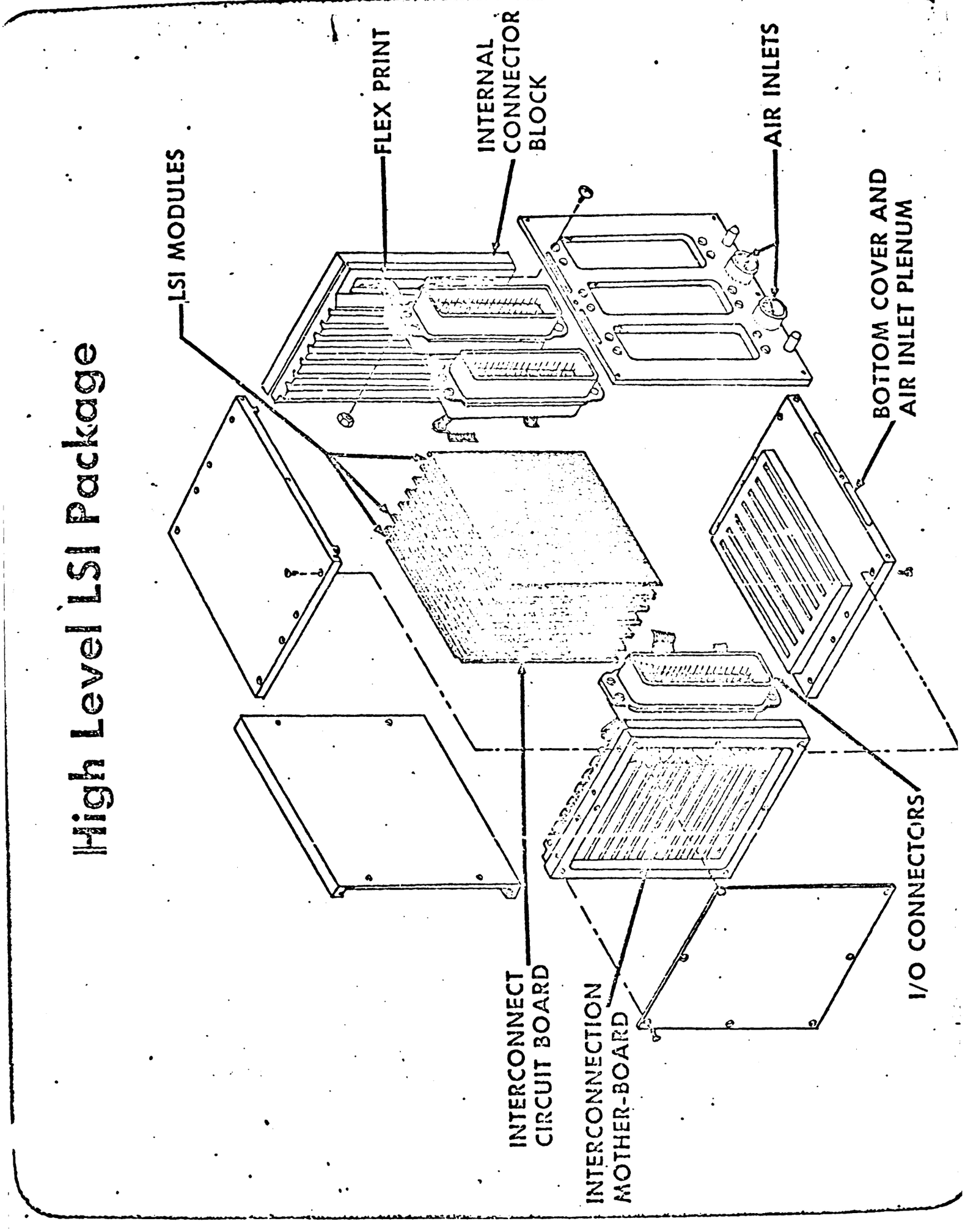


Figure 4.4. High Level LSI Packaging

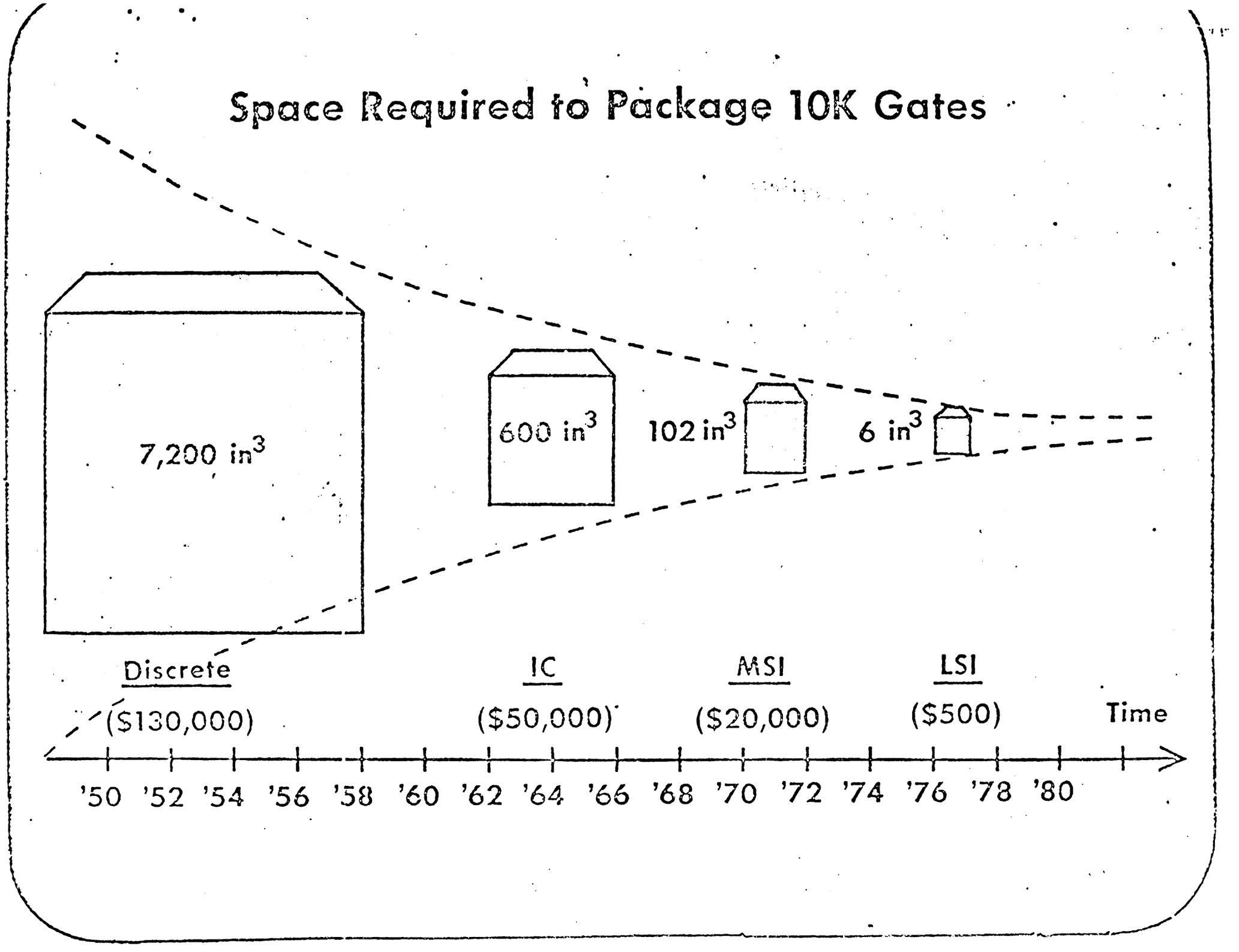


Figure 4.5. Space and Cost for 10K Logic Gates

4.3.2 Semiconductor Technology*

Semiconductor technology, itself, has come a long way since 1969. Among the areas of semiconductor development that the Naval Air Systems Command has supported in the past, or plans to support are [4.5].

1. Materials growth,
2. Electron image projection,
3. Anodic multilevel metalization,
4. Pad relocation for LSI wafers,
5. Eutectic bonding of wafers,
6. Ion implantation,
7. Double insulator semiconductor memory technology.

Four companies, Texas Instruments, Motorala, Monsanto and Tyco Laboratories Inc., all claim they can grow the three-inch diameter wafer necessary for AADC. The Tyco process is particularly interesting because it permits single crystals to be grown in virtually any shape, size or thickness. These single-crystal semiconductors do not require subsequent slicing, which can destroy half the stock; nor do they require polishing - another major source of semiconductor failures [4.6].

*For background information basic LSI technology see [4.28 and 4.29].

The application of electron image projection and multi-level metalization is shown in Figure 4.6. The results of a study by Westinghouse on electron beam projection* are shown in [4.7], and on multi-level flexible film interconnections in [4.8]. They use a selective anodization process which eliminates pinhole breaks in oxide layers [4.6]. The results of a study by Texas Instruments on two-level anodized aluminum interconnections is shown in [4.9]. The advantage of the anodized multi-level process is a reduction in cost of a factor of 5 to 10, and an increase in reliability.

One of the most important developments in LSI semiconductor technology is the development of pad relocation technique to replace discretionary wiring. The difference in the complexity of the mask for the two techniques is shown in Figure 4.7. With MSI and LSI technology not all the logic circuits operate properly, so some method is required for interconnecting the good circuits together and connecting circuits to the outside pins. Instead of testing all the logic circuits and wiring them in the proper order as with the discretionary technique, the new technique, called pad relocation, initially assumes that a certain percentage of the circuits will be good and in particular locations and connects the circuits accordingly. Later when the circuits are tested and found to be in different location than expected, pad relocation connections are made between the actual circuit locations and the assumed good circuit locations.

*Electron beam project is a technique of etching circuit on substrates using electron beams through a mask. Introductory information can be obtained from [4.30 or 4.31].

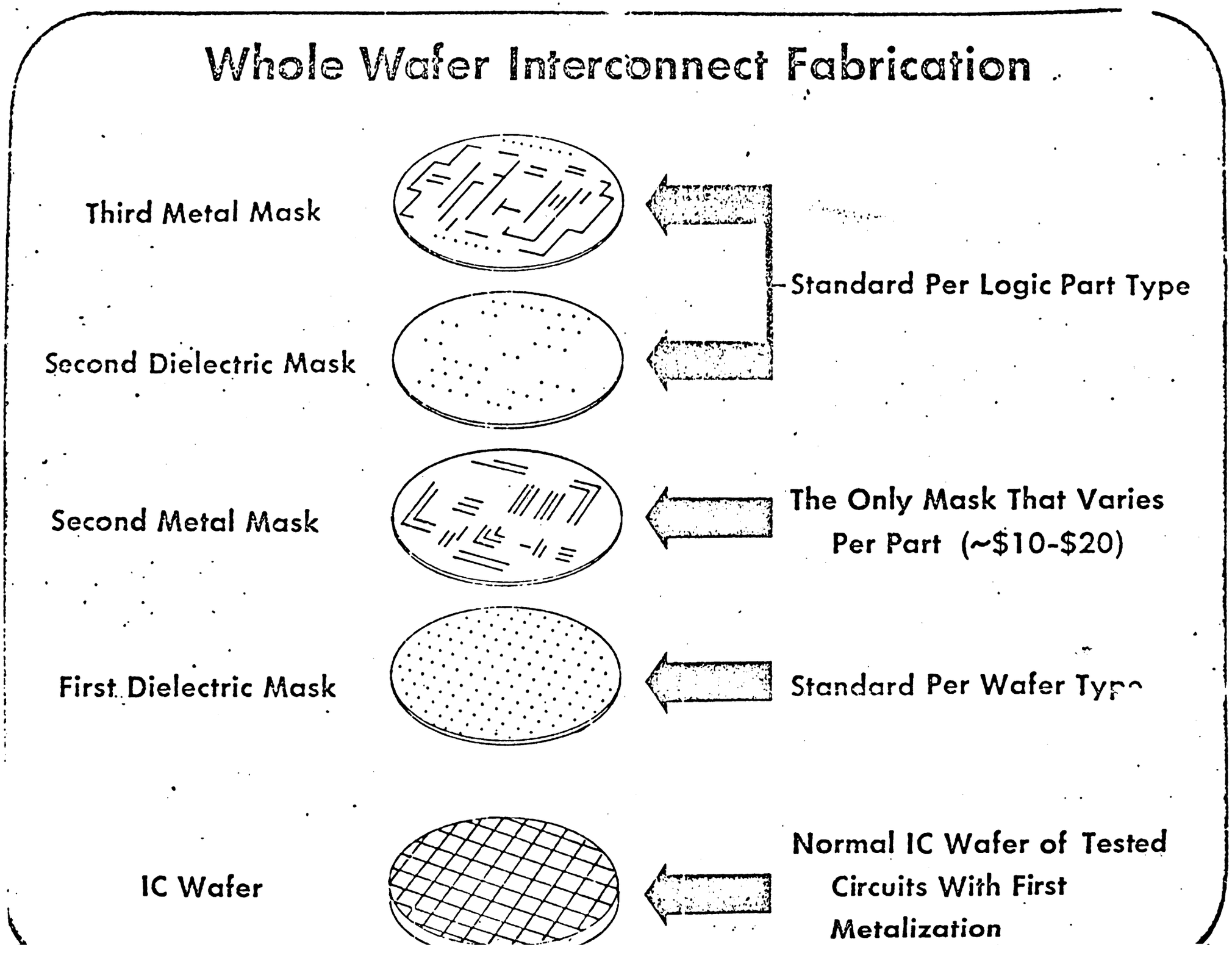


Figure 4.6. Image Projection and Multi-Level Metalization

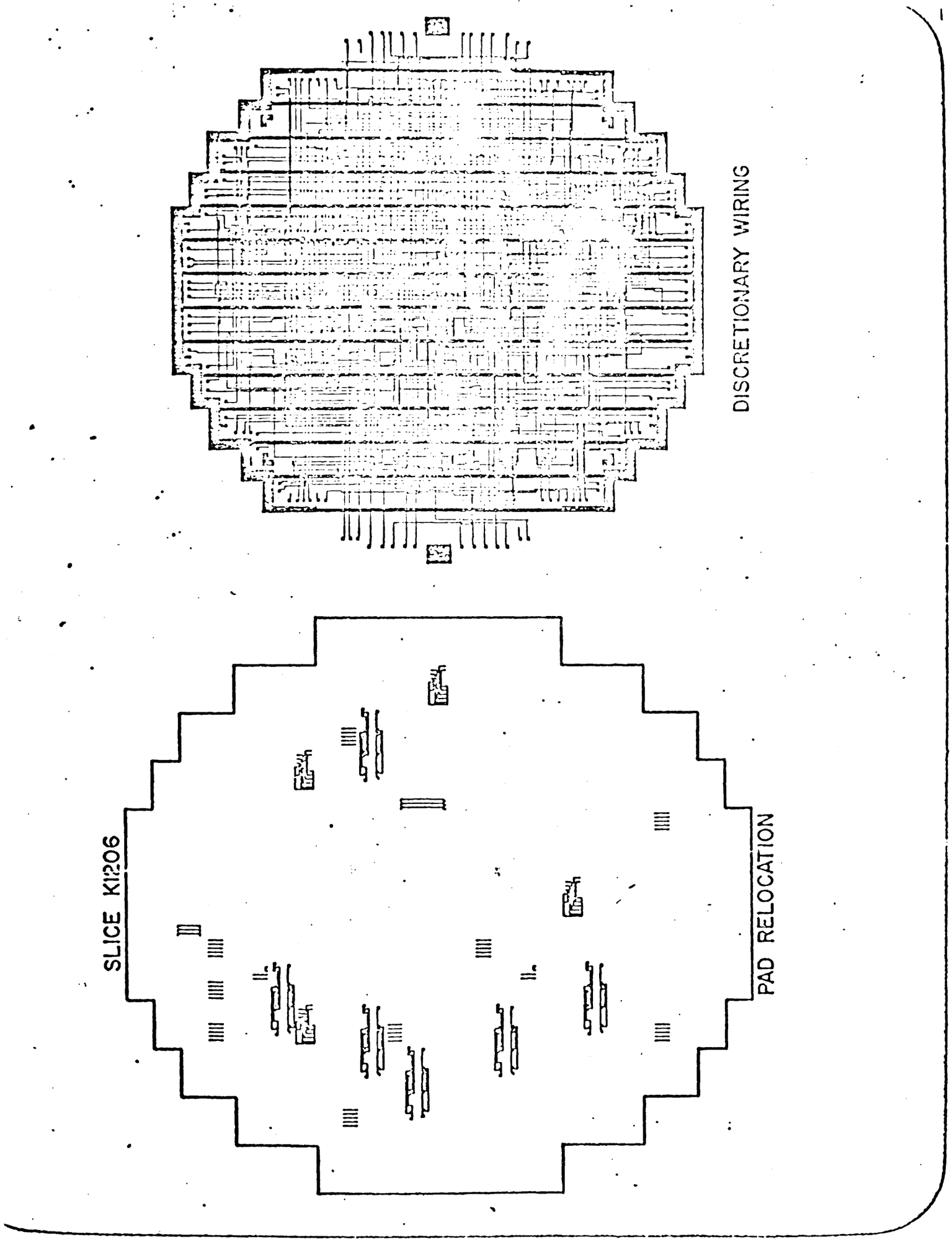


Figure 4.7. Pad Relocation vs Discretionary Wiring

The pad relocation technique is illustrated in Figure 4.8. The slashes in the top left-hand picture show the tested good circuit in the LSI wafer. The circles in the top right-hand picture show the assumed location of the good circuits. The two are superimposed in the lower left-hand picture and the pad relocation specifications are produced by connecting the good circuits to their proper locations in the lower right-hand picture. The insertion of the pad relation layer is shown in Figure 4.6 as the second metal mask. A slightly different explanation is given in [4.6].

With this technique only the pad relocation layer varies with individual LSI wafers and since these are much simplier than the equivalent discretionary wiring interconnections and cost only $10 to $20, this technique makes the production of 5000 gate LSI chips much more realistic. Hughes Aircraft Company have produced two reports on pad relocation [4.10 and 4.11].

Texas Instruments has developed a method of large area wafer bonding using a metal with a very low melting point [4.12].

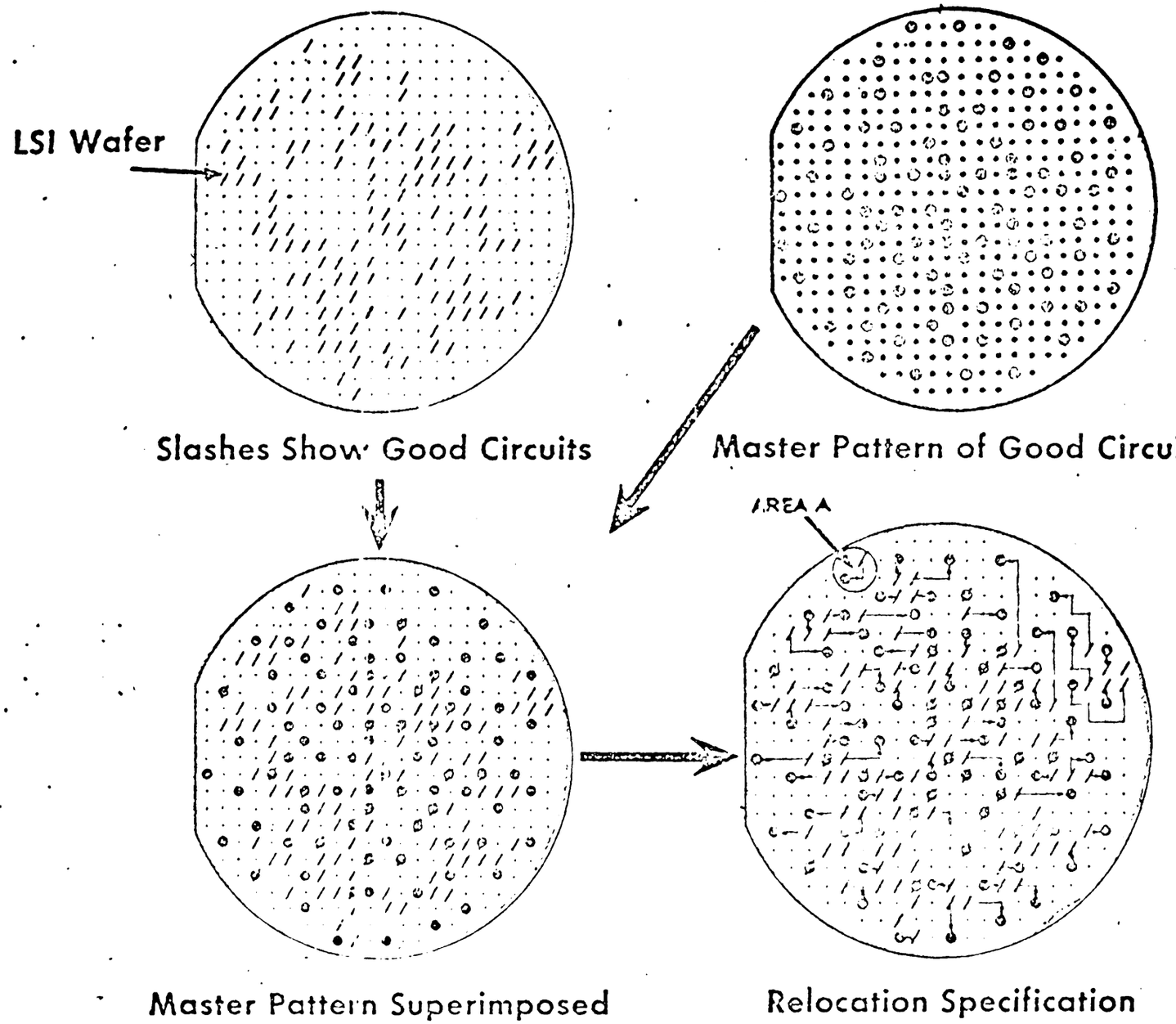


Figure 4.8. Pad Relocation Process

4.3.3 Digital Gate Technology

Table 4.1 is a summary of the speed, power dissipated, cost and size of five different types of digital logic gates. The five types are Metalic Oxide Semiconductor (MOS), Complementary Metalic Oxide Semiconductor (CMOS), Transistor-Transistor Logic (TTL) - probably the most common today -, Emitter Coupled Logic (ECL), and Current Mode Threshold Logic (CMTL). For AADC, the gate transition time should be less than 5 nsec and the power dissipation should be as low as possible. Table 4.1 is taken from [4.15] and is 3 years out of date.

LOGIC TYPE	Tpd (ns)	Pd (^{mw}/gate)	COST	SIZE
MOS	100	.5	LOW	SMALL
CMOS	5	.01	HIGH	SMALL
TTL	15	10	LOW	MODERATE
ECL	.5	50	HIGH	LARGE
CMTL	20	3	MODERATE	LARGE

Table 4.1 Summary of Digital Logic Gates

4.3.4 Innovative Logic Techniques

Two studies have been reported that use non-standard techniques to produce standard logic circuits. The first by Honeywell reports the use of LSI memory techniques to produce universal logic modules [4.13]. The second by RCA reports the use of MOS (Metal Oxide Semiconductor) LSI circuits to produce threshold logic circuit and suggests using these in place of standard logic circuits [4.14].

It is as a result of these other efforts that the future will see single, low cost, semiconductor devices capable of supporting and utilizing more than ten thousand bipolar gates. The existing PE (actually A&C) design, to provide a reference, employs about ten thousand such gates.

4.4 MEMORY TECHNOLOGY

This section will explain in some more detail and give references to the two promising magnetic storage technologies for AADC. Again they are the block oriented ferroacoustic memory for BORAM and the random access closed flux path thin film memory (CFM) for RAMM and TM. This section is a continuation of Subsection 4.1.3. The basic characteristics of the two technologies are repeated here for easy reference.

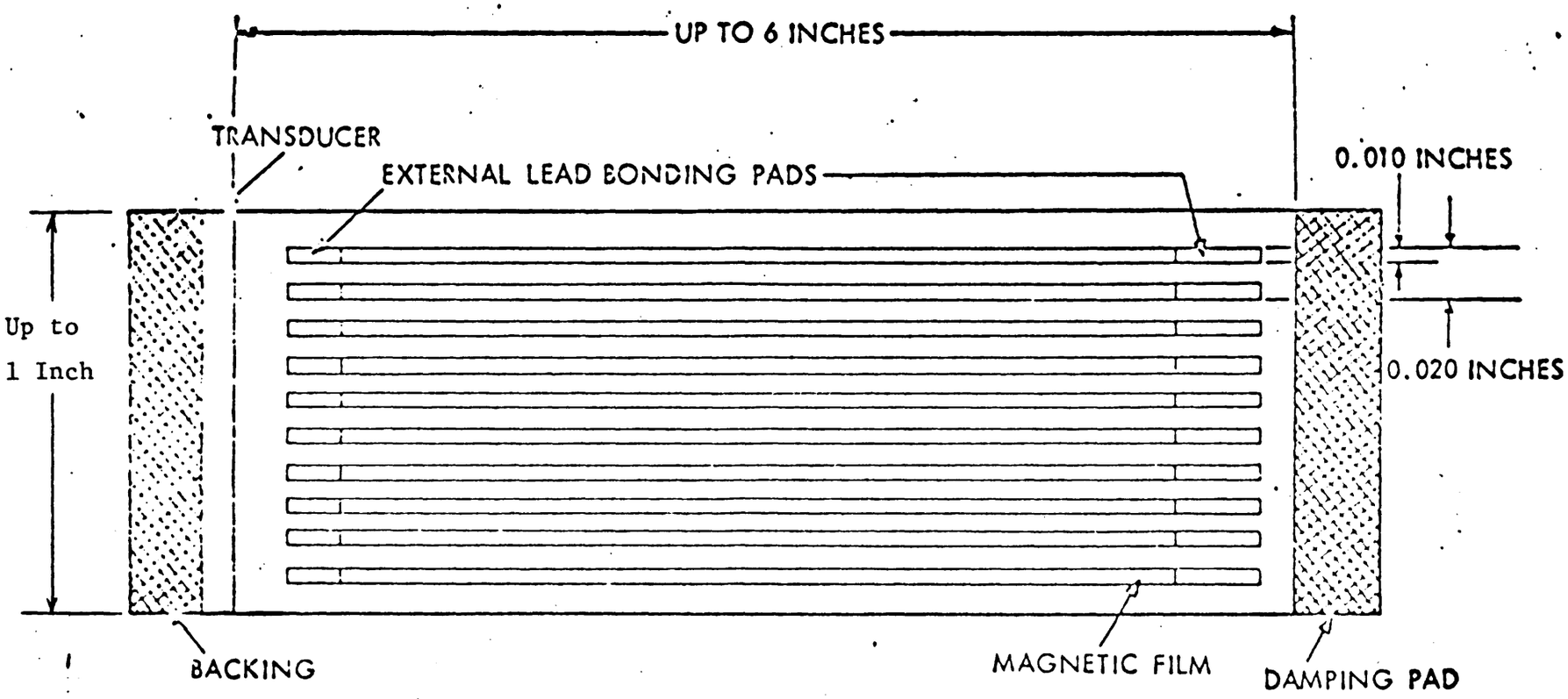
Ferroacoustic technology employs the coincidence of mechanical and electrical energy to write magnetic domains into a permalloy film. These domai are subsequently interrogated by way of an acoustic strain wave. It is low cos high speed, high density, low power, non-volatile and uses the NDRO technique. CFM uses a thin magnetic film which is analog with a magnetic plated wire and offers performance in random access magnetic storage heretofore believed realiz only in semiconductor memories, but without the twin penalties of high power and data volatility. Table 4.2 list the salient features of both magnetic storage technologies [4.1]. By comparison CFM for RAMM and TM is 2 to 30 times more expensive, about twice as fast, occupies as little as half the volume, use 50 times more power, and is 6 times heavier than the ferroacoustic memory for BORAM. The cost, density, power and weight in Table 4.2 includes electronics and power supply.

4.4.1 BORAM

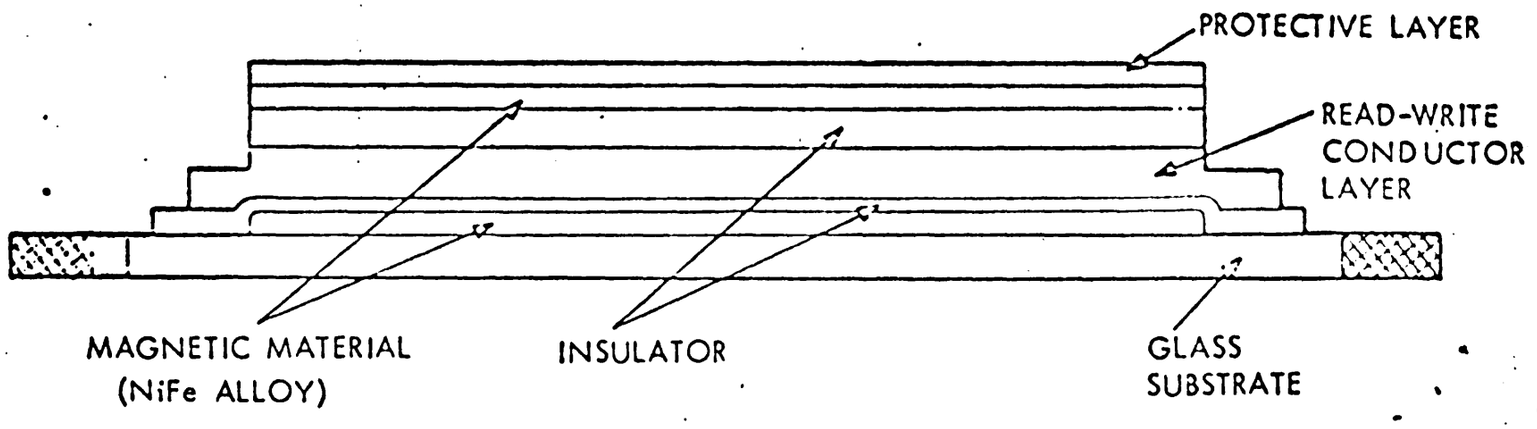
The most promising approach for BORAM is the ferroacoustic memory, in which magnetic domains are written by the coincidence of mechanical and electrical energy and the domains are subsequently interrogated by means of an acoustic strain wave. Figure 4.9 illustrates one form of ferroacoustic memory block, which uses glass for a substrate. There are up to 64 magnetic film conductors across the one-inch wide strip. These permit up to 64 bits of a word to be read or written simultaneously. Plated wire may also be used to

Table 4.2. CHARACTERISTICS OF FERROACOUSTIC AND CFM MEMORIES

Characteristic	Ferroacoustic	CFM
Technology	Closed Flux Path Permalloy	Closed Flux Path Permalloy
Density	5,000 bits/in^3	5,000 to 11,000 bits/in^3
Power	2uwatts/bit	100uwatts/bit
Weight	7.5 lbs (64Kwds x 36bits)	3 lbs (4Kwds x 36bits)
Cost	.1¢ to .5¢/bit	1¢ to 3¢/bit
Access Time	1-2usec to a block	80nsec/wd
Read Cycle Time	150nsec/wd (NDRO)	100nsec/wd (NDRO)
Write Cycle Time	150nsec/wd	150nsec/wd
Interface	TTL	TTL
Volatility	Non-volatile	Non-volatile



(a) TOP VIEW



(b) CROSS SECTION THROUGH FILM STRIPS

Figure 4.9. A Ferroacoustic Memory Plane Employing a Glass Substrate

fabricate a ferroacoustic memory. References [4.16 and 4.17] describe ferroacoustic memories and refer to them as Sonican.

In addition to these technologies, a third magnetic technology, tentatively called Cross Tie Memory, is under investigation for possible application in AADC BORAM. Cross Tie Memory, developed at the Naval Ordnance Laboratory and presented at the last INTERMAG Conference held in Tokyo, Japan, is analogous to Bubble memory, except that it uses an amorphous permalloy substrate, has propogation rates on the order of 100 MHz and does not require an external field to maintain domain wall integrity. Further information on Cross Tie Memory can be obtained from [4.1, paragraph 26].

4.4.2 BORAM for AADC ALL Application Role

The following except from AADC Progress Report No. 10 describes the possibility of using semiconductor memories* for BORAM in the ALL Application Role [4.1, paragraph 4-6].

> 4. Semiconductor Block Oriented Random Access Memory (BORAM): In order to add to the technologies available for construction of BORAMs for procedure and constant storage in versions of AADC for tactical and/or process control applications where non-volatility and read mostly operation are deemed desirable, [other memory technologies are being investigated]. Dual insulator and amorphous semiconductor technologies appear reasonable candidates for this function. For both technologies, write time is far less significant than electrical alterability. Secondly, because these memories are employed in a demand paged hierarchy, fast read cycles can be achieved through word multiplexing. Table 4.3 describes the long term goals for a tactical semiconductor BORAM.
>
> 5. Because of the all application nature of the new AADC, AADC systems will also be used in non-tactical environments such as software preparation centers and system simulation laboratores. Here, the AADC BORAM

*Background information on semiconductor memories can be obtained from [4.32].

TABLE 4.3

DESIRED LONG RANGE SEMICONDUCTOR BULK STORE MEMORY CHARACTERISTICS

1. Organization - Block organized, read mostly (electrically alterable) design with random access to the block level.

2. Storage - Electrically Alterable - The data shall be retained in a non-volatile form and will not be modified by loss of power. The reading process shall be non-destructive (NDRO). The memory array should be capable of handling at least 10^5 writes.

3. Volatility/Retention Time - One year minimum or more (no power applied). No loss of memory data shall occur when power is turned on or off; no special voltage sequencing shall be required to maintain the data stored in the memory.

4. Module Size - 64K words per module - 32 bits per data word and 4 parity bits.

5. Modularity - Each memory module to be self-supporting so that the number of memory words can be increased by the addition of more 64K word modules. It is expected that as many as 8 modules will be harnessed to make a 500K word memory system.

6. Block Size - 128 or 256 word block.

7. Word Size - 32 data bits and 4 parity bits.

8. Word Organization - Word serial bit parallel.

9. Data Readout - The memory system will have the ability to read out a complete block at maximum speed (continuous word stream). It is desirable that the memory organization also permit read out of a block on a interrupted incremental basis,

10. Data Transfer Rate - Write - As dictated by volatility requirements.
Read - 150 nsec or less on system basis.

Multiplexing in read and write modes, to achieve the above speeds, is permissible.

11. Block Access Time - Two usec or less to the first word in any block. The access time shall be defined as the time interval between the instant the block address is received and the instant the first word in the block is available.

TABLE 4.3 (cont'd)

12. Parity - Four parity bits for each data word. Parity logic shall be implemented such that horizontal (word) parity can be checked. Odd parity shall be used. Parity bits shall be available on the output register. Parity shall be checked during read and write operations.

13. Error Rate - 1 bit in 10^{13} bits on one bit basis.

14. MTBF/Reliability - 2.3 million bit (64K) module MTBF shall be 10,000 hours.

15. Radiation Hardness - Similar to that of plated wire memory system.

16. Module Operating Power - 2.3 million bit, 55 watts or less in read or write modes.

17. Module Weight - 2.3 million bit, 6.5 pounds or less.

18. Module Packing Density - 10K bit per in^3 including supporting electronics (less power supplies).

19. Cost - 0.25¢/bit in production.

20. Environment - MIL-E-5400 Class 4X.

21. Electrical Interference - MIL-STD-461A (on modular level).

22. Input/Output - Register shall be provided to accept a 32 bit data word plus 4 bits of parity. The interface shall be TTL compatible. The exact timing and bit allocation will be supplied by the Navy.

23. Packaging - System packaging shall be coordinated with NAVAIRSYSCOM packaging program (AIR-52022D); it is desirable that the technology be amenable to LSI type packaging in a 3-inch diameter hermetically sealed enclosure.

24. Voltages - Effort should be made to minimize types and levels of voltages used in the system; it is desirable that voltage levels be compatible with readily available power supplies.

will require a fast store-back capability. Present non-volitile semiconductor technologies, with the possible exception of MNOS on insulator substrates, may require a write time which exceeds read time by one to two orders of magnitude. They may not, therefore, be useful for these applications. On the other hand, the benign conditions found in a programming center or simulation laboratory may obviate the need for hard non-volitility. If this is true, then a volatile, block oriented MOS or CCD memory with a backup power supply (e.g., a battery) could very easily be used instead. A study may be undertaken next year to examine this new application of MOS device technology. The ferroacoustic memory presently under development for AADC at Microsonics/ Sangamo has a 1:1 read/write ratio. It will serve, therefore, equally well as a tactical and non-tactical BORAM.

6. The responses to the Naval Air Development Center's RFP for semiconductor BORAM were received in late March. Contracts have been negotiated with Litton Guidance and Control and Univac.

(In the above quotation, MNOS refers to Metal N-channel Oxide Semiconductor, MOS refers to Metalic Oxide Semiconductor, and CCD refers to Charge Coupled Logics. Late March in paragraph 6 refers to March 1972).

4.4.3 RAMM and TM

Closed Flux Memory (CFM) uses magnetic recording on a permalloy thin film strip analogous with a plated wire. The CFM is shown diagramatically in Figure 4.10. References [4.18, 4.19 and 4.20] describe CFM memory technology under the name Post-and-Film Memory.

Three other references that are not reviewed here are [4.21, 4.22 and 4.23]. Other references on both BORAM and RAMM can be found in [4.3].

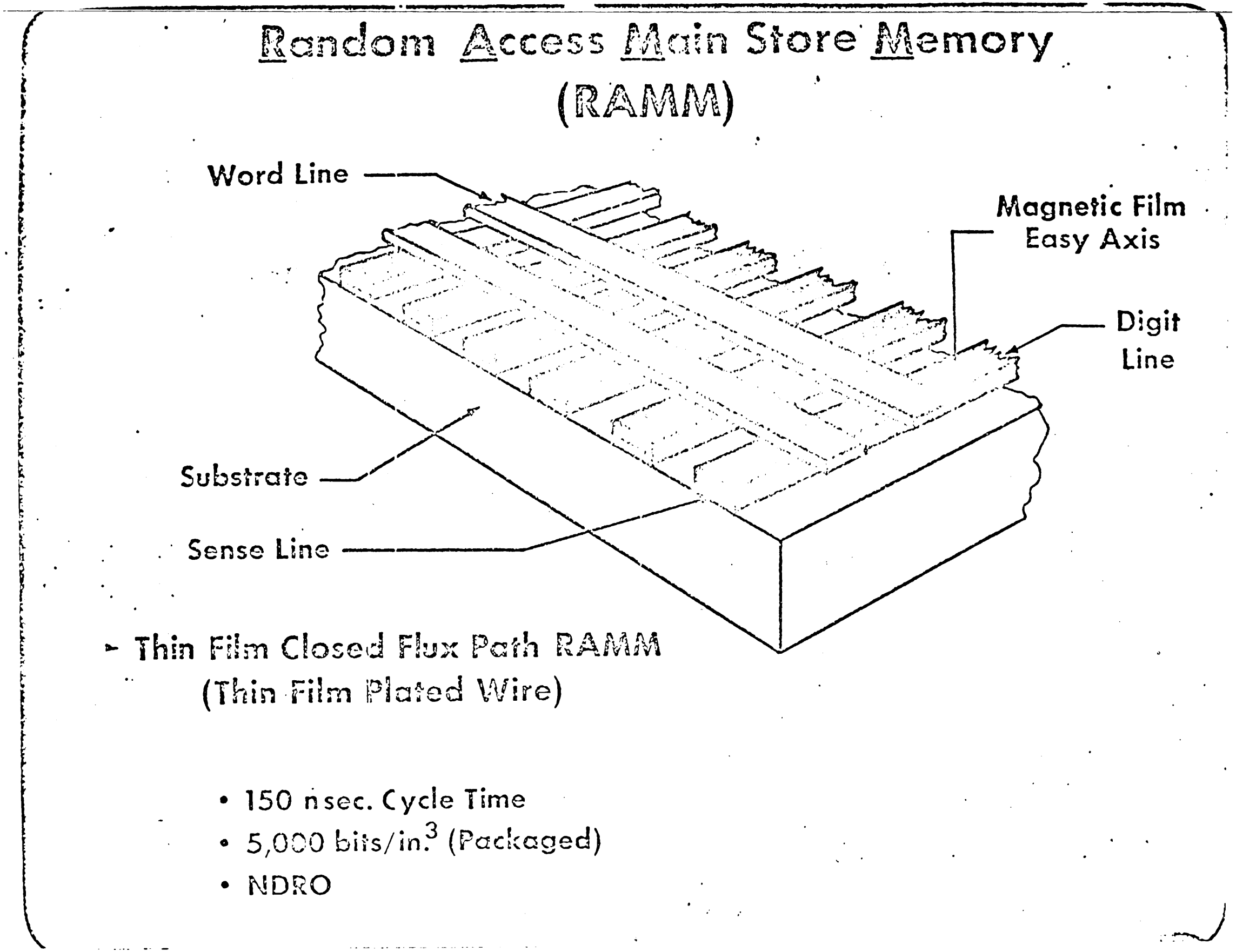


Figure 4.10. Closed Flux Memory for RAMM and TM

4.5 BUSSING TECHNOLOGY*

Because of AADC's very small geometry, modularity and need for very wide bandwidth TDM (Time Division Multiplexing) internal busses, optical communication is being considered seriously as the internal bussing technique. The optical communications offers distinct advantages over all electronic alternatives in the area of noise immunity and ease of connections.

The following excerpt is taken from [4.5]:

> When compared to an all electronic bus implementation, electro-optics appears to have several attractive advantages. These advantages emerge in the areas of noise generation and sensitivity, as well as efficiency and bandwidth. In the realm of interface, too, optical connections may be more easily achieved since they don't require coaxial connectors which are cumbersome, expensive and notoriously unreliable. Furthmore, by taking advantage of the bandwidth afforded by electro-optics, signals may be multiplexed to result in fewer physical lines. This last advantage may prove key to the economic feasibility of an optical communication system, since the fiber optics required to build these buses will probably be the single most expensive element in such data links. This cost can be directly attributed to the physical complexity of multiported, duplex fiber optics. In the AADC, the multiport requirement stems from the need to support a floating executive in the event of a primary MEC failure.
>
> Figure 4.11 illustrates a Simplex Optical Bus of a type which might be used to provide communications from the AADC BORAM. In this system, parallel organized data enters from the left and is immediately converted into a serial bit stream. These bits are then coded, using Manchester or a similar self-clocking code, in order to provide bit synchronization for the data receiver. The encoded signal is then injected into a fiber optic waveguide by means of a Light Emitting Diode. On the other end, the optical data is detected by a Light Detecting Diode, decoded and then converted back into a parallel bit stream. To reduce system costs, one such detector might be used to service a cluster of two or three Processing Elements.

---

*Background information on bussing technology see [4.33].

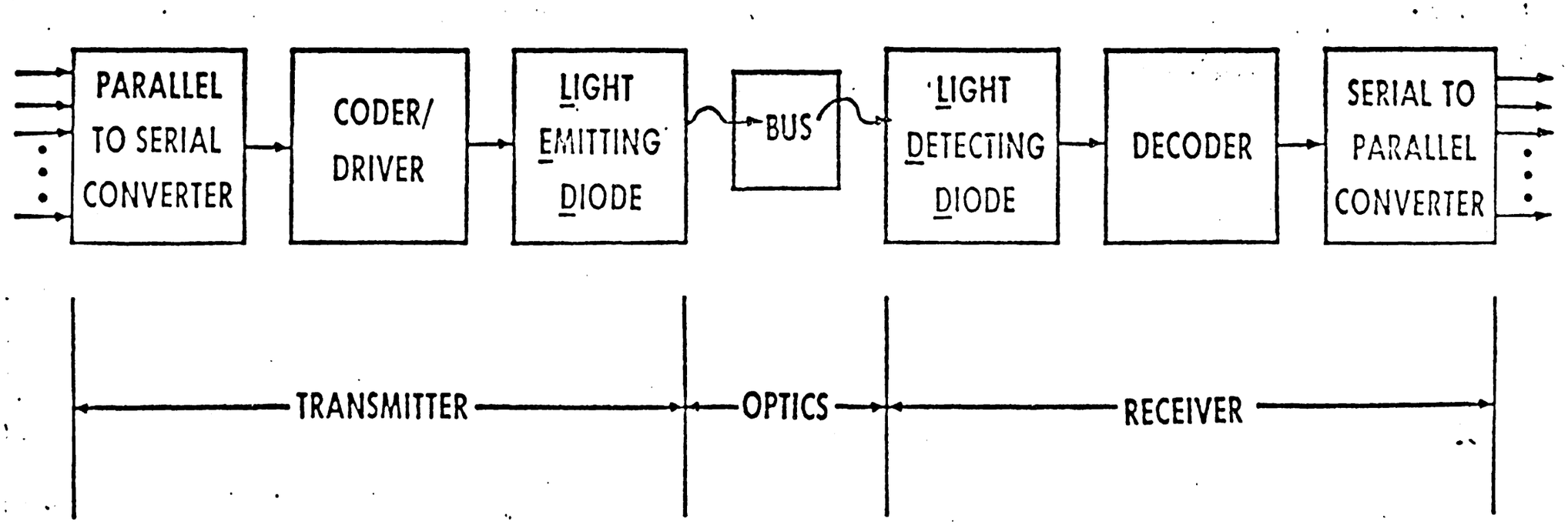


Figure 4.11. The Simplex Optical Bus for AADC

4.6 ELECTRIC POWER SYSTEM

The proposed electric power distribution system for the AADC is to replace the conventional electro-mechanical relay system with an improved power generation and semiconductor control system. The new system, known as Solid State Electric Logic (SOSTEL), will greatly improve the control of electrical power, reduce power consumption, reduce wiring and reduce weight. The reduction in power consumption is the result of leaving some equipment on standby power most of the time and using the very fast switching time of semiconductor logic to apply full power when required.

The current status of Solid State Electric Logic power distribution system is described in [4.24 or enclosure 3 to AADC Progress Report No. 9] and in the Proceeding of SOSTEL Symposium in April 1971 [4.25].

This concludes the presentation of the hardware technology. Many of the diagrams in this chapter are taken from a slide presentation by Ron Entner [4.26]. The author would welcome any suggested improvements in the material in this chapter.

References to AADC Hardware Technology

4.1 AADC Development Program Progress Report No. 10; R. S. Entner, NAVAIRSYSCOM; May 31, 1972; (78, NPS).*

4.2 The New LSI Components; Marcian E. Hoff; Intel Corp.; Digest of Papers for COMPCON 72; IEEE Catalog No. 72CH0659-3C; September 12-14, 1972; (NPS).

4.3 Proceedings of the Advanced Digital Technology Conference - Volumes 1 and 2; Naval Air Systems Command; June 8-10, 1971; (55, NPS).

4.4 Large Scale Integrated Circuit (LSI) Module and Higher Level Packaging Study for the AADC Program - Final Report; Singer Aerospace and Marine Systems; December 3, 1971; NAVAIRSYSCOM Contract No. N00019-70-C-050; (73).

4.5 The Advanced Avionics Digital Computer Revisited; R. S. Entner; NAVAIR-SYSCOM; October 12, 1971; Unpublished paper; Unclassified; (NPS).

4.6 Navy Engineers Break the Rules with Radial Airborne FDP Concept; Electronics; August 3, 1970; pp 89-90; Unclassified; (35, NPS).

LSI Technology

4.7 Electron Beam Image Projection to Fabricate Large Arrays of Integrated Circuit Patterns (U); Westinghouse Research Laboratories; NAVAIRSYSCOM Contract No. N00019-71-C-0066; March 1972; (84).

4.8 Hybrid Multilevel Flexible Film Interconnection (U); Westinghouse Defense and Electronic Systems Center; NAVAIRSYSCOM Contract No. N00019-71-C-0383; March 1972; (85).

---

*AADC Bibliograph number, and availability at the Naval Postgraduate School.

4.9 Engineering Study of Development of Two-Level Anodized Aluminum Interconnects for MSI and LSI; W. R. McMahon and B. G. Carbajal; Texas Instruments Inc.; NAVAIRSYSCOM Contract No. N00019-70-C-0487; September 1971; (83, NPS).

4.10 Development of Automated Pad Relocation LSI, Final Report; Hughes Aircraft Company No. P70-392; September 1970; NAVAIRSYSCOM Contract N00019-70-C-0013; Unclassified-NOFORN; (38, NPS).

4.11 Development of Automated Pad Relocation LSI, Phase 11 - Final Report; J. R. Hall, R. K. Cleghorn and B. B. Bennett; Hughes Aircraft Company; September 1971; NAVAIRSYSCOM Contract No. N00019-70-C-0608; (82).

4.12 Engineering Study of Development of a Large Area Eutectic Wafer Bonding Process (U); Texas Instruments Inc.; NAVAIRSYSCOM Contract No. N00019-71-C-0363; April 1972; (86).

4.13 Universal Logic Modules Implemented Using LSI Memory Techniques; Ken J. Thurber and Robert O. Berg, Honeywell; Proceeding of FJCC 1971; pp 177-194; November 1971; (NPS).

4.14 MOS Threshold Logic; D. Hampel and J. B. Lerch; RCA Government Communications Systems; NAVAIRSYSCOM Contract No. N00019-70-C-0604; January 1972; (88).

4.15 Advance Avionics Digital Computer Hardware Considerations; A. David Klein; NAVAIRSYSCOM; September 1969; Unclassified; (13, NPS).

4.16 SONISCAN - A Sonically Accessed Magnetic-Film Memory; H. Rubinstein, R. Hornreich and J. Teixeira; Proceedings of the 1970 IEEE International Computer Group Conference; June 16-18, 1970; pp. 64-72; Unclassified; (32).

4.17 Soniscan Bulk Store Memory for the Advanced Avionic Digital Computer; GTE Sylvania Inc.; June 11, 1971; NADC Contract No. N62267-70-C-0217; Unclassified; AD-885-807; (56).

4.18 Post-and-Film Memory Deliveries NDRO Capability, Low Noise, High Speed, but Avoids Problem of Creep; R. Vieth and C. Womack; Electronics; January 1970; Unclassified; (21).

4.19 Post-and-Film Memory Development Program Summary Report; Litton Systems Inc.; February 27, 1970; NAVAIRDEVCEN Contract N62269-69-C-0239; Unclassified-NOFORN; AD-868-335; (22, NPS).

4.20 Post-and-Film NDRO Memory Element Development - Final Report; Data Systems Division, Litton Systems, Inc.; February 1971; NADC Contract N62269-71-C-0024; AD-881-741; (51).

4.21 Advanced Techniques in Airborne Computer Memories, Final Report; Sylvania (GTE); February 28, 1970; NAVAIRDEVCEN Contract N62269-69-C-0430; Unclassified-NOFORN; AD-867-468; (25, NPS).

4.22 Advanced Computer Memory Program at NAVAIRDEVCEN; Roman Fedorak; NADC; March 16, 1970; Unclassified. (Enclosure to Progress Report No. 5); (29, NPS).

4.23 High Density Main Store Memory Unit, Final Report; Ampex Corporation; August 26, 1970; NAVAIRDEVCEN Contract N62269-70-C-0216; Unclassified-NOFORN; AD-875-363; (36, NPS).

Electrical Systems

4.24 NAVAIR R&D Program in Aircraft Power Systems for the 1970's; Leonard W. Wendling; NAVAIRSYSCOM; Undated; Unclassified; Available as Enclosure 3 to Progress Report No. 9; pp 59-89 [1.33]; (NPS).

4.25 Symposium on Advanced Aircraft Electric Systems (SOSTEL) Proceeding; Leonard W. Wendling; NAVAIRSYSCOM; April 20-22, 1971; Available for cost of mailing from L. W. Wendling, Project Manager; (NPS).

General

4.26 Slide Presentation on AADC for FY72; R. S. Entner, NAVAIRSYSCOM; Undated, probably spring 1972; Unpublished; Unclassified; (NPS).

4.27 All Application Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceedings not yet available.

4.28 Technical Advances in Large Scale Integration; H. T. Hochman and D. L. Hog Honeywell Inc; IEEE Spectrum; May 1970; pp 50-58.

4.29 Integrated - circuit Digital Logic Families - ECL and MOS Devices; L. S. Garrett; IEEE Spectrum; December 1970; pp 30-42.

4.30 The Application of Electron/ion Beam Technology to Microelectronics; G. R. Brewer; IEEE Spectrum; January 1971; pp 23-37.

4.31 Microcircuits by Electron Beam; A. N. Broers and M. Hatzakis; Scientific American; November 1972; pp 34-44.

4.32 Semiconductor Ramdom-Access Memories; L. L. Vadasz, H. T. Chua, A. S. Grove, Intel Corporation; IEEE Spectrum; May 1971; pp 40-48.

4.33 Communication Channels; Henry Busignies; Scientific American; September 1972; pp 98-106.

Chapter 5

DATA PROCESSING ELEMENT

Table of Contents

List of Figures

List of Tables

Glossory of Terms for PE

A&C - Arithmetic and Control Unit: same as PE.

ALU - Arithmetic Logic Unit: the unit that performs the actual addition, subtraction or logic function on the operands. A subcomponent of the AU. (Raytheon does not use "ALU" but uses "AU" for both the AU and the ALU.)

AP - Arithmetic Processing Execution Unit: executes arithmetic instructions and includes the AU, TVD, deferral unit and programmable control unit.

Ap - Accumulator Stack Pointer: see Section 5.2.5.

APL - Iverson's APL language - contain many very powerful operations especially for vector, matrix and array manipulations.

APQ - Arithmetic Processor Queue: 16 32-bit registers for stacking instructions between PMU and AP.

AP - Scratchpad 16 40-bit deferral registers or accomulators in the AP.

AU - Arithmetic Unit: the heart of the AP and is composed of PAU, SAU, APQ, AP Scratchpad, six registers and control and transfer circuitry.

Aw - Working Accumulator.

CPU - Central Processing Unit.

Ep - External Device Stack Pointer: see Section 5.2.5.

FAU - Fetch Arithmetic Unit or PMU arithmetic Unit: a simplified arithmetic unit to handle address calculations, etc.

HOL - Higher Order Language such as extended CMS-2 or extended FORTRAN.

LSI - Large Scale Integration technology.

MIPS - Millions of instructions per second.

PAU - Primary Arithmetic Unit: actually should be called the PALU for Primary Arithmetic Logic Unit.

PC - Parenthesis Control - instruction action is delayed until all data is available.

PE - Processing Element: the main serial processor of AADC, usually called the CPU. Now called DPE for Data Processing Element.

PF - Parenthesis Field specified beginning of parenthesis control (=1111) or the number of parenthesis to be closed.

PMU - Program Management Instruction Handling Unit: Instruction fetching and control unit of PE.

PMUSP - The PMU scratch pad: 8 index registers plus 4 stack pointers.

Pp - Program Counter Pointer: See Section 5.2.5.

SAU - Secondary Arithmetic Unit: actually should be called SALU; a simplified version of the PAU that is used in 4-bit-at-a-time multiply.

TM - Task Memory: 4K, 32 bit 150 nsec memory.

TVD - A comparison test valid mechanism for setting the sign bit of the accumulator depending on the result of a comparison operation.

DPE - Data Processing Element: new name for the Processing Element to distinguish it from the Signal Processing Element (Chapter 7).

CHAPTER 5

DATA PROCESSING ELEMENT

5.1 INTRODUCTION AND SUMMARY

The AADC Processing Element (PE)* is a very fast, very powerful, very small and very inexpensive central processing unit (CPU) designed for large scale computing systems. It is one of the basic AADC modules and is designed to handle all the serial processing requirements of AADC. It is capable of executing 2.5 to 4 million instructions per second (MIPS), with effective processing rates of 8 to 10 MIPS. Its power is the result of the hardware implementation of a general deferral mechanism and numerous powerful operations, especially the polynomial, matrix and vector operations. Most importantly, this fast powerful processor is packaged in an eight inch cube (0.5 cubic feet) and has an estimated production cost of $600. (As a comparison the CPU on the IBM 360 model 67 - a third generation large scale computer - executes about 0.3 to 0.5 MIPS, does not have the same powerful instructions, occupies about 125 cubic feet and costs $698,000.) This section will present an overview of the PE features, while later sections will include a more detailed presentation.

In order to obtain the desired speed it was necessary to overlap the fetching of instructions and their executions. The instruction fetching operates at 2.5 MIPS including an indexing operation and 3.3 MIPS without indexing. Since the PE is a Task Memory oriented element, the need for indexing is greatly reduced over previous computer designs, and the latter speed is more appropriate. These speeds are based on a memory cycle time of 150 nanoseconds (nsec). On the other hand, the instruction execution takes 100 nsec for short instructions (equivalent to Adds) and 800 nsec for fixed-point multiplications. With an assumed ratio

---

*Now called DPE for Data Processing Element to distinguish it from the SPE - Signal Processing Element - described in Chapter 7. PE and DPE are used interchangeably.

of 7 short instructions to 3 multiplications, the instruction execution rate of 3.3 MIPS is also possible. Since the proposed floating point multiplications are faster than the fixed point, the instruction execution rate with floating point operations is 4.0 MIPS.

The overlapping of instruction fetching and program execution is obtained by dividing the PE into a Program Management Instruction Handling Unit (PMU) and an Arithmetic Processing Execution Unit (AP). The two subsystems operate independently and asynchronously permitting the PMU to fetch instructions well ahead of their execution, and while the AP is processing previously fetched instructions. This is generally referred to as "look-ahead," where instructions are prefetched along the most probable branch path. If the results of a branch instruction are not along the expected path, then the stockpile of instructions is discarded and instruction fetching is initiated along the other path. To hold the stockpile of instructions, a sixteen-register queue connects the PMU with the AP.

The power of the AADC PE is demonstrated by the fact that it has many very powerful instructions, many of which are not even available in higher order languages and certainly not implemented in hardware on a general purpose computer. For example, the PE has the following features implemented in hardware:

1. All 16 possible boolean functions,
2. A recursive subroutine call capability,
3. A general deferral mechanism that executes arithmetic, boolean and conditional expressions directly without reordering the operations or using excessive storing and fetching of intermediate results,

4. A rapid polynomial calculation capability for trigonometric, logorithmic, hyperbolic and expcnential functions (all coefficients are loaded by a block transfer.),
5. Vector/matrix block handling mechanism for 15-component vectors and small 3 x 4 matrices.

The particular significant of these features to the programmer is that, (1) the general deferral mechanism allows the mixing of arithmetic, boolean and conditional expressions in a single statement - providing the accompanying higher order language is upgraded -, and (2) the vector/matrix mechanism allows operations such as the vector dot product and the matrix product to be specified in two machine language statements. In both these cases the higher order language will have to be upgraded beyond FORTRAN or CMS-2 before that language can use these powerful machine language (or hardware) features.

As well as being very fast and powerful, the PE is very small and inexpensive. A rough estimate of the PE logic is:

1.	The AP (arithmetic processor)	6,000 gates,
2.	Basic PMU (control unit)	1,000 gates,
3.	Queue between PMU and AP	1,000 gates,
4.	Parentheses control and vector/ matrix mechanism	1,000 gates,
5.	Instruction decoder and controller	1,000 gates,
	Total	10,000 gates.

These 10,000 gates are placed on two 3-inch diameter LSI chips and housed in an 8-inch cube having a total volume of 0.5 cubic feet. It is also estimated that the production cost of the PE will be about $600. Rather unbelievable?

If this design is achievable at this cost, or even at 100 times this cost, then it is going to be the biggest breakthrough in computer hardware development since the transistor. In order to achieve the maximum benefit from this new development, many of the programming aids, such as very powerful operators and extensive debugging features that were previously too expensive to implement will now have to be included in the design. Otherwise the AADC PE will be almost immediately replaced with another computer containing these extra programming aids.

This section would not be complete without some comment on the feasibility and current status of the PE. At present LSI 1-1/2-inch diameter chips with 1000 to 1500 gates are being produced at a cost of about $1000 each. The set-up costs, including drawing all the circuits, is about $50,000 for each different type of chip. (Ref. Dr. Ray N. Nilsen, University of California, Los Angeles). Also the CPU for the SUE computer - a small scale microprogrammed computer - is built on two LSI chips and costs less than $1000.

Although this section is written as though the PE actually exists, it must be realized that it is based on design specifications only and that even these are still under development. The information in this section is based almost exclusively on Raytheon's report [5.1]. A later 1972 report has been produced but is not yet available at NPS. Section 5.8 describes some of the latest PE developments as reported at the January 1973 AADC Symposium [5.4].

5.2 FUNDAMENTAL SYSTEM CHARACTERISTICS

5.2.1 Instruction Speed

In order to obtain more than 2 MIPS with current technology it is not possible to fetch instructions and execute them sequentially, but instead, it is necessary to overlap these two operations. Thus the PMU for fetching instructions and the AP for executing them operate autonomously and asynchonously. The average time to execute instructions depends on the longer of the average time to fetch the instructions and the average time to execute them, assuming the execution (or AP) never has to wait for instructions.

Assuming a 150 nsec memory cycle time for the task memory, the time for the PMU to fetch an instruction and its operand is (approximately):

Instruction fetch	150 nsec
Indexing operation	100 nsec
Operand fetch	150 nsec
Total	400 nsec

Thus the PMU operates at 2.5 MIPS with indexing and 3.3 MIPS without indexing. Since the PE is a task memory oriented processor with a relatively small 4 K-word memory, the latter speed is more realistic.

To calculate the execution speed a ratio of 7 short instructions (equivalent to adds) to 3 long instructions (defined as multiplies) is assumed. This is worse than the 8 to 2 ratio observed in present Navy avionics programs. In order to operate at 2.5 MIPS reqaires the short instructions take 125 nsec while the long ones take 1000 nsec. To operate at 3.3 MIPS requires that the short instruction take 100 nsec and the long one take less than 800 nsec. Raytheon believes that these speeds are realistic for current technology.

Although the PMU and AP operate autonomously and asynchronously and both operate at 3.3 MIPS, this is no guarantee that they will produce a throughput of 3.3 MIPS on a given problem. Since the AP queue is limited to 16 instructions, any time there are twelve short instructions in a sequence or four long instructions in a sequence, then the AP has to wait because the APQ is empty or the PMU has to wait because the queue is full, respectively (assuming APQ was initially half full under steady state conditions). In general whenever the short-term mix ratio is not 7 to 3, then either the PMU or AP has to wait and the throughput decreases.

As will be discussed in the next section, a feature called the Parenthesis Control, will reduce the number of store and fetch operations by 50 percent. According to Raytheon about 50 percent of all instructions in the analyzed Navy programs involved the storing and fetching of intermediate results. The reducing of this non-functional overhead means that AADC programs will be 25 percent shorter and thereby the effective throughput will be 4.4 MIPS ($\frac{100}{75}$ x 3.3). Furthermore, since many of the PE instructions are equivalent to macros on existing computers, the effective throughput will probably be doubled again to 8 to 10 MIPS when compared to conventional third generation computers. This is 15 to 30 times faster than an IBM 360/65.

5.2.2 Parenthesis Control

Parenthesis Control (PC) is a PE feature which enhances the relationship between the problem specification in a Higher Order Language and its execution on the PE. Parenthesis Control was originally developed to handle the parenthesis portion of algebraic equations, but has now been expanded to handle conditional and logical expressions. In essence, it automatically defers

program actions until such time as sufficient information (or data) is available to complete them. This obviates the need for compiler rearrangement of the input stream and eliminates many redundant stores and fetches of intermediate results - thus reducing the complexity of the compiler and increasing the executing speed of the generated code.

The basic principle of the Parenthesis Control is that parenthesis are given equal weight with op codes (functions) and operands; i.e., all affect the order of execution. By deferring action until the data is available, PC reduces the number of single address instructions required to perform an algebraic task to an absolute minimum (one instruction per operand).

The order of execution follows the normal algebraic procedure being read left to right with two exceptions:

1. Multiply/divide's are performed before add/subtracts,
2. Parenthesis take precedence over other operations,

   i.e. they say, "Don't do this now, execute what is inside parenthesis and then come back and do this."

In practice, each of these exceptions is classified as a "deferred action" when it arises and are handled in the same way by the computer.

Since the computer operates in the left to right sequential preference instead of the multiply-add preference, the expression A + BC must be presented the computer as A + (BC) to distinguish it from (A + B)C. A standard compiler, when presented with the first expression above, would invert the order of execution. However, using Parenthesis Control, the computer would handle the terms in the correct sequence by the following procedure:

1. load the value of A
2. defer the addition until the product of B and C is formed, then perform the deferred addition.

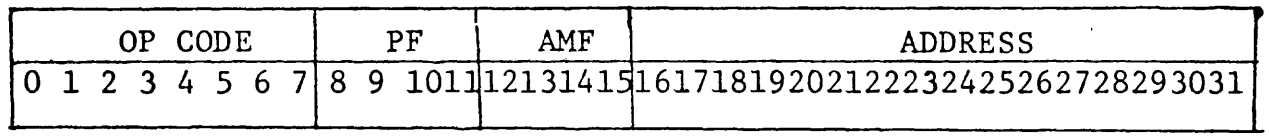
This is the procedure used in the PE but, before the actual implementation can be discussed, it is necessary to describe the PE instruction format.

5.2.2.1 Instruction Formats

The word size chosen for this machine is 32 bits. The basic format for the majority of computer instructions has been designated as Format 1,* below.

FORMAT I

AP ADDRESSABLE



-In this format, bits 0-7 are termed the OP-CODE. They specify the type of operation to be performed. The OP-CODE is specified in hexadecimal notation.

-Bits 8-11 are termed the parenthesis field.** All instructions using Format 1 are subject to Parenthetical Control.

-Bits 12-15 represent the address modification field (AMF). Specifically, bit 12, when set specifies that indirect addressing is to be performed to obtain the effective address of the operand. Bits 13-15 specify a PMU Scratch Pad register for the automatic indexing operation.

---

*In [5.1] an R field was specified but it has been eliminated because the ADDRESS field had to be increased to 16 bits.

** Now only bits 9-11, with bit 8 being the data precision bit.

-Bits 16-31 represent the primary address. The 16 bits in this field are capable of directly referencing 64K words in the virtual Task Memory. The contents of this field, as modified by indexing and indirect addressing, become the effective address of the operand.

The Parenthesis Field (PF) in the above format contains four bits. One combination of these bits (0000) specifies no parenthetical action, i.e., the specified operation is performed immediately on the data. Another combination (1111) specifies that this instruction begins a parenthesis, i.e., that the operation is to be deferred. In no case is it necessary to begin more than one parenthesis, but the user may write more than one.

The remaining 14 hexadecimal combinations of PF specify the number of parentheses to be closed, i.e., how many deferred operations can be completed at this time. Thus, a maximum of 14 parentheses can be closed at any time, and a total of 15 accumulators can be involved with a single instruction.

The PE contains a scratchpad memory containing 16 accumulator locations, each 40 bits long: 32 bits for data and 8 bits to hold the op code for deferred operations.* A four-bit address register is also used to specify which accumulator is the current working accumulator. [The current working accumulator is sometimes designated as Aw in examples given.]

Using this structure, the parenthesis field code can be used to sequence instructions as follows:

1. "No parenthesis code (PF = 0000)" - operation specified by the instruction word is performed immediately.

---

* Now 41 bits with the extra bit being for data precision bit.

2. "Begin Parenthesis (PF = 1111)" - The operand is loaded into the next sequentially numbered accumulator and the operation specified by the op code is not performed, but the full op code (8 bits) is also stored in this accumulator. In addition, the contents of the old accumulator is preserved for later use.
3. "End N parentheses (PF = N)" - the specified instruction is performed with the present accumulator, then the most recent deferred operation (OP CODE stored in the working accumulator) is performed with the first answer as one operand and the most recently stored accumulator as the other. The process is repeated N times.

5.2.2.2 Comparing PC With Other Methods

Consider an elementary expression: X = AB + CD + EF executed on a single address machine with a single accumulator and with multiple accumulator. The best possible compiler could not produce code better than shown in the left two columns of Table 5.1.

SINGLE ACCUMULATOR	MULTIPLE ACCUMULATOR	PARENTHESIS CONTROL
Load A	Load A in Acc 1	Load A
Multiply B	Multiply B in Acc 1	Multiply B
Store AB	Load C in Acc 2	Add (C
Load C	Multiply D in Acc 2	Multiply D)
Multiply D	Add 2 to 1	Add (E
Add AB	Load E in Acc 2	Multiply F)
Store AB + CD	Multiply F in Acc 2	Store in X
Load E	Add 2 to 1	
Multiply F	Store in X	
Add AB + CD		
Store in X		

Table 5.1 Instructions for X = AB + CD + EF

The multiple accumulator computer saves two store operations and two accesses to main memory by performing two register-to-register operations instead of two memory-to-register operations. Thus it saves 2 instructions and four memory cycles.

With Parenthesis Control the number of instructions is reduced by two more to one per operand - the minimum possible. The program would look like the following, where Aw, A_1 and A_2 are the working, the first and the second accumulator contents, respectively:

Load A	Aw = A_1 = A
Multiply B	Aw = A_1 = AB
Add (C	Aw = A_2 = C[Add Held], A_1 = AB
Multiply D)	Aw = A_2 = CD[Add Held], A_1 = AB, then Aw = A_1 = AB + CD
Add (E	Aw = A_2 = E[Add Held], A_1 = AB + CD
Multiply F)	Aw = A_2 = EF[Add Held], A_1 = AB + CD, then Aw = A_1 = AB + CD + EF
Store X	

In this case the multiply D and the multiply F both cause two arithmetic operations to be performed. Thus the program to calculate this expression takes only 7 instructions with PC compared to 9 with multiple accumulators and 11 instructions with a single accumulator and no PC - a saving in program size of 22 and 36 percent, respectively. Although the load on the PMU will be reduced by these percentages, the reduction is the total execution time will be minimal because of the unfavorable mix ratio. The execution times shown in Figure 5.1 shows a saving of only 5 percent (assuming the loading of a register and an add both take 100 nsec and a multiply takes 800 nsec).

Single Accumulator	PMU	L , M , S , L , M , A , S , L , M , A , S ,	
	AP	,L, , M ,S,L, M ,A,S,L, M ,A,S,	3.7 nsec
Multiple Accumulator	PMU	L , M , L , M , A , L , M , A , S ,	
	AP	,L, , M ,L, M ,A,L, M ,A,S,	3.5 nsec
With PC	PMU	L , M , A , M , A , M , S ,	
	AP	,L, , M ,L, M ,A,L, M ,A,S,	3.5 nsec

Figure 5.1 Times for $X = AB + CD + EF$

As a further example, a program for the expression

$$A(BC + DEF + G(HI + JK) + L)$$

is illustrated in Table 5.2. Note that the final result is in the working register A_1 while registers A_2 through A_5 are left with intermediate results.

TABLE 5.2

PROGRAM AND REGISTER CONTENTS FOR
A(BC + (DEF) + (G(H I + (JK))) + L)

STEP	INSTRUCTION	PF	Aw	A_5 EXT	A_4 EXT	A_3 EXT	A_2 EXT	A_1
1	LOAD A	000	A_1					A
2	MUL (B	111	A_2				B MUL	A
3	MUL C	000	A_2				BC MUL	A
4	ADD (D	111	A_3			D ADD	BC MUL	A
5	MUL E	000	A_3			DE ADD	BC MUL	A
6	MUL F)	001	A_2			DEF ADD	BC + (DEF) MUL	A
7	ADD (G	111	A_3			G ADD	BC + (DEF) MUL	A
8	MUL (H	111	A_4		H MUL	G ADD	BC + (DEF) MUL	A
9	MUL I	000	A_4		HI MUL	G ADD	BC + (DEF) MUL	A
10	ADD (J	111	A_5	J ADD	HI MUL	G ADD	BC + (DEF) MUL	A
11	MUL K)))	011	A_2	JK ADD	HI + (JK) MUL	G(HI+JK) ADD	BC + (DEF) + G(HI + JK) MUL	A
12	ADD L)	001	A_1	JK ADD	HI + (JK) MUL	G(HI+(JK)) ADD	(BC + (DEF) + (G(HI + (JK)))+L MUL	A(BC + (DEF) + (G(HI + (JK))) + L)

NOTE: Although only Add and Multiply operations are shown, the process can be used with any combination of arithmetic operations, such as addition, subtraction, multiplication, and division.

Before continuing, one problem with PC that has been completely ignored by Raytheon will be discussed. Although the restriction of a single left parenthesis is not a limitation, it does generate some problems in interpreting user written expressions. For example the expression

$$(((AX + B)C + D)E + F)G$$

written by a user would have to be presented to the computer as

$$AX + BC + DE + FG$$

for straight forward execution from left to right. Here is the problem: The computer now has to execute this expression ignoring the normal precedence of multiplication over addition - which is the reason for the user inserting the brackets. The dilemma is that if the computer executes the expression left to right giving precedence to parentheses only, then the users cannot use the normal algebraic precedences and he must insert the necessary brackets. On the other hand, if the users are allowed to write expressions with the normal algebraic precedence, then a scanner - as part of the compiler - must insert or delete parentheses as required. Another example may help clarify this delemma.

Consider the example $A + B \times C \uparrow D$ where $\uparrow$ represents exponentiation (which is very conspicuously ignored in Raytheon's report). If the user presents the expression to the computer in the above fashion assuming the computer knows about algebraic precedences and if there is no scanner, the expression will be executed as $((A + B) \times C) \uparrow D$ instead of $A + (B \times (C \uparrow D))$. If there is a scanner it can insert the necessary parentheses; otherwise the user must know the computer executes left to right and insert the necessary parentheses himself. On the other hand, if the user wanted the expression executed as $((A + B) \times C) \uparrow D$, he would probably insert the parentheses as shown and a scanner would have to

remove them for presentation to the computer. Alternately, the user must know about the right to left rule and must never insert two left parentheses together. This decision is actually one of how sophisticated are the users expected to be; for the unsophisticated user the normal algebraic precedence is always better, but for the sophisticated user the left to right rule is much more general and explicit. Raytheon did not make this decision and it is still undecided.

The same PC mechanism for evaluating algebraic expressions is also used to evaluate Boolean expressions. In order to minimize the number of instructions in evaluating a Boolean expression, all sixteen possible functions of two variables are implemented (See section 5.6.2.4 for a listing). Boolean operators have short execution times (similar to add) and this further reduces the total execution time for evaluating Boolean expressions. Examples of evaluating Boolean expressions are included in the next section.

5.2.3 Combining Arithmetic, Boolean and Conditional Expressions

The Parenthesis Control concept described in the previous section can also be used to handle Boolean and conditional expressions in an HOL. First the comparison process in a conditional expression will be presented before considering a combined example.

The comparison process is broken into two separate parts or instruction elements - COMPARE instructions and TRANSFER instructions.

1. The COMPARE instructions are AP addressable types, similar to arithmetic instructions. Each one specifies an operand and a condition. The operand is compared with the contents of the accumulator for the specified condition. If the test is valid, a special monitoring unit, called the Test Valid mechanism (TVD), sets the sign bit of the accumulator positive, otherwise it is set negative. (The comparison operation can be any of the six possible standard comparisons.)
2. The TRANSFER instruction observes the status of the sign bit of the accumulator. If the sign bit agrees with the condition specified in the TRANSFER instruction Op Code, the PMU is interrupted and the branch is effected. (APQ is also cleared.) Otherwise, normal program sequencing continues.

Thus the PMU continues fetching instructions along the most probable branch path and filling the APQ while waiting for the results of the test part. If the branch is required, the PMU is interrupted, the APQ is cleared and the PMU begins fetching instructions along the other path. This look-ahead along the most probable path allows the programmer and the compiler to generate very efficient loops, since the execution normally transfers within a loop several times before executing a single transfer out of the loop.

Notice that this method of mechanizing comparison operations requires two instructions for each comparison. This is the price for the look-ahead capability.

To execute the HOL statement

IF A > B, GO TO M, ELSE, CONTINUE.

where the commas are simply separators, the PE program would be:

Load A	A is placed in the accumulator
CGR B	B is compared with A. If A is greater than B, then the sign bit of the accumulator is set positive, otherwise it is set negative.
TRP M	Transfer on accumulator positive (i.e. the test was valid) to M, otherwise continue processing.

An example of a more complex expression using PC is:

IF A > B AND (C ≠ (DE - F)), GO TO M, ELSE, CONTINUE.

for which the program is shown in Table 5.3.

STEP	INSTRUCTION	PF	Aw	A_1	A_2	A_3
1	Load A	0000	A_1	A		
2	CGR B	0000	A_1	A > B		
3	AND (C	1111	A_2	A > B	C AND	
4	CNE (D	1111	A_3	A > B	C AND	D CNE
5	MUL E	0000	A_3	A > B	C AND	DE CNE
6.	SUB F))	0010	A_1	A > B AND (C ≠ (DE - F)	C ≠ (DE - F)AND	DE - F
7	TRP M	0000	A_1			

Table 5.3 Boolean and Conditional Program and Register Contents

In step 2, A_1 is set by the condition A > B and thus the sign bit of A_1 is positive or negative. The remainder of the program should be self explanatory. It is recommended that the reader try an example such as

IF A > B AND (C > D) OR (E > F), GO TO M, ELSE, CONTINUE.

Note the AND has precedence over the OR and thus extra parentheses are not necessary. Also notice that all these examples contain sufficient parentheses so that there is no ambiguity over whether the arithmetic, logical or conditional operators have the highest precedence. (It would make sense to have the arithmetic operators with the highest precedence, conditional operators next and logical operators with the lowest; but, on the other hand, the straight left to right precedence is the simpliest. Apparently no decision has yet been made.)

The major advantages of Parenthesis Control are:

1. It ensures the minimum number of instructions by eliminating many needless load and store orders.

2. It reduces the complexity of the compiler by eliminating the need to rearrange terms in an expression.
3. Program sequence remain in the original algebraic order thereby producing a more understandable listing and reducing the side-effect errors. NO REARRANGEMENT OF TERMS IS EVER NECESSARY, unless all 16 accumulators are full.
4. It allows the mixing of algebraic, Boolean and conditional expressions in the same statement.

5.2.4 Deferred Store Instruction

To remain consistant with the "as written" or left-to-right program execution as defined above, and to allow the standard assignment statements like A = B + C instead of the more accurate B + C → A, it is necessary to define a deferred store operation. The expression A = B + C becomes A(= B + C) and is programmed as:

```
DST  A          A (actually the address of A) is stored in
                a deferral register or accumulator, DST is held.
Load B
Add  C)         Add C and perform deferred operation DST.
```

The deferred store operation has the advantage of allowing assignment statements within assignment statements, which can often reduce the recomputing of sub-expression and make a more readable program. This ability has even been left out of most HOL in the past.

5.2.5 Stacking Mechanism

The design of the PE includes a set of Task Memory pointers which can be used for a variety of reasons including a hardware stack. There are four pointers as part of the 12 scratch pad registers in the PMU (the other eight are for index registers). They are defined as,

1. External Device Pointer (Ep)
2. Program Counter Pointer (Pp)
3. Accumulator Pointer (Ap)
4. Unspecified.

In addition to instructions to load and store pointers, two instructions are implemented for manipulating the stack:

1. Advance Accumulator Stack (AAST) causes the contents of the accumulator to be stored in memory location specified by value in Ap. The value of Ap is incremented. The next AAST will cause the accumulator to be stored in the next sequential memory location.
2. Return from Accumulator Stack (RAST) causes the contents of memory location specified by the decremented value of Ap to be loaded in the accumulator. The decremented value of Ap is placed in the Ap register.

The PC mechanism incorporates the ability to generate an AAST instruction whenever the number of right parentheses exceed the number of left parentheses. Thus the expression A + B) will be implemented by: Load A

Add B)

which causes the sum of A and B to be stored automatically in the accumulator stack in Task Memory by generating in interrupt AAST instruction. This saves an extra store instruction.

Similarily the APST and RPST instructions cause program branching by advancing the program stack (i.e., placing contents of the program counter into the memory location specified by Pp and incrementing Pp) and returning from the program stack (i.e., placing the contents of the memory location specified by the decemented value of Pp into the program counter), respectively. The program pointer Pp, is used to stack previous values of the program counter when branching by using the APST. Thus subroutine returns are easily facilitated and programs may be nested, or called recursively, without danger. (Note AAST and APST are implemented as a single instruction with a different PMU register specified in a 4 bit field in the instruction.)

The external stack is used to facilitate certain I/O word-at-a-time transfers.

5.2.6 Two-Address Instructions

Although two and three address instructions were called for in the original RFP (Request For Proposal) and two address instructions are described by Raytheon, they are not considered seriously for the PE. The advantage of two address instructions is that both a load and an arithmetic operation or an arithmetic and a store operation can be specified in a single instruction; but the disadvantage is that with a 32 bit word it is not possible to maintain PC and still refer to any location in Task Memory with both the primary and secondary operand addresses. Being able to refer to only part of memory with the secondary address is a terrible programming restriction. Therefore, according to Raytheon, single address instructions with PC are superior to two address instructions.

5.2.7 List Linkages - Search Techniques

Since many avionics problems involve a scattered set of linked operands, an easily altered linkage mechanism is implemented in the PE. The mechanism allows indirect addressing with the primary address as the beginning of a table and the secondary address as some location in the table, as well as, addressing of tree structured data (such as used by Burrough for structuring arrays on the B5500). The list linkage mechanism can be used with the stacking mechanism and with comparison instructions for searching lists. For further details on the operation of the list linkage mechanism, refer to [5.1, p 2-27 to 2-30].

5.2.8 Self-Defining Subroutines

With the aid of the APQ, it is possible to define self-modifying subroutines that have some instructions modified while in the APQ while others remain fixed. By loading the queue, specifying the number of words to be modified and controlling the positioning of the queue address pointer, it is possible to execute a routine such as $A(B + C(D + E)) \rightarrow F$ for several sets of operands without reloading the instructions. If the operands are sequentially ordered in the Task Memory then they can be retrieved by simple indexing; otherwise they may be retrieved using the list linkage mechanism described previously.

5.2.9 Polynomial Computations

Many mathematical functions are, or can be, expressed in terms of power series or polynomials. Some of these functions, including sine, cosine and tangent, logarithm and antilogorithm, and their associated power series are shown in Table 5.4.

Upon close observation all functions listed have the general form:

$$Y = A_o + A_1x + A_2x^2 + \ldots + A_nx^n = \sum_{k=0}^{n} A_kx^k,$$

which can be written as:

$$Y = A_o + x(A_1 + x(A_2 + \ldots x(A_{n-2} + x(A_{n-1} + xA_n)\ldots)),$$

which would require n deferral registers to be computed, or as

$$Y = A_nx + A_{n-1}x + A_{n-2}x \ldots + A_2x + A_1x + A_o.$$

in which it can be executed directly in left-to-right order as a series of multiply-then-add operations.

TABLE 5.4

FUNCTION POWER SERIES

(Sheet 1 of 2)

POWER SERIES:

$$\text{SIN } x = 0 + 1x + 0 - \frac{x^3}{3!} + 0 + \frac{x^7}{7!} + 0 + \frac{x^9}{9!} + \ldots \qquad x^2 < \infty$$

$$\text{COS } x = 1 + 0 - \frac{x^2}{2!} + 0 + \frac{x^4}{4!} + 0 - \frac{x^6}{6!} + 0 + \frac{x^8}{8!} + 0 + \ldots \qquad x^2 < \infty$$

$$\text{TAN } x = 0 + x + 0 + \frac{x^3}{3} + \frac{2x^5}{15} + 0 + \frac{17x^7}{315} + 0 + \frac{62x^9}{2835} + \ldots \qquad x^2 < \pi^2/4$$

$$\text{SIN}^{-1} x = 0 + x + 0 + \frac{x^3}{6} + 0 + \frac{3x^5}{2\cdot4\cdot5} + 0 + \frac{1\cdot3\cdot5}{2\cdot4\cdot6} \cdot \frac{x^7}{7} + 0 + \frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8} \cdot \frac{x^9}{9} + \ldots \quad x^2 < 1$$

$$\text{COS}^{-1} x = \pi/2 - \sin^{-1} x \text{ or } = \sin^{-1} \sqrt{1 - x^2} \qquad x^2 < 1$$

$$\tan^{-1} x = 0 + x + 0 - \frac{1}{3}x^3 + 0 + \frac{x^5}{5} + 0 - \frac{x^9}{9} + \ldots \qquad x^2 < 1$$

$$\text{SINh } x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} + \ldots$$

$$\text{COSh } x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} + \ldots$$

$$\text{TANh } x = x - \frac{1}{3} x^3 + \frac{2}{15} x^5 - \frac{17}{315} x^7 + \frac{62}{2835} x^9 - \ldots \qquad x^2 < \pi^2/4$$

$$\text{SINh}^{-1} x = x - \frac{1}{6} x^3 + \frac{1\cdot3x^5}{2\cdot4\cdot5} - \frac{1\cdot3\cdot5}{2\cdot4\cdot6} \frac{x^7}{7} + \ldots \qquad x^2 < 1$$

$$= \log(2x) + \frac{1}{2\cdot2x^2} - \frac{1\cdot3}{2\cdot4\cdot4x^4} + \frac{1\cdot3\cdot5}{2\cdot4\cdot6\cdot6x^6} - \ldots \qquad x > 1$$

TABLE 5.4

FUNCTION POWER SERIES

(Sheet 2 of 2)

POWER SERIES (Cont):

$$\text{COSh}^{-1}\, x = \sinh^{-1} \sqrt{x^2 - 1} \qquad x > 1$$

$$\text{Tanh}^{-1}\, x = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \ldots$$

$$\log_e x = 0 + (x - 1) - \frac{1}{2}(x - 1)^2 + \frac{1}{3}(x - 1)^3 - \ldots \qquad 0 < x \leq 2$$

$$X = 1 + \ln x + \left(\frac{\ln x}{2!}\right)^2 + \left(\frac{\ln x}{3!}\right)^3 + \ldots \qquad \text{where } \ln = \log_e$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots$$

$$e^x - e^{-x} = 2\left(x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots\right) = 2 \sinh x$$

$$e^{ix} + e^{-ix} = 2\left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots\right) \qquad \text{where } i = \sqrt{-1}$$

$$\sin x = \frac{1}{2i}(e^{ix} - e^{-ix}) \qquad \sinh x = \frac{1}{2}(e^x - e^{-x})$$

$$\cos x = \frac{1}{2}(e^{ix} + e^{ix}) \qquad \cosh x = \frac{1}{2}(e^x + e^{-x})$$

Therefore, a hardwired polynomial instruction can be implemented by multiplying by x and adding each coefficient into a partial answer iteratively until the final answer is obtained.

Only one register is required. This instruction can compute any of the following functions:

1. Simple trigonometric functions,
2. Arc (or inverse) of simple trigonometic functions,
3. Hyperbolic functions,
4. Arc of hyperbolic functions,
5. Logorithms,
6. Antilogorithms, and
7. Natural exponential.

These functions are implemented by a Format 1 instruction called PLY for Polynomial of Accumulator, in which the R field* specifies the number of terms in the polynomial to obtain the desired accuracy and the address field specifies the first location for the sequentially located coefficients for the particular function. The operands are block loaded into the APQ, thus freeing the PMU for other processing.

If the amount of TM used to store the coefficients is too large, it may be possible to take advantage of the fact that the hyperbolic functions use the same coefficients as the other trignometry functions except for a sign change on some coefficients. Also it would be possible to calculate all the trigonometric and hyperbolic functions in terms of the exponential series but at reduced speed.

*Eliminated in later design [5.4].

If the PMU cannot be used for other processing, then there is little advantage in this type of mechanism because the fetching of operands is slower than the multiply-add sequence. However, it may be possible to perform I/O or to load other tasks into the Task Memory during this liberated fetch time, thereby increasing the efficiency and justifying the hardwired polynomial computation.

5.2.10 Vector/Block Modes*

The original PE specifications required that PE be microprogrammed to act as backup to the matrix and array processor should the need arise, but Raytheon suggests that, with little hardware cost, it is possible to make the PE hardware handle all common vector and matrix processes. This section presents a fully integrated scheme for solving all common vector and matrix problems with a simple mechanism and maximum efficiency. The operations which are performed by the mechanism on limited sized vectors and matrices include:

1. Vector or matrix add or subtract,
2. Vector dot product and vector magnetude,
3. Matrix multiply,
4. Calculate determinants and cofactors,
5. Invert a matrix and solve simultaneous equations, and
6. Transpose a matrix.

Probably the most significant feature of this mechanism is that any of the operations can be specified by only two machine language instructions, thus freeing the PMU for other activities. This is better than most HOL since matrix operations usually must be specified by element-by-element manipulations in one or two loops. (With these powerful machine language instructions, it is mandatory

*This entire subsection is based on [5.1] and does not take into account the 16 bit ADDRESS field or the new 256-word array capability [5.3 and 5.4].

that very powerful matrix and vector manipulation features be added to the HOL, such as those in the APL language. This is a subject for discussion in Chapter 7.)

Basically, the performance of all the vector and matrix operations requires an available storage area of 16 registers capable of holding data and operation codes. Although these registers could be different than those for PC, it is assumed that they are the same. In fact, using the same registers for PC and Vector/Block mode may reduce execution speed when both types of operations are in the same statement, but this is not considered serious.

The availability of 16 registers generally restricts the Vector/Block mode to vectors of length 15 and 3 x 3 matrices, although the exact number depends on the operation. The actual restriction is that the number of components stored in the AP scratchpad plus the number of temporary answer registers must be less than or equal to 16. Thus the following maximum-sized operations can be done:*

1. The addition or subtraction of two 15 components vectors or matrices since no answer registers are required,
2. The multiplication of two 15-component vectors since only one temporary register is required,
3. An $N \times M$ matrix times an $M \times P$ matrix where $N(M + 1) \leq 16$, because N times M locations are required to store the matrix and N for temporary answer registers, (For example a 3 x 4 times a 4 x P matrix takes 15 locations, whereas a 4 x 3 times a 3 x P matrix takes 16 locations.),

*According to Raytheon's presentation at 1973 AADC Symposium [5.4], these restrictions are no longer valid since the accumulator stock in TM automatically stores and reloads accumulator registers. The current restriction is an array must be less than 256 elements.

4. Any operation with 3 x 3 matrices.

The reasons for these restrictions should be more understandable after the next section.

5.2.10.1 Vector and Matrix Adds and Multiplies

This subsection describes the instruction format and the PE operation for the vector and matrix add and multiply instructions.

Since, in the PC operation, the combination of PF and R both not equal to 0000 is meaningless, the open parenthesis code (PF = 1111) with the R field* containing a value N is now given the following meaning:

1. The PE enters the Vector/Block Mode of order N and each subsequent instruction, until the mode is terminated, is assumed to be a function of N operands.
2. The N operands located in successive memory locations specified by the effective address are sequentially stored in the AP registers (starting at the current working accumulator) and the instruction op code is stored with each.

Once in the block mode there are three parameters that control the execution as follows:

1. The value in the PF field in subsequent instructions establishes the number of temporary answer registers to be used and thereby establishes the number of operation pairs (current op code + deferred op code) that are perform with a particular second operand (the one not in the scratchpad),

*No longer valid since R field was eliminated in later version [5.4].

2. The value of N, as specified on entry to the block mode, establishes the number of operation pairs that are performed (or the number of operands stored in the AP scratch pad that are used) before a block cycle is completed.

3. The value in the R field establishes the number of block cycles that are to be repeated. In other words, it is the number of times the operations are repeated on the scratch pad set using different sets of second operands. If $R \neq 0$, the contents of the answer registers are stored in the accumulator stack in memory via an AAST interrupt after each block cycle of N operands. After R repetitions the mode is terminated. If $R = 0$, the answers are not stored and the Vector/Block mode is not terminated.

The use of these parameters is explained further with the following discussion of particular vector and matrix operations.

The vector (or matrix) add or subtract is specified by the following two machine language instructions.

1. The first vector or matrix, A, of N components is loaded into the AP scratch pad (or deferral accumulators) by:

LOAD	1111	N	AMF and ADDRESS of A
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 to 31

where the ADDRESS is the memory location of the first component of A.

2. The second vector or matrix, B, is then added to (or subtracted from) the first by:

A D D	0 0 0 0	0 0 0 1	AMF and ADDRESS of B
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 to 31

where the PF = 0000 means there are no answer register and no second operand is repeated, and R = 0001 means the answers are stored in the Accumulator stack via an AAST and the entire process is to be performed only once.

The results of executing these two instructions is that the sum of the two vectors or matrices is stored in the accumulator stack, if $R \neq 0$. In the case where $R = 0$, the sum remains in the AP scratch pad and another vector or matrix could be added to the sum by repeating the second instruction with the appropriate new ADDRESS part.

The assembly language equivalent to these two instructions is:

1. LOAD (N A
2. ADD)$_o$ 1 B.

The vector dot product between A and B, which is mathematically defined as

$$C = \sum_{i=1}^{N} a_i b_i,$$

is specified by the two instructions:

1. The first vector is loaded in the AP scratch pad with the deferred operation of Add by:

A D D	1 1 1 1	N	AMF and ADDRESS of a_1
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 to 31

or as ADD (N A in assembly language.

2. The multiply by B is ordered with each operand used once and with one answer register ordered (PF = 0001), and the cycle is to be performed once and the answer register stored (R = 0001) by:

MULT	0001	0001	AMF and ADDRESS of b_1
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 to 31

or MULT $)_1$ 1 B

Since only one answer register is specified, after each operand is multiplied by its respective accumulator value, the product is functionally combined into the answer register using the deferred operation, in this case ADD. Thus the operations of multiply and add repeat on successive operands, and, at the end of the sequence, the answer register contains the dot product of the two vectors.

A variation to the above procedure is to use the original vector, A, in the second instruction and then after the two instructions are completed, take the square root of the resultant sum. This produces the magnitude of the vector A.

The matrix multiplication of two matrices A and B is mathematically defined by:

$$c_{ij} = \sum_{k=1}^{K} a_{ik} b_{kj} \quad \begin{array}{l} \text{for } i = 1 \text{ to } I \\ \text{and } j = 1 \text{ to } J, \end{array}$$

where A, B and C are I x K, K x J and I x J matrices, respectively. The multiplication of a 2 x 3 matrix by a 3 x 4 matrix is specified by the two instructions:

1. The first matrix A is loaded in the deferred accumulators along with the deferred ADD operator by:

A D D	1 1 1 1	0 1 1 0	AMF and ADDRESS of A
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 31

or ADD (6 A which causes the following results:

a) The AP enters block mode with N = 6,

b) The six operands starting in the location specified by ADDRESS are fetched from the Task Memory,

c) The six operands are stored in the AP scratch pad (the deferral accumulators) starting in the working accumulator.

d) The operation ADD is also stored in each scratchpad location as a deferred operation.

2. The multiply by B is ordered with two answer registers (PF = 2) and each operand used twice (PF = 2) and the entire process repeated four times (R = 4) by:

M U L T	0 0 1 0	0 1 0 0	AMF and ADDRESS of B
0 1 2 3 4 5 6 7	8 9 10 11	12 13 14 15	16 31

or MULT $)_2$ 4 B in assembly language.

The operation of the AP under these two instructions, assuming the matrices are stored column-wise (i.e. $a_{00}\ a_{10}\ a_{01}\ a_{11}$ ...), is as follows. First two

answer registers are set up and the products $a_{00}\ b_{00}$ and $a_{10}\ b_{00}$ are stored in them. (Note b_{00} is used in each product.) Next, the second two products $a_{01}\ b_{10}$ and $a_{11}\ b_{10}$ are formed using b_{10}, but since there are only two answer registers the products must be combined functionally, after each is formed, with the partial answers in the answer registers using the deferred operation ADD. Thus the answers in the two registers are now $a_{00}\ b_{00} + a_{01}\ b_{10}$ and $a_{10}\ b_{00} + a_{11}\ b_{10}$. Third, the operand b_{20} is brought in and repeated, and functionally combined, forming in the answer registers:

$$a_{00}\ b_{00} + a_{01}\ b_{10} + a_{02}\ b_{20} = c_{00}$$

$$a_{10}\ b_{00} + a_{11}\ b_{10} + a_{12}\ b_{20} = c_{10}$$

which is the first column of the answer matrix. This ends the block cycle (N = 6) since all six operands in the AP scratch pad set has been used; thus, the answers are stored and the answer registers are cleared.

Since the R field called for the repeat of this block cycle (of 6 steps) four times, the following answers are also created and stored:

$$a_{00}\ b_{01} + a_{01}\ b_{11} + a_{02}\ b_{21} = c_{01}$$

$$a_{10}\ b_{01} + a_{11}\ b_{11} + a_{12}\ b_{21} = c_{11}$$

and $$a_{00}\ b_{02} + a_{01}\ b_{12} + a_{02}\ b_{22} = c_{02}$$

$$a_{10}\ b_{02} + a_{11}\ b_{12} + a_{12}\ b_{22} = c_{12}$$

and $$a_{00}\ b_{03} + a_{01}\ b_{13} + a_{02}\ b_{23} = c_{03}$$

$$a_{10}\ b_{03} + a_{11}\ b_{13} + a_{12}\ b_{23} = c_{13}.$$

Thus generating the four columns of the matrix product. Therefore the matrix multiplication is specified completely by ONLY TWO machine language instructions.

The student may find this procedure a little novel, and it is. First, when matrix multiplication is done manually only one accumulator is used and all the terms for the first answer component (c_{00}) are combined before other components are formed. Here two (or PF) answers are constructed as a group. The advantage of this method for the computer is that each component of the second matrix is used (and thus retrieved) only once. Second, the number of columns in the first matrix, (and the number of rows in the second one) are transparent to the computer; thus, instead of using this number as a looping parameter, the total number of components in the first matrix is used to indicate the end of a block cycle. The third difference is that the standard manual convention is to determine the top row of the answer matrix first, whereas here the left-most column is determined first. In fact this procedure will NOT work if the matrices are stored row-wise (or in row order). However, the answers are the same in both cases (neglecting roundoff errors), and using the PE procedure requires only two simple machine language instructions instead of requiring a description of how every element is manipulated. Thus, this method is much easier to use and operates faster. As a further example, consider the multiplication of 3 x 4 matrix by a 4 x 5 matrix. Here N = 12, PF = 3 and R = 5. Notice that the two parameters for the second instruction are actually the size of the resulting matrix. The remainder of this example is left as an exercise for the reader. Another example is available in [5.1 pages 2-37 to 2-40].

In general the matrix procedure described here can handle any N x M matrix times a M x P matrix where $N(M + 1) \leq 16$*; since N answer registers are required in addition to the locations for the first N x M matrix.

The time to execute a matrix operation can be calculated by adding together the time to load the scratchpad (assuming the instructions are already in APQ), the time to execute N multiplied by R instruction pairs and the time to store the results in the accumulator stack. The time to load the second matrix can be neglected because it is overlapped by the execution. For the example above of a 2 x 3 times a 3 x 4 matrix, the time would be calculated as follows:

1. 6 x 150 nsec = 900 nsec to load scratch pad,
2. 6 x 4 x (800 + 100) = 2160 nsec to do 24 multiplications and additions (actually only 18 additions are used, but 6 clear accumulator are also used), and
3. 8 x 150 nsec = 1200 nsec to store the results in the Task Memory,

for a total of 4.26 microseconds, neglecting any overhead. (This is approximately the time to do one multiplication on the IBM 360/65 computer.)

5.2.10.3 More Complex Matrix Operations

Several instructions were invented to accomplish the remaining vector and matrix operations. The most essential is the Calculate Cofactor instruction, which permits the computation of Vector Cross Product, Determinants, and Inverse Matrices. All of the vectors and matrices handled in this area are three dimensional, this being the a) only reasonable size for the scratch pad to efficiently handle, and b) most likely size for computations.

*This number is now 256, according to Raytheon at the 1973 AADC Symposium.

To investigate this area, let us first examine a 3 x 3 matrix:

$$A_{00} \quad A_{10} \quad A_{20}$$

$$A_{01} \quad A_{11} \quad A_{21}$$

$$A_{02} \quad A_{12} \quad A_{22}$$

An interesting observation can be made that the subscripts are the numbers $0 \rightarrow 8$ in the ternary systems, with column taking precedence over row. Expressing each of the subscripts in binary notation by digit, this expression would be:

0000 0100 1000

0001 0101 1001

0010 0110 1010

which, expressed as whole numbers in decimal, would be

0 4 8

1 5 9

2 6 10

The cofactors of this matrix can be expressed as the cofactors of each individual term, thus:

Cofactor of $0 = 5 \cdot 10 - 6 \cdot 9$

Cofactor of $5 = 0 \cdot 10 - 2 \cdot 8$

Cofactor of $6 = 0 \cdot 9 - 1 \cdot 8$

Returning now to the binary representation, it is seen that each cofactor term has a direct relationship, in pairs of bits, to the four elements of its corresponding cofactor.

Thus, the cofactor of 0000 is 0101·1010 - 0110·1001 and, to obtain the elements of the cofactors of 0000, the following device may be applied.

Positive Elements → Add 1 to the left pair and 1 to the right pair
→ Add 2 to the left pair and 2 to the right pair

Negative Elements → Add 1 to the left pair, 2 to the right pair
→ Add 2 to the left pair, 1 to the right pair

More briefly,

Positive Elements: Add 1, 1
Add 2, 2

Negative Elements: Add 1, 2
Add 2, 1

Observe that this scheme holds true for any cofactor term, using mod 3 addition (1 + 2 = 0), thus, the cofactors of 0101 are found to be:

Positive Elements: Add 1, 1 → 1010
Add 2, 2 → 0000

Negative Elements: Add 1, 2 → 1000
Add 2, 1 → 0010

This says that the

Cofactor of 0101 = 1010·0000 - 1000·0010

or Cofactor of 5 = 10·0 - 8·2

which agrees with the previous determination.

Therefore, by implementing a mod 3 (ternary) loading scheme and finding the cofactors of any term by doing the ternary adds of 1, 1 & 2, 2 for positive elements and 1, 2 & 2, 1 for the negative elements, all the cofactors of a 3 x 3 determinant can be regularly ascertained and calculated by an iterative process.

This method can now be applied to the vector/block mechanism to obtain Vector Crossproduct Calculation. Begin by loading the element of vector A into registers 0,1,2:

A_0 contains A_0

A_1 contains A_1

A_2 contains A_2

Now load the elements of vector B into registers 4,5,6 (maintaining the ternary loading pattern).

A_4 contains b_0

A_5 contains b_1

A_6 contains b_2

Then the three terms of the cross-product A x B, which are:

$$A_1b_2-A_2b_1$$

$$A_0b_2-A_2b_0$$

$$A_0b_1-A_1b_0$$

are, in fact, the cofactors of 8, 9, and 10 as previously defined. The mechanism operates in a regular fashion to produce any or all of these terms.

Thus, assuming a ternary loading scheme, a calculate cofactor instruction would have to specify a) the number of cofactors to be calculated and b) the first cofactor to be calculated. (Cofactors are specified by 0,1, 2,3,4,5,6,8,9,10,0,1,2....) Thus, the cofactor instruction would have the following format:

C O F								1st			NUMBER OF COFACTORS																						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		

This instruction is not addressable (i.e. contains no address field). To facilitate the processing of this type of instruction, the loading of vectors and matrices are specified as being in ternary mode.

Given the ability to calculate cofactors, it is possible to define another instruction of the same type:

Calculate Determinant

Nine operands are loaded into the scratchpad in ternary mode as a precondition. Cofactors 0,1 and 2 are calculated and multiplied by their respective terms (i.e., term 0 x cofactor 0, etc.). The three final products are summed and the final answer placed in deferral register 15. The final answer is also stored in the memory stack.

Returning to the Calculate Cofactor instruction, one further field can now be defined. A 0010 in the R field causes each cofactor calculated to be divided by the contents of deferral register 15.

Combining these instructions with other block mode instructions yields a very powerful hardwired vector and matrix processing set:

1. Vector Cross Product:

    a) Two vectors must be loaded:

        LD (3 A

        LD (3 B

    Ternary loading is automatically sequenced.

    b) The appropriate cofactors must be calculated. These, as previously described, are the cofactors of 8,9 and 10 (or, three cofactors starting with 8).

        COF)$_3$ - 8

2. Calculate Determinant (3 x 3)

    a) The determinant is loaded:

        LD (9 A

    b) The determinant is calculated. The value is stored in the stack and in deferral register 15,

        DET)$_1$

3. Calculate Cofactor [calculate N cofactors starting with P]

    a) A determinant (3 x 3) is loaded:

        LD (9 A

b) The cofactors are calculated

COF)$_N$ - P

4. Calculate Inverse Matrix - (3 x 3)

a) The matrix is loaded (automatically in the ternary mode)

LD (9 A

b) The Determinant is calculated. The value is not stored in the A stack, but is placed in deferral register 15.

DET 0

c) All cofactors are calculated and divided by the value of the determinant, thus yielding the terms of the inverse matrix in correct column order

COF)$_9$ 2 0

5. Solve Simultaneous Equations

a) Do an invert matrix sequence as above.

b) Reread the inverse matrix into the scratchpad with a held add (Ap = A stack pointer):

AD (9 Ap

c) Multiply with three answer registers repeating each operand of B three times (B = constant vector):

MUL)$_3$ 1 B

The foregoing instructions give the PE the ability to perform all basic vector and matrix operations except the creation of a transpose matrix. To make the set complete, and give the PE as much versatility as possible in the vector/matrix field, a Transpose Matrix instruction is included in the instruction set.

This instruction takes a matrix stored in memory in column precedence form and loads it into the scratchpad in row precedence form. The transpose instruction does not require a full matrix to operate. If a partial matrix is given, the terms of the matrix which are specified will be loaded into the correct row positions in the scratchpad. For example, if one column (4 terms) of a 4 x 4 matrix are given, they will appear in the scratchpad in registers 0,4,8, and 12.

5.2.10.3 An Application of Parenthetical Control

When a load transfer instruction is used, it causes a set of spaced accumulators to be loaded, i.e., LDTN (4 A loads A_0, A_1, A_2, A_3 into AC_0, AC_4, AC_8, AC_{12}. Parenthetical Block Operation will now be effective.

AD (B

will Load B_0 to B_3 (with deferred AD) in AC_1, AC_5, AC_9, AC_{13}

ML) C

will multiply C_0-C_3 by B_0-B_3 and release add into A_0 - A_3, thus producing A + BC values in AC_0, AC_4, AC_8 and AC_{12}.

Thus, up to four sets of simultaneous subroutines using parenthetical control may be implemented.

5.3 ARITHMETIC PROCESSOR

5.3.1 Data Types, Mode Control and Number Systems

Although Raytheon suggests there are many advantages to either tagging each data word with a data type or to eliminating data types altogether and using only floating point operations, they do not recommend these approaches for AADC because they think that most potential AADC users will find it difficult to accept 24-bit limitation on integers when a 32-bit word size is available. (Tagged data words also mean longer words in memory for a given accuracy.) The advantages to using a single data type is that the compiler is much simplier, no conversion from fixed to floating or from floating to fixed is required, floating point hardware can be made as fast or faster than fixed point and there is no need to worry about integer overflow. If an integer overflow occurs internally when using floating point hardware, nothing at all happens. If it occurs in the output, the user gets his answer with the appropriate scale factor, but minus some significance, instead of the normal "Terminated due to Integer Overflow" message. Also accuracy is enhanced because fixed point numbers are often entered without factional parts or scale factors to ensure that overflow conditions do not occur.

As an alternate to eliminating data types, Raytheon suggests a mode control to determine the types of operation. This has the disadvantage of not allowing mixed mode fixed and floating arithmetic but does have the advantage of reducing the number of necessary op codes. Without any form of data insensitivity control, all arithmetic, conditional, polynomial, vector and matrix instructions and all subroutines must have counterparts for every data type. The other disadvantages of mode control method is that modes must be established, fixed overflow can still occur and all instructions are in fact implemented in hardware.

The mode switch would probably be implemented with the load or branch instructions or with subroutine calls.

Raytheon has recommended the sign and magnitude number system for AADC claiming that multiplication and division algorithm are much simplier for this system, especially when compared to 2's complement system, and that it is the only system that can cause an overflow in fixed point multiplication.

The author does not agree with Raytheon on many of the statements concerning number systems. For example, they state that one of the reasons for choosing the sign and magnitude system is that multiplication and division can be implemented by considering only positive numbers, but they ignore the fact that there are 2's complement algorithms that treat positive and negative numbers alike, i.e. Booth multiplication algorithm.

5.3.2 Data Formats

Four data formats are presently designed for the PE. The floating point format is:

S	MANTISSA	S	EXPONENT
0	1 to 23	24	25 26 27 28 29 30 31

where the decimal point is assumed to be to the left of bit 1. The fixed point or integer format is:

S	MAGNITUDE
0	1 to 31

where the decimal point is assumed to be to the left of bit 1 for fixed point but to be to the right of bit 31 for integer format. The complex data format is:

S	REAL MAGNITUDE	S	IMAGINARY MAGNITUDE
0	1 to 15	16	17 to 31

where the real and imaginary parts are considered as 16-bit fixed point numbers, with decimal points assumed to be to the left of bits 1 and 17. Although floating complex arithmetic (single word format) is not considered as part of the study, it could be included using the format*:

S	REAL MANTISSA	S	IMAGINARY MANTISSA	S	EXPONENT
0	1 to 11	12	13 to 23	24	25 to 31

where the real and imaginary parts are each represented by only 12 bits (equivalent to 3 decimal digits of accuracy) and both parts have the same 7 bits exponent.

All floating point exponents are considered binary numbers, not hexidecimal, thus the mantissa may be shifted only one bit.

Raytheon brags about the large 7-bit exponent being capable of representing the range of all conceivable numbers for avionics applications. The 7-bits represent a range of $10^{\pm 38}$ which is the same as on the IBM 360/67 (and often restrictive in scientific applications).

5.3.3 Arithmetic algorithms

The floating point algorithms for addition, subtraction, multiplication and division are fairly standard for sign and magnitude number systems, except that they do not normalize until it is necessary. This results in more complex circuitry but the faster speed, apparently justifying the extra logic cost. The floating point algorithms actually use the fixed point arithmetic hardware to do the arithmetic.

The particular implementation of the adder and/or subtractor network is left to the final designer providing it produces a sign and magnitude answer in about 100 nsec. The multiplier is a 4-bit-at-a-time multiplier that actually

*This format has been replaced by a double word complex number format.

uses two 2-bit-at-a-time adders working simultaneously. Sufficient circuitry is added to perform the additions and shiftings as required. A very fast logical carry function is used so the second adder has this input at almost the same time as the first adder. Further details are available in [5.1, pages 3-19 to 3-21].

Multiplication now takes only 8 steps for a 32-bit word. If each step takes 100 nsec then the fixed multiplication takes 800 nsec. Since the floating point mantissa is only 24 bits long it takes only 6 steps or 600 nsec. With the 7 to 3 mix of additions to multiplications, the AP runs at 3.3 MIPS for fixed point and 4.0 MIPS for floating point operations.

Since the division instruction is a low frequency instruction, a very simple one-bit-at-a-time subtract and check algorithm is implemented for division. The hardwired divide algorithm would take about 3.2 msec.

Integer arithmetic uses the identical algorithms for fixed point. Addition and subtraction are identical. Multiplication is the same except that the answer appears in the low order product register. Division is the same except that the single dividend word is placed in the low order product register before division.

Since an analysis of the usages of complex arithmetic in avionics missions revealed that 16 bits is sufficient for either the real or imaginary parts, fixed point complex arithmetic was implemented in half-word format with essentially no loss in efficiency. Even complex multiply (an operation requiring four multiplications and two additions) is accomplished in almost the same time as a normal full word fixed point multiplication. Two conventional full length words can be used with complex instructions. For example, a two-word load,

followed by a complex store, is equivalent to combining the left half of two words into a single word. The multiply operation can produce either two full-word answers or two half-word answers depending on whether the result is used as separate real and imaginary values (or double precision) or as a complex answer. Thus complex arithmetic has been conveniently and efficiently included in the PE structure.

As a result of implementing the complex mode operations, it is very easy to include half-word operations on either the left or right half of the word. Half-word and full-word operations can be mixed and can be specified in all modes including fixed, floating and complex (and probably integer, although this is not mentioned).

Twenty-four load instructions are implemented in the PMU of the PE. These include combinations of 1) load normal or load with negative sign, 2) fixed point, floating point, complex or integer, and 3) full word, left half word or right half word. A complete listing is shown in section 5.6.2.6 and [5.1, pages 3-19].

This section has been an overview of the arithmetic operations implemented in the PE, and is intended to present the PE features available to the programmer, rather than the details of the algorithms. The arithmetic algorithms are described in much more detail in [5.1, pages 3-9 to 3-29].

5.4 ARITHMETIC PROCESSOR DESIGN

Figure 5.2 shows the basic Arithmetic Unit (AU) - the heart of the Arithmetic Processor (AP). (The other parts of the AP include a comparison test valid unit, a deferral unit and a programmable control unit.) The AU contains two memory interface registers (M_1 and M_2), two accumulators (A_1 and A_2), two low-order product registers (L_1 and L_2) and an Arithmetic Logic Unit (ALU). The two memory registers provide the interface between the Task Memory and the rest of the AP. The low-order product registers are built as logical extensions to the accumulators for multiplication and division, but also can be used as accumulators. The Arithmetic Logic Unit performs the actual execution of the addition, subtraction and logical operations. ("ALU" is my own term because Raytheon uses "AU" to refer to both the ALU and the Arithmetic Unit - which contains 6 registers, the ALU and associated control and transfer logic.)

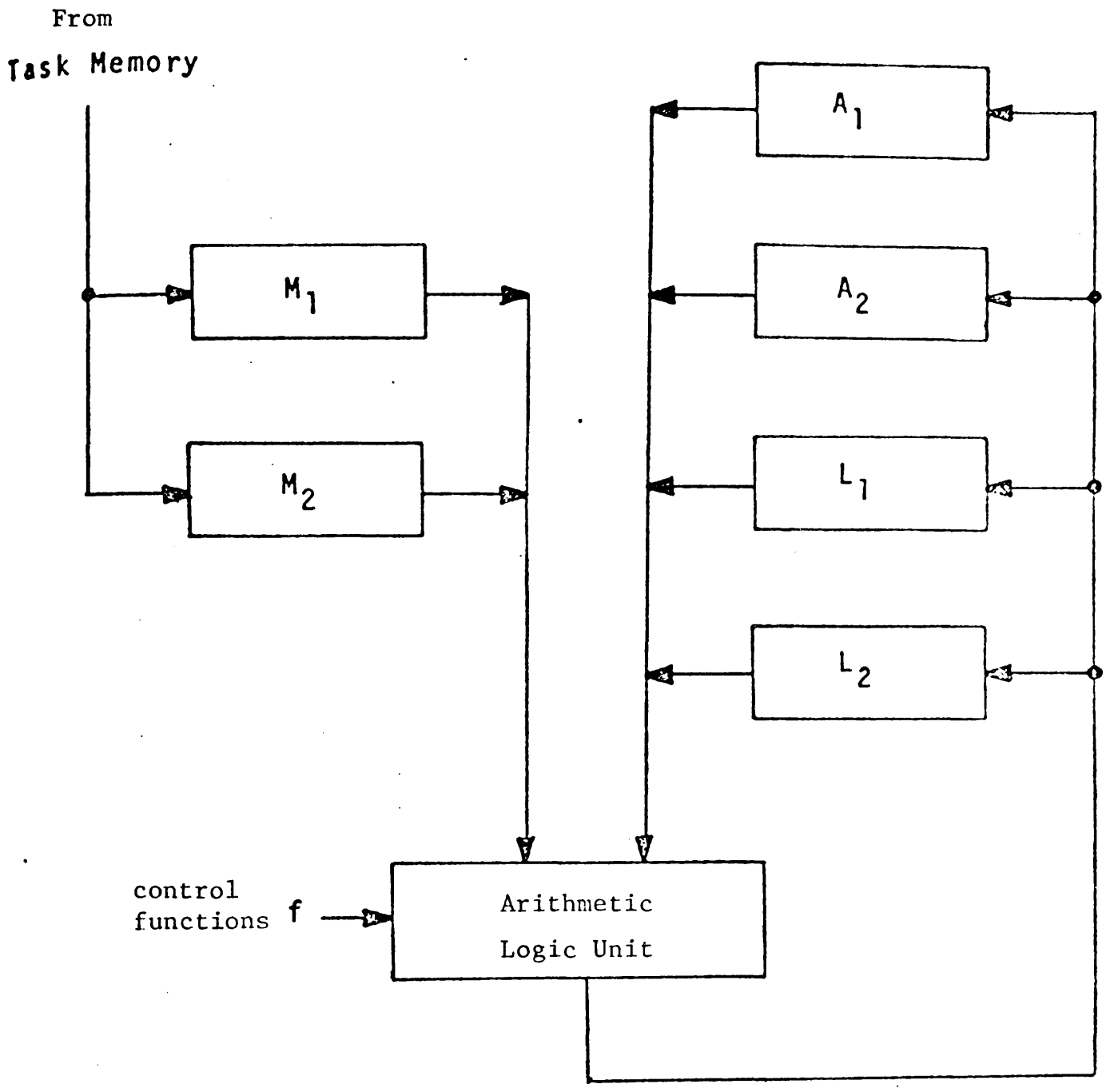


Figure 5.2 Simplified Diagram of Arithmetic Unit

In the final design, the use of control signals to determine which A or L register is to be used as the current accumulator is generalized to include the M registers. Thus the M registers are equivalent to A and L registers, and any of the six registers can be used to accept operands from memory, and can be used as the input or output to the ALU, as shown in Figure 5.3. In other words, the source and destination of any operation can be changed by a control signal. In Figure 5.3 the inputs and outputs of the ALU are as follows: f is the control function that determines the particular operation to be performed; M is the operand from memory (i.e., addressed in the current instruction), A_S is the operand from the Accumulator and A_D is the output from the ALU.

By changing control signals, the output from the M registers can be decremented, incremented, complemented, shifted left one or two bits (2 x M or 4 x M) or ignored before being applied to the ALU. The output from the A and L registers can be complemented, shifted right one bit or 4 bits and shifted left one or two bits before being applied to the ALU. The input to the A and L registers can be from the scratch pad registers as well as the following modified outputs from the ALU: The sum directly, sum positioned two bits right and the sum positioned one bit left.

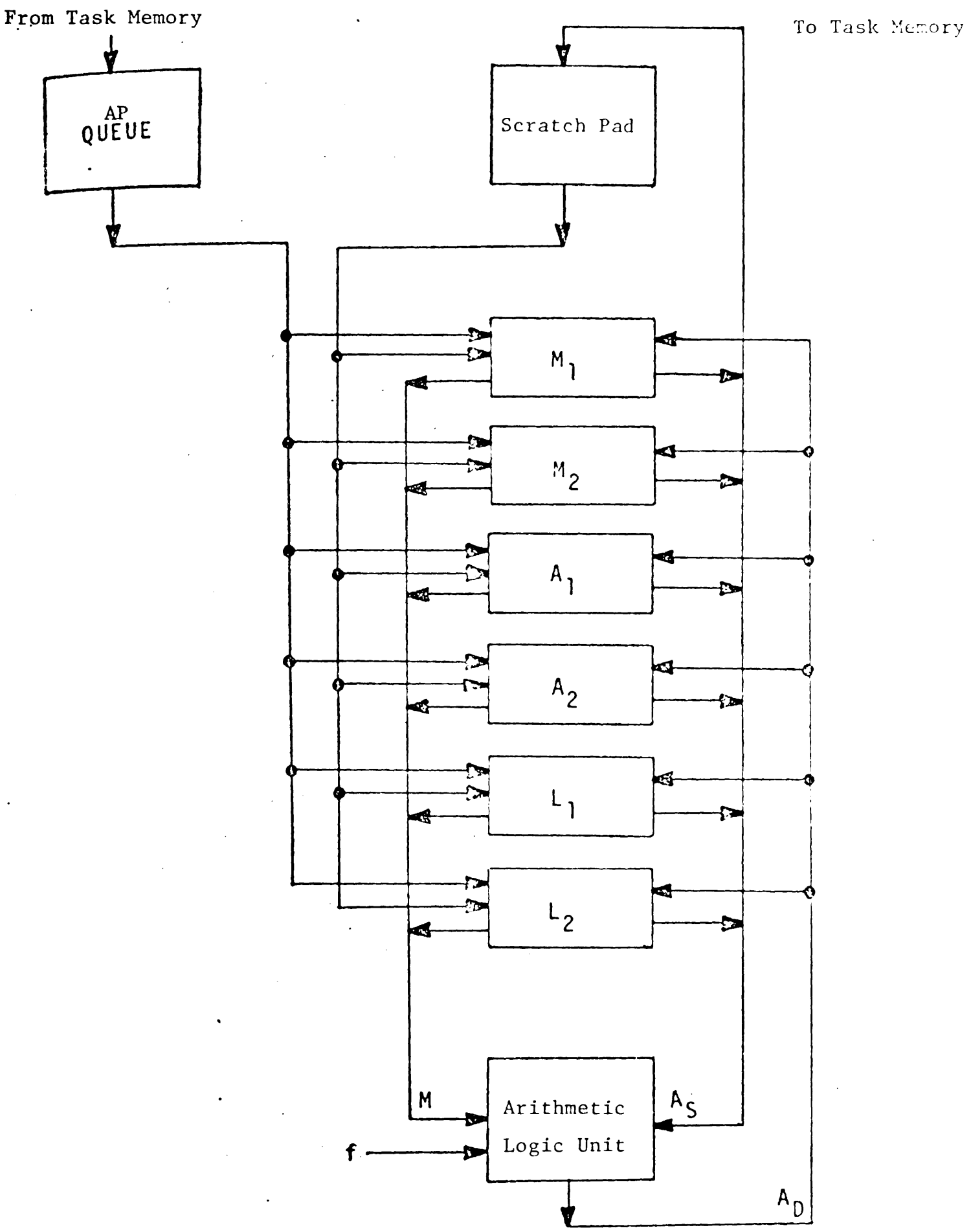


Figure 5.3. Arithmetic Unit Final Design

In actual fact there are two arithmetic logic units called the PAU and SAU for Primary Arithmetic Unit - the one described above - and Secondary Arithmetic Unit. (Again, Raytheon does not distinguish between the AU and the ALU, which makes it a little confusing here. PAU and SAU should be PALU and SALU respectively.) The SAU is designed for use during the 4-bit-at-a-time multiply and thus has only the addition and subtraction logic. Otherwise, it is essentially the same as the PAU described above. Some examples of the AU functioning is described in [5.1, pages 4-17 to 4-21].

5.4.1 Macro-Micro Programming

Raytheon intends to implement some micro-programming features into the PE by making the control signals, that were used to implement the steps of the macro instructions, available within the macro instruction set. Thus the macro instruction set will include the vast majority of useful micro-commands needed for any use by the arithmetic processor. If the macro instruction set contains all the elementary micro-functions as a subset, the total number of control lines in the AU could be reduced to eight (the number of bits in the OP CODE). Thus, Raytheon claims that micro instructions can be included in the macro instruction set by providing the AU with an eight line input which would be decoded internally to provide the important micro instructions.

Another explanation of the micro instructiom implementation is given in [5.1, pages 4-14 to 4-16], but it is not any more detailed than the above. How many micro instructions can be accomodated and which ones should be implemented has not yet been decided. Also see questions 5.16 and 5.17.

5.4.2 Deferral Unit

The deferral unit is a portion of the AP which contains the necessary equipment to handle the Parenthesis Control and Vector/Block Mode. The deferral unit contains a pushdown set of 16 scratchpad 40-bit registers which are connected to the arithmetic unit via the accumulators (See Figure 5.3).

Implementation of the deferral unit will not be described here because it is quite elaborate and because it is believed that the explanation of the operation in Sections 5.2, 5.9 and 5.10 is adequate for our purposes. For further details the reader is referred to Reference [5.1], particularly Figures 4.8 and 4.9 which are a detailed functional block diagram of the overall deferral unit and pages 4-25 to 4-35 which explain the operation of the deferral unit. The other two parts of the AP - the test valid unit and the programmable control unit - are also not described here [5.1, pages 4-23, 4-24, 4-35, 4-36].

5.5 PROGRAM MANAGEMENT UNIT

The Program Management Unit (PMU) handles all the functions of the PE which do not require the computation and processing capabilities of the AP. The functions - which essentially all deal with the processing, utilization and stor of task memory addresses - fall into three major categories: normal instruction and operand fetching, execution of program management-type instructions, and I/O control. Of the three, the normal instruction and operand fetching is the most important in terms of speed and efficiency enhancement.

The PMU contains the follow basic components:

1. A program counter (P) which contains the address of the memory location containing the next sequential instruction.
2. A Primary Address Register (I_A) which typically contains the effective address of the memory location containing the operand of the current instruction. The effective address is the ADDRESS part modified by the AMF (address modifier field) part.
3. A Secondary Address Register (I_B) which contain the secondary address if two address instructions are implemented.
4. A Control Register (C) which holds parameters of the instruction being processed (or accessed) in the PMU. It presents this information to the control unit for decoding and to generate the necessary PMU control.

5. A PMU Scratch pad (PMUSP) which contains right index registers used by the system, and also contains four stack pointers.
6. A Fetch Arithmetic Unit (FAU) which is capable of handling the simple arithmetic operations which indexing and PMU instructions require. The FAU is not nearly as elaborate as the arithmetic unit in the AP.
7. An Arithmetic Processor Queue (APQ) which queues operation codes and operands for presentation to the AP. The APQ is actually an interface unit but since it is most responsive to the PMU and its functions (half-word manipulations) approximate PMU functions, it is included in this section.

Figure 5.4 shows the component of the PMU. For further detail the reader is referred to [5.1, Figure 5.2] which shows a more detailed block diagrams of the PMU and its control signals. The actual operation of the PMU is considered beyond the scope of this report. The interrupt handling mechanism and the APQ is also not included in this report [5.1, pages 5-8 to 5-12].

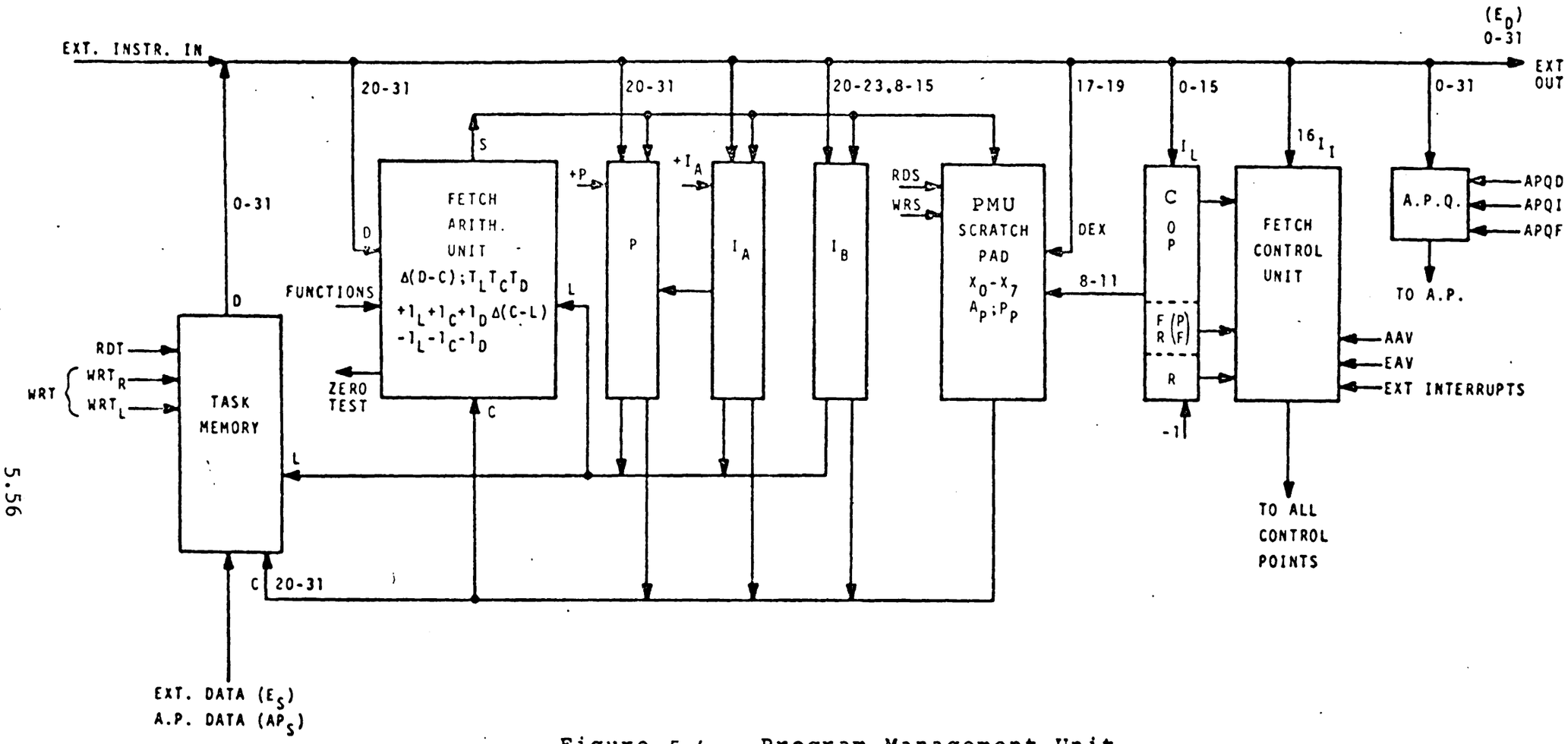


Figure 5.4. Program Management Unit

5.6 THE INSTRUCTION SET

This section lists the instruction set of the PE - an instruction set wide in scope but simple in format according to Raytheon. The instruction set is based on a report by Systems Consultants [5.5] and, while it is a considerable modification to the set in that document, it maintains the spirit of the recommendations throughout. This section will list all the instructions which should, according to Raytheon, be included in the AADC DPE. Subsections 5.6.2 and 5.6.3 describe the instructions that deal with the Arithmetic Processor, including those dealing with vector and matrix operations [5.3]. Subsections 5.6.3 and 5.6.4 describe PMU instructions which may subject to further modification since the material is based on an earlier reference [5.1].

5.6.1 Basic Instruction Format

The basic instruction format called Format 1*, was given in Section 5.2.2.1 but is repeated here for convenience:

OP CODE								PF				I	X			ADDRESS															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16				to											31

Sometimes other names are given for some fields.

In computer operation, an instruction word is obtained from the Task Memory by the PMU. The left 12 bits of the word is used by the APQ; the right half is used to obtain the operand required for the instruction execution. PF is the parenthesis field. Bit 12 is set equal to 1 for indirect addressing, while bits 13 to 15 are used to specify any of eight PMU index registers. Bits 12 to 15 are also referred to as the Address Modifier Field (AMF). ADDRESS is the virtual Task Memory location of the operand.

---

*Shown here with the new 16-bit ADDRESS field.

5.6.2 Arithmetic Processor Instructions

5.6.2.1 General Considerations

Each instruction in the AADC instruction set that involves obtaining an operand from Memory for use in the Arithmetic Processor, is equivalent to several instructions normally delineated separately in other computers. Firstl each instruction is effectively modified by its associated Data Type. The AADC Date Types are:

1. Integer
2. Logical
3. Literal
4. Real
5. Imaginary
6. Double Precision Real
7. Complex

Secondly, all arithmetic operations are performed as double precision operations, with the appropriate bits selected depending on the data type of the result. Thus a pair of integers, a pair of real number, a real and an integer or a pair of double precision real numbers are all added by a double precision addition operator but the result is stored as a 32 bits integer, a 24-bit mantissa and a 8-bit exponent real number, or a double precision real number depending on the data type of the result. Thus there is no speed disadvantage associated with double precision but there is still a doubling of the memory required.

Thirdly, each operand may be a simple scalar, or a member of a vector or other form of array. Full capabilities of Mixed Data Types as well as mixed scalar, vector, matrix operations further make for instruction variations which

would be exceptionally difficult to individually enumerate. In addition to these instructions, a set of Program Management instructions complete the Instruction Compliment. If all of this were not enough, the capability of Parenthetical Control with the facility of controlling the precision of instruction results permits maximum control of the operational sequence.

The operation of the Arithmetic Processor when dealing with arrays needs further elaboration. Scalars are, in general, extended to equal the size of the array. Thus, if the scalar is contained in the Accumulator, an instruction which would normally be accumulator destructive will not affect the original scalar until completion of the entire array. If the vector is in the accumulator, the scalar will be repeated from the Queue.

A similar extension will be made for vectors of unequal size. If the shorter vector is contained in the Queue, the last term will be repeated until completion. If the shorter vector is in the accumulator, it is iterated until completion.

Providing Parenthetical Control for use with Matrices permits assembly of sparse matrices, and other operations which would otherwise be difficult to achieve. The instruction set which follows is virtually complete with respect to the Arithmetic Processor. Additional PMU instructions are still required to complete the set.

5.6.2.2 Standard Arithmetic Instructions

The standard arithmetic instructions are add, subtract, multiply and divide. The arithmetic instructions apply to all data types, as well as, scalar, vectors and matrices.

There are three instructions corresponding to addition. They are Add accumulator and memory (A + M → A), clear and add (+M → A) and an no OP (+ A →A).

There are four instructions corresponding to subtraction. They are Subtract memory from accumulator (A - M → A), Subtract Reverse (-A + M → A), Clear and Subtract (-M → A), and Change Sign (-A → A). Two related monadic instructions are Absolute Magnetude (|A| → A) and Set Sign Negative (-|A| → A).

Five multiply/divide operations have been defined. They are Multiply accumulator by memory (A x M → A), Divide accumulation by memory (A ÷ M → A), Divide Reverse (M ÷ A → A), Residue (A ÷ M with R → A) and Residue Reverse (M ÷ A with R → A). If a literal data type is used with a Divide Reverse instruction, the literal 1 can be used to generate the reciprocal. The complex divide is semi-complex, i.e., $(A_L + iA_R)/M$ where A_L and A_R are the left half and right half of the accumulator, respectively.

In the earlier report [5.1] it was stated that arithmetic operations can be performed on the right half or the left half of a word as well as on the full word, but this was omitted (intentionally?) from the later report [5.3].

5.6.2.3 Logical Instructions

This section describes the logical operations in detail. All sixteen possible combinations of 2-value Boolean variables are implemented eliminating many unnecessary inverting operations and thereby improving the operating speed.

The logical functions are described in Table 5.5, where A and M represent the accumulator and memory contents, respectively. Each bit of M(each M_i) and each bit of A(each A_i) can assume a value of zero or one. All instructions use Format 1 and are subject to parenthesis control. All operations apply on a bit-by-bit basis on 32-bit words. Each operand can be any of the seven data types and furthermore a logical operation does not make the accumulator logical.

FUNCTION	OPERATOR	NAME OF SYMBOL	EQUIVALENT OPERATION
$0 \rightarrow A_i$	0	CLEAR	clear accumulator
$A_i \wedge M_i \rightarrow A_i$	$\wedge$	AND	
$A_i > M_i \rightarrow A_i$	>	Greater than	accumulator AND NOT memory
$A_i \rightarrow A_i$	+	Unary plus	NO operation
$A_i < M_i \rightarrow A_i$	<	Less than	NOT accumulator AND memory
$M_i \rightarrow A_i$		Monadic load	Logical load
$A_i \neq M_i \rightarrow A_i$	$\neq$	Not equal	Exclusive OR
$A_i \vee M_i \rightarrow A_i$	$\vee$	OR	Inclusive OR
$A_i \overline{\vee} M_i \rightarrow A_i$	$\overline{\vee}$	NOR	NOT (A OR M)
$A_i = M_i \rightarrow A_i$	=	Equals	Equivalence
$\sim M_i \rightarrow A_i$	$\sim$	Load complement	NOT memory
$A_i \geq M_i \rightarrow A_i$	$\geq$	Greater or equal	Accumulator OR NOT memory
$\sim A_i \rightarrow A_i$	$\sim$	Unary NOT	Complement accumulator
$A_i \leq M_i \rightarrow A_i$	$\leq$	Less or equal	NOT accumulator OR memory
$A_i \overline{\wedge} M_i \rightarrow A_i$	$\overline{\wedge}$	NAND	NOT (A AND M)
$1^i \rightarrow A_i$	1	SET	

Table 5.5 Logical Functions and Operators

Since logicals use operators which have other meanings when applied to arithmetic or non-binary (Boolean) operations, the High Order Language must distinguish between these functions. The operators could be followed by a second symbol to specify that these are logicals or the variables could be marked. It would seem that an operator subscript or second symbol would be best. The use of combinational symbols is necessary to avoid a phenomenal number of operational symbols.

One unusual feature of using logical operations on non-logical data types is that the exponent and its sign are not affected by the logical operations. Thus a logical load instruction can be used to load an arithmetic magnitude into the accumulator without affecting the Sign and Exponent of the original accumulator. The accumulator data type will not be altered. Also the unary NOT, which produces a one's complement of the accumulator magnitude and does not alter the Sign or Exponent, is not the same as a unary minus instruction which changes only the sign of the accumulator. This "feature" is of dubious value since its use could easily lead to errors in program results.

For example if the programmer specified complement accumulator and the data type was real when he expected it to be integer, then only the magnitude will be complemented and errors would probably result.

5.6.2.4 The Comparison Instructions

There are six comparison instructions which define all possible permutations of two variables. For consistency, these instructions are assigned names coinciding with left to right languages. The comparison instructions are greater, less, not equal, equal, greater or equal and less or equal. Comparisons are arithmetic and are made with any of the data types. Thus, there are actually many comparisons, (i.e., integer, floating, etc.)*

The result of a comparison is a Boolean value (True = 1 and False = 0). To permit logical operations on these comparisons, the results are placed in the Accumulator as the Arithmetic numbers +1 and +0, respectively. The creation of an arithmetic 0 or 1 permits the results of compares to be used arithmetically as well as the normal Boolean operation.

In general languages, comparison operation is Accumulator destructive. Since Array operations are provided wherein scalars can be effectively extended to the length of a vector, Compares are capable of being used in a non-destructive form.

Since comparisons against zero are often made with the desire to branch, and since zeros do not require an address field, an additional set of instructions have been provided which are essentially test and branch instructions. These instructions are essentially non-destructive of the Accumulator. They result in an immediate transfer. Transfers provide the address branch code in the instruction address field. They may be virtual or direct. If the comparison is true, the contents of the Queue are cleared and the branch

*Note if comparisons between different data types are allowed, then the programmer should have the option of flagging all such comparisons as possible errors.

code is sent to the Program Management Unit to perform a Transfer Unconditional (TRU) as follows.

FUNCTION	SYMBOL	NAME OF OPERATION
A > 0 → TRU	>	Transfer Positive
A < 0 → TRU	<	Transfer Negative
A ≠ 0 → TRU	≠	Transfer Not Zero
A = 0 → TRU	=	Transfer Zero
A ≥ 0 → TRU	≥	Transfer Not Negative
A ≤ 0 TRU	≤	Transfer Not Positive

Since the results of normal comparison instructions produce Boolean or Arithmetic answers of zero or plus one, these instructions are capable of following a comparison or logical and can thus be used as conditional transfers following the comparisons.

Since the conditional transfer set is complete, it would have been possible to provide a single comparison instruction which is an effective three-way compare yielding; +1, 0, Or -1, depending upon whether A is greater than, equal to or less than M. This instruction could then be followed by one or more of the branches. Since there are valid reasons for both solutions, the AADC instruction set contains both.

The instruction Transfer Unconditional (TRU) exists as a Program Management Unit instruction only. The instruction is effectively forced as the result of any of the conditional transfers above. It is necessary to complete the list of instructions in the Area of Branches.

Two other comparison instructions are provided in the Arithmetic Processor system. These are as follows:

FUNCTION	SYMBOL	NAME
A > M then M → A	⌊	MINIMUM
M ≤ M then A → A		
A < M then M → A	⌈	MAXIMUM
A ≥ M then A → A		

These are used to select the smaller or larger value of a pair. Other variations of comparison instructions are provided when dealing with a vector or list. These essentially involve the displacement addresses whenever the comparisons are true. These will be considered elsewhere.

5.6.2.5 Shift Instructions

Shift instructions apply to both Scalars and Arrays. The shifting of a scalar implies moving bits within the accumulator while shifting of an array implies moving elements of the array. The direction of shifting is controlled by the sign of the operand - right shifting if the operand is positive, left shifting if negative. There are three basic shifting instructions - Rotation, Drop and Take. Rotation is cyclic shifting either of bits in the case of scalars or of elements in the case of array. Array Rotation means the N^{th} (the instruction operand) element becomes the first element and the array is completed around. The Drop instruction causes the first (or last) N bits (or array elements) to be dropped. Thus Drop the left most N bits shifts the accumulator to the left filling the right most bits with zeros. Similarily Drop the right bits cause a right non-cyclic shift. The Take Instruction causes the first (or last) N bits (or array elements) to be taken from a scalar (or array). Take can be used as a mask since all bits remain in the same position. Further explanation of Drop and Take operation can be obtained from any APL reference manual such as [5.6].

Other simple arithmetic instructions include square root, the Floor of A (next lower integer of a real accumulator), and Ceiling of A (next higher integer of a real accumulator).

5.6.2.6 Polynomial, Vector and Matrix Instructions

According to an earlier reference [5.1], a hardware implementation of one polynomial instruction, PLY as defined in Subsection 5.2.9, can be used to generate all the trigonometric and logarithm instructions listed in that

subsection. In the later reference [5.2 and 5.3], the trigonometric functions to be implemented in hardware are listed as sin, cos, tan, sinh, cosh, tanh, arcsin, arccos, arctan, arcsinh, arccosh and arctanh. The later reference list the natural logarithm and the natural antilogarithm as hardware functions also, but suggests that the logarithm to an arbitrary base and exponentation to an arbitrary power should be implemented by subroutines.

The basic vector and matrix operations, such as adding and subtracting, are performed by the standard arithmetic instructions presented above, because the operands can be, in all cases, scalars or arrays. This also applies to loading and storing operations that will be discussed in Subsection 5.6.3.

When a comparison scalar is used against a vector, the address of the vector term where the comparison is made can be entered into the accumulator. This represents the first occurance rather than a simple Boolean True. The Scalar can be replaced by an Array. If no bit is found, the usual zero can be recorded.

Compression is the result of creating a vector from A for each 1 of a Boolean vector M and discarding an A for each corresponding zero of M.

Expansion is the result of creating a vector from A for each 1 of M and adding a zero term for each zero of M.

More powerful vector and matrix instructions are presented in the next subsection.

5.6.2.7 Composite Array Functions

One composite function is reduction. The reduction operation symbol "/" specifies that each term of a vector (or if a matrix, then each term of a column) is sequentially combined functionally in accordance with an operator contained in the OP Code field. For Matrices, the operation is repeated for

each column, producing a vector of answers. Reduction is monadic. To operate across rows, the matrix should be loaded Transposed. Each possible operator in the arithmetic instruction set can be used with reduction. Thus a + operator will produce the scalar sum of the vector terms (+/).

A second composite function is the generalized inner product. The inner product operation code specifies that the address field of the instruction contains two operation codes. The first operation code is applied term by term for each member of a column of M against each member of a row of A. This operation theoretically produces a matrix of answers, but these answers are reduced (see above) by the second operation to produce a vector of reduced answers.

Thus A x.+M is the ordinary matrix product of M and A. Again each of the dual operators can be any of the possible operation codes.

A third composite function is the Generalized outer product, wherein each term of M is functionally combined with every term of A producing, for example, a matrix from two vectors. The Outer-Product Operation Code requires a single operator in the address field, and corresponding new dimension words are created with the result.

A fourth composite function is the reduced outer product. Each term of M is applied to A through the first operator producing a vector of length A and each successive vector produced by subsequent terms of M is reduced with the first vector in accordance with a second operation code, also provided in the address field.

M v / = A ≡ v / M o . = A This will produce the function of Membership*, i.e., which elements of M are present somewhere is A.

Other powerful matrix operations that were described in [5.1] such as the determinant, cofactor and divide cofactor, have been left out of the later report [5.2 and 5.3]. Apparently these instructions will be implemented by

*These equations are taken directly from [5.3], but according to the right-to-left rule it appears that they should read M = / v A ≡ v / M =, o A.

subroutines rather than DPE hardware. Other array manipulating instructions, including loads and stores, are described in Subsection 5.6.3.

5.6.2.8 Programmables

According to [5.1] programmable Arithmetic Processor instructions be defined as required. The statement that any instruction deemed useful could be specified at any time and placed in the instruction set seems to be too general because there seem to be some quite serious restrictions on the variety of possible micro instructions. The limitation is that the total number of instructions presented above, all the PMU instructions and the micro instructions must be less than 256 - the number allowed by the 8 bit OP Code. Even Raytheon's example of coordinate conversion as a possible function that could be "micro programmed" using the available control signals is a relatively simple function and is not representative of the range of functions for which microprogramming would be useful.

In the later version of Raytheon's report they have omitted reference to programmable arithmetic Processor instructions so this feature may have been withdrawn [5.2].

5.6.2.9 Omitted Instructions

The latest references [5.2 and 5.3] have omitted several instructions that were under consideration previously [5.1] and that should be reconsidered. They are memory plus one to accumulator, memory minus one to accumulator, add magnitude and subtract magnitude. The first two are particularily important when one considers the number of counters that are normally incremented or decremented in an average program.

Also two very useful bit manipulating instructions Set Bit N and Reset Bit N has been eliminated apparently. Also the instruction Reverse Bits has been deleted but it would appear to have limited application.

5.6.3 Special Handling Instructions (AP and PMU)

The special handling instructions require the Program Management Unit (PMU) as well as the Arithmetic Processor for processing. These instructions include loading, storing and array manipulating instructions.

As previously described, the simple clear and add instruction may be a scalar which requires very little PMU involvement other than the data Fetch. Or, the data type may be double precision or Complex, in which event the PMU will obtain two consecutive operands and place them in the Queue marking the data type appropriately. If the data is an Array, the PMU will assign a Task Memory Address to start the Array and send this address to the Accumulator appropriately marked as an array, and proceed to enter the entire array into task memory beginning with this location. To accomplish the transfer (or load) the PMU needs a Task Memory Pointer, for addressing purposes. Also required is a counter to count each word as it enters. The Array may already be located in the Task Memory, in which case it usually must be picked up and moved, as in a Memory-to-Memory transfer, since the Accumulator version of the Array is subject to modification, while the original array is not to be changed.

To avoid unnecessary array transfers, several additional instructions which are largely handled by the PMU or the RAMM PMU have been defined.

Load Column: The instruction addresses the first word of the column. The column dimension is read first from Memory and followed by consecutive words which comprise the column vector. The remainder of the operation is treated as a conventional vector load instruction.

Load Row: The instruction addresses the first word of the row. The row dimension is read first from Memory. Memory addresses are incremented by the column length to produce the desired row vector. The instruction is then treated as a conventional load.

Load Shape: The instruction addresses the Array as usual for reading the entire array. Reading stops with the last dimension word. The operation is subsequently treated as a conventional load instruction.

Monadic Shape: The dimension vector of the Array which presently resides in the accumulator replaces the entire array.

Reshape: The vector M is read from Memory and these dimensions of M replace the dimensions of A. If the total length of M is shorter than A, the size of A is appropriately foreshortened. If A is shorter, the A terms are repeated from the beginning until length required is satisfied. Cycling Array A is the normal process in dealing with A as a Matrix (i.e., see inner product).

Two pages of Task Memory are assigned to Matrix Operations, and each time the Array in the accumulator is modified, the Working Page is moved. This makes all Matrix Operations dynamic. The above Monadic Shape moves the dimension vector, and effectively POPs the other page pointer. Reshape thus finds space for the dimension vector even though the original vector was shorter.

Catenate: The dimension of Array M modify the dimension of Array A. If both are vectors, then lengths are added. If both are Matrices, then row dimensions are added. One combined Array is formed by transferring first Array A, then following it by Array M.

Catenate by Column: The column dimensions of M are added to A. A single array is formed by reading a column of A followed by a column of M until both arrays are completed. The number of rows should be identical. (If not, the dimensions of M governs as in reshape and A will be truncated or repeated as required).

Laminate: An additional dimension will be formed by increasing the dimension vector. The new dimension will be length 2. The two arrays are assembled as in Catenate.

Laminate Column: A new dimension is formed with the contents of the last old dimension (i.e., Vector length becomes Row length) and the column length becomes 2. Arrays are merged by alternating words of each.

Load Transpose: The Row and Column dimensions are exchanged. The Memory reads each row in sequence rather than columns first.

Transpose: Same, except applied to the Array in the accumulator.

Reverse: The Array in the accumulator is addressed backwards, to produce the new array.

Store: The accumulator is stored in the Task Memory at the address supplied. Store is accomplished according to specified data types, for example there is an instruction Store Integer Vector. Overflow and other indicators are provided for all store operations.

5.6.4 PMU Only Instructions

The instructions listed in this section are taken from an early reference [5.1] and may have been changed. The Branch and Task Memory Instructions are:

1. NOP No operation
2. XEC* Execute instruction located at ADDRESS
3. TRU Transfer unconditional
4. TRS* Transfer to subroutine
5. INC Increment memory contents
6. DEC Decrement memory contents

*Explained further in [5.1].

The PMU Scratch Pad (PMUSP) Instructions and Stack Operations are:

1. LDSP Load scratch pad register
2. STSP Store scratch pad register
3. TDSP* Transfer on decremented scratch pad:
   PF designates PMU Scratch Pad register and R equals decrementing amount. If the decremented PMUSP value equals zero, a transfer is made to the ADDRESS, otherwise the PMUSP register is replaced with the decremented value.
4. APST* Advance Program Stack Pointer:
   The pointer in PMUSP designated by PF is incremented by one and the program counter stored in this PMUSP register, and the ADDRESS value is put in the program counter.
5. RPST* Return program stack pointer:
   The 12 least significant bits of the memory location specified by the PF replaces the contents of this location are decremented by one.
6. AAST* Advance accumulator stack:
   The contents of the A stack pointer located by PF field, are incremented by one, and the contents of the accumulator are stored in the incremented memory location. Incremented A stack pointer is restored.

*Explained further in [5.1].

7. RAST* Return accumulator stack pointer:
   Contents of memory location specified by PF are sent as an operand to APQ. The RAST OP-CODE is interpreted as a NO MODE load accumulator and the value of the A-stack pointer is decremented and returned.

8. AEST Advance external device stack pointer:
   Similar to AAST.

9. REST Return external device stack pointer:
   Similar to RAST.

For some unknown reason Raytheon refers to the PMUSP as the FSP for Fetch Scratch Pad in this section.

5.6.4.1 Load and Store Instructions

Two basic load operations (LD = load and LN = Load Negative) are combined with four mode options (A = fixed point, F = floating point, C = complex and I = integer) and three word length options (blank = full word, LH = left half and RH = right half) to produce 24 load instructions. The load instructions actually cause a mode change to the designated mode. All the load instructions are listed in [5.1], but they are not included here because of the probability that they will be changed.

The store instruction STA is defined for full word, left half and right half word. A Deferred Store instruction, DST, is also defined (see subsection 5.2.4).

5.6.4.2. Input/Output Instructions

The development of an adequate set of Input/Output instructions was impeded by the lack of definition of the relationship of the DPE to the external

*Explained further in [5.1].

subsystems. However, one major instruction, LDE or Load External, is defined in which bits 8 to 11 specify the various subsystems the PE may wish to communicate with. Examples (and assigned code) are as follows: Main memory or RAMM (0), BORAM (1), Matrix processor (2), bulk processor (3), I/O #1 to #4 (4-7), DPE #1 to #4 (8-11), Master Executive or MEC (12), System Clock (13), Operator's Console (14), and undefined (15). (Note the present design calls for a Signal Processing Element rather than a matrix processor and a bulk processor.)

The processing of the Load External instruction in the PMU involves placing a 40-bit word on the external cable to the I/O subsystem. The 40-bit word is composed of bits 8 to 15 of the instruction plus the full 32 bit word from the memory address specified by the effective address.

Besides Load External instruction, other I/O and interface instructions such as STE (store external), LDB (Load BORAM) and STB (Store BORAM) would also be implemented. Since all I/O instructions have the same format to the DPE and merely get interpreted differently by the device, new subsystems and instructions can be added without affecting other elements.

In summary when the list of AADC array instructions is examined (including reduction, inner product, outer product, index generator, ravel, dimension, catenation, lamination, rotation, transposition, reshape, take, drop, reversal, expansion, compression and many array manipulation instructions), it should be obvious that the Data Processing Element has many of the features necessary to execute the APL language directly in hardware [5.3 and 5.6].

5.7 DETAILED DESIGN

The detailed design of the PE is considered beyond the scope of this report. Section 7 of Reference 5.1 contains 84 pages of logic diagrams, wiring diagrams, logic equations and explanation which represents the detailed design of the PE. Although parts of some diagrams are unreadable, this section seems to be quite satisfactory for further study in the detailed design of the PE, including a detailed simulation. Raytheon's 1972 reports are not available at NPS and therefore it has not been determined if part of Reference 5.1 has become outdated [5.3].

5.8 CONCLUSIONS

5.8.1 Current Status of DPE Design

The current status of the Data Processing Element as reported at the 1973 AADC Symposium in January 1973 [5.4] is that:

1. The Raytheon interim report [5.3] has been superceded by a final report dated December 1972 (Not yet available at NPS).
2. Part III of the Final Report referenced above is a DPE Users Reference Manual describing how to use the DPE as it now exists. The DPE now exists as two simulations in APL - one simulating the PMU and the other simulating the Arithmetic Processor. This document formally defines the basic DPE operations by describing the operations at the bit level using APL. The DPE simulator can be used to verify programs written for the DPE. With fairly light load (about 10 other users) it takes about 15 minutes of elapsed time to complete 1000 additions on the DPE simulator.
3. The DPE Advanced Development Model (ADM) is scheduled for delivery in March 1974. The PMU for the ADM will be four modules (compared to one card on final AADC version) with about the same number of modules for the Arithmetic Processor. The ADM will operate at 2 MIPS using 11 nanosec/gate off-the-shelf logic. It will use a 4K, 36-bit, 150-nsec Task Memory.
4. The ADDRESS field has been extended to 16-bits by dropping the R field. This means that 64K of virtual memory can be addressed and thus any program module or array can be 64K

words long. A program may still have many program modules.

5. All arithmetic operations are completed as double precision operations. The results is then stored, as specified by the result data type, as a 32-bit integer, a 24-bit mantissa and 8-bit exponent real number, or a 56-bit mantissa and 8-bit exponent double precision number.
6. The DPE now has a multi-bus system feature that is actually a one word input queue that is always available for access from the bus. Thus any unit can communicate with a DPE by simply sending a word on the bus. The unit does not have to test for DPE busy or wait because the input queue is always available (every 150 nsec).
7. The Program Management Unit (PMU) is a modular stand-alone minicomputer. It has its own instruction set and operates on 16-bit words. It can address all 64K words of virtual Task Memory. It performs support for the Master Executive Control (MEC) using a microprogrammed hardware ROM and special stack instructions.
8. The PMU is also being considered as the external I/O control unit - see chapter 2.

5.8.2 Conclusions and Future Research

I will have to agree with Raytheon in that, "...this has been one of the most far reaching and significant studies with respect to the Processing Element Analysis, Design and Architecture. The tremendous capabilities incorporated into the AADC system will have an effect upon future computer architecture of any system attempting to make machine design more compatible

with High Order Languages. Conversely, HOL design can be improved by a consideration of the architectural concepts of AADC."

When the size and cost of PE is considered in addition to its tremendous capabilities, this design has to be the biggest breakthrough in computer hardware development since the transistor.

The problem now - assuming the PE can be built according to the design specifications - is to develop the rest of the AADC equipment to effectively utilize this powerful processor and to develop High Order Languages and Problem Oriented Languages so that the user can easily and effectively program the powerful AADC system. The AADC with all its power is not going to make any significant impact on any of the major computing problems unless the AADC also reduces the cost of software development and maintenance.

Problems on the PE

5.1. Design a four bit at a time fixed point multiplier using,

a) AND, OR and NOT gates,

b) NAND gates.

5.2. Design the polynomial execution hardware using,

a) ax + b modules,

b) register transfer modules (RTM),

c) logic gates.

5.3. Assume the instructions are not uniformly mixed at the ratio of 7 short instructions to 3 long ones, but are bunched so that, for any set of 16 sequential instructions, the ratio is significantly different. Calculate the actual throughput for different ratios. Calculate the probability of getting 16 instructions with a given ratio when the long term ratio is 7 to 3. Plot the actual throughput verses the probability of getting that throughput (i.e., that short term ratio).

5.4. Calculate the expected queue length in each case in problem 5.3. What would be the significance of doubling APQ?

5.5. Calculate the actual throughput for various ratios of branching along the less probable path, assume a uniform mix ratio.

5.6. Construct an example where the execution time is significantly reduced, as well as the length of the program, by Parenthesis Control.

5.7. How many coefficients are needed to obtain 32 bit accuracy for each of the functions in Table 5.2.2? How many for 20 bit accuracy? What is the minimum set of coefficients to calculate all the functions? Discuss the tradeoff between speed of execution and the amount of storage for the coefficients.

5.8. Estimate the time to complete each of the vector and matrix operations. How does the operand fetching time compare to the actual execution time?

5.9. In APL the matrix product has been generalized to apply to any two operations as well as the standard multiply and add operations. Assuming this can be done on the PE and that the execution time for each instruction is equivalent to add (i.e., takes 100 nsec), estimate the time to complete this operation for a matrix with N elements. Compare it with the standard matrix multiplication time. Also compare the instruction fetching time to the execution time for this operation. (An example of the use of this operation may involve the equal and add operations to find the sum of the number of places that two matrices have the same elements.)

5.10. (a) Construct a set of HOL constructs to take advantage of the powerful vector and matrix machine language instructions in the PE.

(b) Draw a flowchart of a compiler to convert the HOL constructs into PE instructions.

(c) Check the operation of your compiler by writing a computer program and testing it.

(d) Derive algorithms that do not limit the size of the matrices in the HOL. Hint: This probably involves partitioning the matrix and may involve recursive calls.

(e) Repeat (b) and (c) above for the algorithms in (d).

(f) Estimate the execution speed for the operations in (a) and (d) above.

5.11. How do the execution times for the algorithms in question 5.10 (d) change if there are 32 registers capable of manipulating 5 x 5 matrices?

5.12. If the AP scratch pad were increased to 32 registers could the cofactor mechanism discussed in the report be generalized to handle 4 x 4 and 5 x 5 matrices.

5.13. Determine the rationale for the statement, if it is true, that multiply and division algorithms are simplier for the sign and magnitude number system than for 2's complement number system.

5.14. What algorithms exist in the literature for fast multiplication and division using 2's complement, 1's complement or sign and magnitude number systems?

5.15. What are the advantages and disadvantages of using a module N number system for AADC? Hint: consult notes by Ray Nilson, UCLA.

5.16. Simulate the block diagram shown in Figure 4.5 of Reference 5.1 (which is a detailed version of Figure 5.3) and get a listing of all the functions which could be produced by selecting various control signals. Which ones might be useful for micro programming and should be added to the macro instruction set.

5.17. Count the total number of OP codes defined in this report. How many micro instructions can be added into the 8-bit macro OP code? What is the minimum set of micro instruction that must be added to the PE so it can be an effectively micro programmed.

5.18. How could P and V operators [Dijkstra 5.7] be implemented in the PE? (The P and V operators are two primitive operations used to simplify the communication and synchronization of processes or tasks, the primitives prevent any group of tasks from blocking each other and causing a deadlock. The primitives operate on non-negative integer variables called "semaphores" and have the property:

1. V(s): s is increased by one in a single indivisible action; the fetch, increment and store operations cannot be interrupted.
2. P(s): If s is not zero decrement s by one in a single indivisible action. If s equals zero, the P operation must wait until s is not zero.

The indivisibility of the P and V primitives assures the integrity of the value of the semaphore [5.8, section 3.3].)

References for the Processor Element (PE)

5.1 AADC Arithmetic and Control, Functional Block Diagram, Design, Analytical Study; Raytheon Company; Report No. BR6154; December 1970; NAVAIRDEVCEN Contract N62269-70-C-0210; Unclassified-NOFORN; AD-880-510; (44 NPS)*.

5.2 AADC Development Program Progress Report Number 10; R. S. Entner, NAVAIR-SYSCOM Code 5333F4; May 31, 1972; Unclassified; (78, NPS).

5.3 Interim Report for the Arithmetic and Control Logic Design Study for AADC; Raytheon Corp.; NADC Contract No. 62269-72-C-0023; May 1972; (76). (Section 4 is also available as Enclosure 4 to [5.2].

5.4 All Applications Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceeding not yet available.

5.5 Report on the Determination and Specification of the Preliminary Instruction Repertoire for the AADC (U); System Consultants Inc.; February 1970; NAVAIR-DEVCEN Contract N62269-69-C-0574; Unclassified; AD-867-055; (24, NPS).

5.6 APL/360 Reference Manual; Sandra Pakin; Second Edition; Science Research Associates; 1972; (NPS).

5.7 Solution of a Problem in Concurrent Programming Control; E. W. Dijkstra Comm ACM 8, 9 (Sept) 1965; p 569; (NPS)

5.8 Logical Design of Operation Systems; Alan C. Shaw; Draft copy of a book; Dec 1972; Section 3.3; (NPS).

5.9 Final Report for AADC Arithmetic and Control Logic Design Study; Part I, II & III; Raytheon Company; Report #BR-7162; Aug 1972; Unclassified; AD-909-00 (NPS).

*AADC Bibliography number and available at Naval Postgraduate School

Chapter 6

M A S T E R

E X E C U T I V E

C O N T R O L

TABLE OF CONTENTS FOR MEC

TABLE OF CONTENTS FOR MEC (Cont)

List of Figures

List of Tables

Glossory of Terms for MEC

A&C - Same as Processor Element or PE for this chapter. Actually a PE is an A&C plus a Task Memory, see Chapter 5.

AM - Associative Memory: heart of hardware MEC and used for fast searches of current PMID words and hardware resource words.

Baseline - The largest AADC architecutre designed to satisfy worst-case conditions. It consists of a large BORAM, RAMM, several PEs each with its own TM, dedicated I/O units, a high speed I/O multiplexor and probably a hardware MEC.

BORAM - Block Oriented Random Access Memory: stores PMs in blocks of 128 to 512 words.

Bulk Parallel Processor - Same as Matrix Parallel Processor.

MEC - Master Executive Control.

MID - Mode Independent Data: data that is used to communicate between PMs and between modes.

MINCOMS - Multiple Internal Communication System: a standard interface to the rest of the aircraft.

MMM - Multiple Memory Multiprocessor: an AADC architecture similar to Baseline except the PEs have no TMs and Program Modules are executed from RAMM.

MPP - Matrix Parallel Processor: for AADC parallel processing of radar signals, video signals and multiple targets. Also called Bulk Parallel Processor and Signal Processing Element.

OS - Optimized Simplex: same as OSP.

OSP - Optimized Simplex Processor: the minimum AADC architecture consisting of a single PE with its TM, RAMM, BORAM, I/O interface and possibly a MPP.

PE - Is synonomous with Processor in this section; a powerful serial processing central processing unit or CPU capable of executing 3.3 MIPS (Chapter 5).

PM - Program Module: a segment of a program.

PMID - Program Module (Task) Identification Words.

Processor - Short for Processor Element or PE.

RAMM - Random Access Main Memory: used to store mode independent data and to buffer I/O.

ROM - Read Only Memory: used by hardware MEC for permanent data.

TDM BTM - Time Division Multiplexed Block Transfer Multiprocessor: an intermediate AADC architecture similar to the Baseline system but with no hardware MEC.

TM - Task Memory: 4K word random access memory attached to a PE for holding the currently executing PM and temporary variables.

CHAPTER 6

MASTER EXECUTIVE CONTROL

6.1 INTRODUCTION AND SUMMARY OF RESULTS

This chapter presents the results of a design study of the executive systems, or operating systems, for the All Application Digital Computer. Each of the functional building blocks comprising an AADC system must perform its functions under the guidance provided by an executive system called the Master Executive Control (MEC).

This chapter is a report on the design study for the MEC and includes design philosophy, design tradeoffs, MEC capabilities, operating characteristics, MEC evaluation criteria, and methods of implementing MEC functions including some English language flowcharts. The chapter is based primarily on a design report by Honeywell [6.1] and a paper [6.5]*.

6.1.1 Introduction

The development of the MEC is based on advanced technology and methodologies such as:

1. AADC architectures based on modularly expandable functional building blocks,
2. New memory development and complex memory hierarchy,
3. LSI packaging,
4. Micro programming,
5. A powerful Processor Element,
6. Multiplexed I/O.

This chapter is basically a simplified and shortened version of Reference [6.1].

*Unfortunately a later Honeywell report [6.7] was not available at the time of writing and is not included in this Chapter. See current status of MEC in Subsection 6.7.3.

References [6.2 and 6.3] give an initial view of the MEC and may be consulted for historical purposes. References [6.4 and 6.5] give a simplified version of the material in [6.1]. Whereas Reference [6.1] considers a total of ten combinations of hardware organization and Master Executive Control systems, this report will only consider three of the four recommended combinations plus one extra. The ten combinations include all possible combinations of the four hardware organizations with the three MEC systems, with two exceptions. The four hardware organizations include the AADC Baseline system, the Time Division Multiplexed Block Transfer Multiprocessor, the Multiple Memory Multiprocessor and the Optimized Simplex organizations. The three MEC systems include a Special Purpose Hardware MEC, a Software MEC using a Dedicated Processor and a Floating Software MEC. One of the combinations that is not considered is the TDM Block Transfer Multiprocessor organization with hardware MEC since this organization is essentially a Baseline organization with a failed hardware executive. (In [6.1] the two software executives for the TDM Block Transfer Multiprocessor organization are included along with the discussion of the Baseline organization.) The other combination which is not considered in [6.1] is the Optimized Simplex organization with the hardware MEC because of the simplicity of the organization and the relative expense of the hardware MEC.

The four recommended combinations are the Special Purpose Hardware MEC with the Baseline and the Multiple Memory Multiprocessor organizations, the Floating Software MEC with the TDM Block Transfer Multiprocessor organization and the Dedicated Software MEC with the Optimized Simplex organization. Notice that Honeywell, in the recommended combination, has changed the definition of the Optimized Simplex organization; the Optimized Simplex has a single Processor

lement (PE) designed as a minimal system but Honeywell has recommended a second rocessor to execute a Dedicated MEC. The justification for the second processor s that "a single processor with floating software does not exhibit the eliability, graceful degradation or speed deemed necessary for this system." ince the single PE is still a viable configurate for AADC, the Optimized Simplex ith Floating Software MEC is also included in this report. By their own dmission the Honeywell's report is "quite formidable and includes a large amount f detail", containing some 247 single-spaced pages - not counting the six ppendices which explain the basic assumptions of the study and calculation of he logic and memory requirements. It is hoped that this chapter will be a more eadable version of that material, but that it will still contain the essential lesign and operating information of the MEC.

This chapter is divided into seven sections including this one. The second section describes the hardware MEC on the Baseline system. This system, in a slightly modified version, could also apply to the Multiple Memory Multiprocessor organization. The third and fourth sections describe the other two recommended combinations of AADC hardware and MEC systems as described above. The fifth section contains the true Optimized Simplex Configuration with a floating MEC. The four sections above all contain subsections on system operation, MEC functions and operations and the basic criteria for comparing MEC Systems. The sixth section is a summary and critique of the Honeywell evaluation techniques and recommendations. The last section contains recommendations for further developmental study of the MEC, as well as recommendations for improving the course material in this chapter.

6.1.2 Summary of Results

This subsection is a quotation from [6.1].

> A study of MEC implementations has been completed for all system configurations mentioned in section 1.1, [actually section 6.1.1]. Where applicable, three methods of implementation were flow charted, timed, and evaluated. The resultant English language flow charts, and detailed memory and timing estimates are included in this report. Summary flow charts are included to serve as a key to the operation of the overall Master Executive Control.
>
> All implementations were studied with the functional and operational characteristics of the basic AADC concepts in mind. Wherever the MEC implementations required a characteristic of an AADC functional unit or a program module which is not specifically covered in the baseline definition, an explanation of this characteristic was presented.
>
> In order to effectively evaluate the MEC implementations studied, a list of attributes was formulated. Each attribute was assigned a weight corresponding to its assumed relative importance. For each system configuration, a table was constructed and the candidate implementations were scored for each attribute. From these tables a weighted sum for each implementation was obtained. This weighted sum is a measure of the efficiency of the implementation method when used in the particular system for which the table was constructed.
>
> The evaluation of implementations was complicated by the need for certain information which is not, as yet available. Examples of this sort of information are:
>
> - Total tasks in a system
> - Number of tasks in a mode
> - Average run time of a PM
> - Average number of MEC functions required per PM
>
> The recommended implementations for the four system architectures considered in this study were made with this in mind.
>
> In the Baseline and Multiple Memory Multiprocessor systems the special purpose Hardware MEC is recommended. This is due largely to its speed advantage, a factor of about four to one over the dedicated software, and eleven to one over

the floating software in the baseline system. Other factors which pointed to this recommendation are those relating to the fact that this unit is specifically designed for MEC functions. This report shows that a special purpose hardware implementation of a Master Executive Control unit can be very effective when utilized in a very complex multicomputer or multiprocessor system. The use of an associative memory in such a hardware executive can result in very high speed execution of executive functions. It has also been shown that the hardware complexity of such a unit will be considerably less than that required in a single general purpose processor. This infers that a special purpose executive should have cost, reliability, size, weight and power advantages over the use of an entire processor to accomplish the MEC functions.

The floating software MEC implementation was recommended for the Time Division Multiplexed Block Transfer system primarily because of graceful degradation, cost and the other related attributes of size, weight and power. The floating software is an ideal MEC implementation in a system which does not require a heavy executive load. The overhead time required for this implementation is quite formidable and greatly affects the computation time of some executive functions. The required storage of a MEC kernel in one processor at all times also places a restriction on the size of program modules.

The Dedicated software MEC implementation was recommended for the Optimized Simplex system due to its characteristics in every attribute except cost, size, weight and power. The use of a single processor (floating software) did not exhibit the reliability, graceful degradation or speed deemed necessary for this system. This appears to warrant the cost of the additional processor.

As a result of this study, the following conclusions can be drawn:

- The use of a special purpose hardware executive utilizing an associative memory can execute executive functions considerably faster than either software implementation.

- If a large enough quantity of special purpose hardware executives are built, they have the potential of being less expensive than a system processor dedicated as the executive.

- A special purpose executive can be made more reliable than the proposed system processors.

- A floating software executive has high overhead requirements and should only be used in a system with low executive function load.

- A 4096 word task memory should be sufficient for all software executive requirements.

- A software executive requires each system processor to contain a real time clock and a loop counter.

A section of this report has been devoted to the definition of MEC related studies which are felt to be necessary to insure the smooth evolution of the AADC concept. These recommendations were made to show the steps which would result in the implementation of a MEC to be used in a prototype AADC system.

6.2 HARDWARE MASTER EXECUTIVE CONTROL

The hardware Master Executive Control (MEC) is by far the fastest and the most powerful MEC for the AADC system, being four times faster than the dedicated software MEC and eleven times faster than the floating software version. The hardware MEC is the recommended version for the Baseline and the Multiple Memory Multiprocessor Systems.

The hardware MEC is responsible for the following basic functions:

1. Monitor the various processing elements in the system to meet the requirements of all (externally-requested) modes of operation of the aircraft.
2. Assign operational programs to the various processing units.
3. Supervise data transfer between units within the AADC.
4. Supervise the overall system operation, for such items as processor failures, interrupt requests, etc.

The design and operation of the hardware MEC as it pertains to the Baseline System is discussed in this section. The means of communicating between MEC and other AADC components is presented, along with a summary flow chart of the MEC operation. All the routines in the MEC are listed and two sample functions are presented in detail. Some of the material in this sub-section is taken from [6.5].

6.2.1 Applicable AADC Configurations

The Baseline configuration contains the BORAM for Program Modules, RAMM for mode independent data and I/O buffers, several PE's (probably at least three) each with its own Task Memory for serial processing, Matrix Parallel

Processor for fast parallel processing, a high speed multiplexed digital interface as an interface to MINCOMS, dedicated I/O units for any PE with heavy I/O requirements, a programmable channel selector switch capable of connecting any PE with any dedicated I/O Unit, and a special purpose hardware MEC for controlling the operation of the AADC system. The Baseline system is shown in Figure 6.1. For further description refer to Chapter 2.

6.2.2 System Bussing for Hardware Executive

Four distinct busses are used to transmit tasks and data throughout the Baseline system. This bussing concept allows the MEC to maintain an orderly flow of data throughout the entire AADC system. These are shown in Figure 6.1 and defined below.

1. Program Module (PM) Transfer Bus. This unilateral bus is used to transfer PMs for the BORAM to the processor (or PEs).

2. Data Bus. This dual width bus is used to transfer data simultaneously between PEs and I/O or mode independent data (MID) storage areas of RAMM.

3. Processor Bus. This bus allow each PE to communicate with its nearest neighbor. It is not capable of bypassing a failed processor. An alternate path around a failed processor is though the data bus. The primary use of this bus is in executing "special processing" PM's on adjacent PE's.

4. Executive Bus. This bus provides the communication and control between MEC and all other system resources. It is used to transmit all interrupt requests within the AADC system. It must be an ultra-reliable bus because the consequence of its failure could be catastrophic.

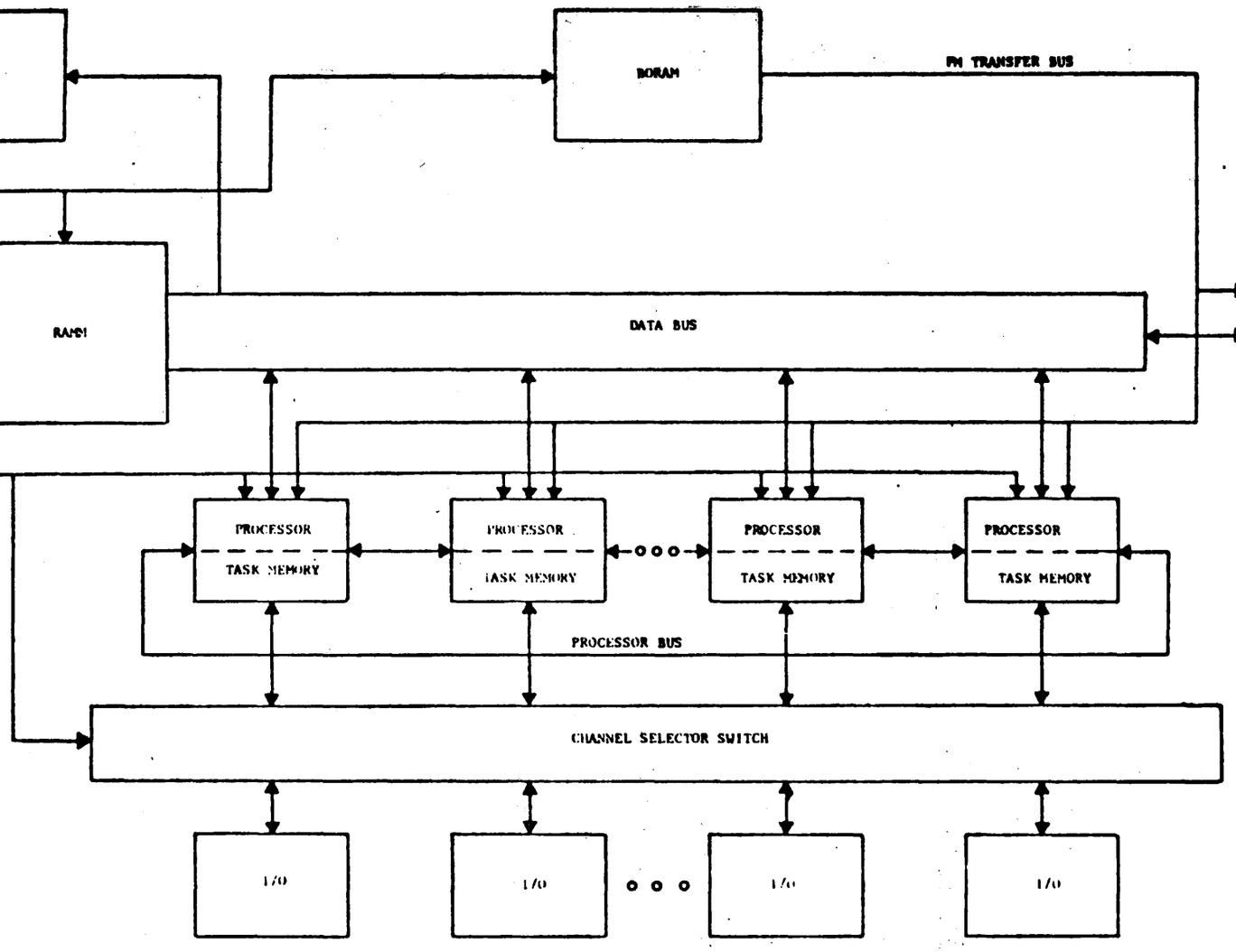


Figure 6.1. Block Diagram of the AADC Baseline System

The purpose of the buses is to allow the remainder of the system to continue its operation while PMs or data are being transferred without tying up other system resources, (except for a small amount of interference on the executive bus). The first two buses make it possible to have PMs being loaded, mode independent data from RAMM being transferred, and I/O data being transferred to different Processors simultaneously with the only chance of conflict, the occurence of a simultaneous request for MEC on the executive bus.

6.2.3 AADC Baseline System Operation

The job stream of the aircraft computer system is separated into a number of modes of operation. Each mode is segmented into a number of computational tasks called Program Modules (PM's). The proper operation of the system requires the computation of PM's at a given rate and in the proper sequence to effect completion of all required tasks in a given mode.

Initially, all PM instructions and data are stored as a block, or series of adjacent blocks, in the BORAM. Requests for input data required by each PM are stored in the PM as instructions. PMs initiate interrupts to the MEC when mode independent data is required from the RAMM. PMs make requests directly to the I/O memory of the RAMM through the buffer access switch for external input data.

The following types of PM's are assumed to be resident in the system:

- Iterative tasks which have real time requirements.
- Real time tasks which are activated after the completion of other system tasks.
- Real time tasks activated by external interrupts.

A program module can call in a block or blocks of storage from the BORAM via an instruction. This instruction is sent to the MEC as a data transfer request. By this means, a PM can call up its own successor, pull in another page of its own program, or cause an overlay of itself (or part of itself) while maintaining control of the processor. In the case where the PM continues to call itself, the PE is considered "dedicated" to the computation of that PM.

Two types of PMs which require "special processing" are considered in this study. These are the following: 1) PMs that require two task memories and 2) pipeline processing PMs. In the first case, the second task memory is accessed through the control unit interface. In the second case, a group of adjacent processors is configured, each of which will compute the proper PM. Special processing PMs are given assignment precedence over other PMs with equal time constraints.

PMs that require two task memories can be handled by allowing PMs to call up another PM or its own "second page". Therefore the special processing case in which a PM requires two task memories is not needed; however, the references to this case of special processing have been discussed here to emphasize this special capability.

In order to assist in the operation of the MEC, a set of resource words are available. There is a word for each resource in the system and these words describe each resource and its present and past states. These words are stored in memory and can be read out when desired. For each mode* the PM resource words may differ. Therefore, a new set of PM source words must be set active at the start of each mode.

---

*The term "mode" is borrowed from avionic applications where it refers to the type of mission or particular part of the mission, i.e., cruise, alert, search, attack, bombing, etc.

Idle processors will notify the MEC by means of an interrupt. Upon detection of processor availability, the MEC will determine which PM has the lowest assignment deadline and assign it to the available processor. At this time, the PM is transferred from BORAM to the processor's task memory. Thus, in the task memory of a processor, the program module exists as an ordered set of instructions and data ready for execution.

A PM may be assigned and processed a number of times while the system is in a given mode. Just prior to each execution, the PM is transferred from BORAM to task memory of the selected processor. Complete PMs are never returned from task memory to BORAM. Only selected data resulting from the execution of a PM may be written into the RAMM from the task memory.

6.2.4 Description of MEC Functions

The Executive Control functions in this section apply to all AADC systems, with only minor variations. The Executive Control recognizes when a mode change is encountered, evaluates on the basis of priority and importance criteria which (if any) program modules of the old mode are to be assigned to a processor. It also ascertains which tasks (Program Modules) are to be processed in the new mode and update the Input/Output data sensing for the requirements of the new mode.

The MEC is cognizant of the status of the "channel selector switch" for dedicated I/O, and handles all access conflicts. Executive Control also presents the I/O switch, interfacing with the MINCOMS system, with mode information such that the switch will properly multiplex I/O data. When a processor becomes available for a new task assignment, the MEC determines the proper task,

based on the priority and importance criteria of the tasks and the capability of the available resources. Upon receipt of a transfer request, the MEC initiates a transfer of data by properly signalling the resources which are to send and receive data. It then monitors the transfer of data and keeps other units from interferring with the transfer. Any contignous block of data (including tables) may be transferred by a single request by specifying the first address and the numbers of words to be transferred.

The MEC overhead from data transfer is small because the MEC does not continuously monitor the data transfer. Instead, the MEC initiates the data transfer and then releases control of the data transfer, and the MEC performs other processing while the transfer is occuring. Eventually either the data transfer will time out or a transfer complete interrupt will occur. At this time the MEC again enters into the previously initiated data transfer and takes control of the now completed (or possibly erroneous) data transfer. There is very little overhead involved in this process.

The MEC accepts all system interrupts, determines their priority, and processes the interrupts at the proper time. Internal interrupts are sent from system elements to MEC. External interrupt data are stored in the I/O memory while the interrupt (with its priority) are sent to the MEC.

The MEC also monitors the operation of all system resources. If faulty operation is detected, the control will initiate a test of the unit in question to determine and categorize a resource as operable, degraded, or inoperable.

6.2.4.1 Special Safety-of-Flight MEC Function

An additional function, that of the capability for safety-of-flight function, is performed independently of all AADC functions to assure "safety of flight". This function is not under control of MEC and acts independently

of the AADC system. The AADC acts as a backup to this system, and it will provide the computational capability in case the safety-of-flight computer fails. (The safety-of-flight computer may also be a AADC computer.)

Several assumptions and three possible methods of implementing the safety-of-flight function are presented in [6.1, pages 14, 15], but the actual implementation has not yet been designed.

6.2.5 Why a Special Purpose Hardware Executive Control Unit?

In a multicomputer system such as the AADC baseline system there are three basic approaches to the implementation of MEC:

1. Use of a system processing element dedicated to the performance of MEC functions (dedicated software MEC).

2. A software MEC program which is resident in BORAM and floats between available processing elements in the system for execution of executive tasks (floating software MEC).

3. Utilization of a special purpose ultra-reliable hardware unit designed specifically to handle the executive control (hardware MEC).

The floating software MEC and dedicated software MEC in such a system are executed on standard system processing elements which were designed to perform general purpose arithmetic, data handling, and logical operations. These processing elements were designed and sized to handle the programs of a typical aircraft mission.

A special purpose hardware unit could be optimally designed by restricting the complexity and computational capabilities to those necessary to perform the MEC functions. This unit could also take advantage of hardware concepts which are not conventionally included in system processing elements.

Two examples of such concepts would be the use of ROM to store the executive program which enhances the MEC reliability, and the use of associative memory to speed up the many search operations required in MEC processing. Thus it appears that the optimum implementation would be a special purpose hardware unit.

6.2.6 Philosophy and Operation of Hardware MEC

The MEC operation is complicated by the asynchronous nature of interrupts in a real time system. Depending upon the frequency of occurrence, the interrupts could become nested and thus cause excessive amounts of overhead computation and delaying the completion of the processing of some interrupts for a long time. Nesting of interrupts can be avoided by processing interrupts in a list structure, that is, during the processing of an interrupt no other interrupt can be processed. If an interrupt occurs it is placed on the list to be completed at another time. In order for such a system to be effective, the processing time required by each interrupt must be small. (At present, the longest interrupt takes 11 μsec to be handled). This is, of course, the simplest method of handling interrupts and is adequate providing no interrupts must be handled immediately.

The proposed MEC is based upon the following philosophy:

1. Keep each executive function simple to minimize execution time.
2. Utilize the search capabilities of an associate memory to minimize execution time of executive functions.
3. Do not nest interrupts.

4. Place interrupts on a list in order of priority.

5. Process interrupts to completion based on priority.

6.2.6.1 Allocation of Hardware Resources

In order to properly allocate the hardware resources of the system the MEC must be cognizant of the status of all hardware resources. This is accomplished through the use of resource identification words stored in the associative memory (AM) of the MEC. Thus each processor, bus, I/O unit, etc. has a unique associative memory word, called an identification word, which contains such information about the particular resource as:

- Failure status
- Assignment status
- Diagnostic information
- Resource identification.

Each of these word types are of different numbers of fields, ranging from 4 to 7, and different lengths from 8 to 22 bits. The MEC also has a MEC ID word. These hardware resource words are initially stored in the MEC read-only memory because of the assumed volitility of the AM. These resource words are explained further in [6.1 pages 22-28].

6.2.6.2 Program Module (Task) Identification Words

To assist in the assignment of PMs to processors, each PM has an identification word associated with it for each mode in which it is active. In each mode of operation, the proper PM identification words will be stored in the associative memory of MEC. These words are used to describe the past and present status of the program modules. Each word consists of twenty-five fields of

fixed or variable lengths. The fields contain the following types of information about the associated PM:

- Identity of PM
- Address of PM in BORAM
- Area of RAMM reserved for data storage
- Diagnostic information
- Priority in mode
- Scheduling information
- Precedence relationship to other PMs in mode
- Resources assigned to process PM
- Resources reserved to process PM
- Execution status.

The names of each of the fields in the PMID words, along with their size and whether fixed or variable length, is given in Figure 6.2. Most of the field names suggest their purpose except for:

1. The resource field which contains a designation for PMID word, Processor ID word, Bus ID word, Dedicated I/O word, Memory ID word, Matrix Parallel Processor ID word, transition PM, AM Word Available, MEC ID word or List Entry.

2. The PM class which contains a code to represent normal sequential processing, parallel processing, task memory not sufficient or pipeline processing.

The size of the PMID word is 148 bits which is significantly larger than the hardware resource words and, therefore, each PMID word would be segmented into several associative memory words. For further description of resource words see [6.1, pages 16-29].

	1	2	3	4	5	6	7	8	9
	RESOURCE TYPE	RESOURCE NUMBER	BORAM ADDRESS	NUMBER OF BLOCKS	RAMM ADDRESS	NUMBER OF WORDS	PM CLASS	PM TIME OUT	PIPELINE PROCESSOR NUMBER
FIELD LENGTH	4	8	10	6	8	5	3	1	2
FIXED/VARIABLE	V	F	F	F	F	F	F	V	F

	10	11	12	13	14	15	16	17	18
	IMPORTANCE CRITERIA	ITERATION PERIOD/ COMPLETION DEADLINE	TASK DEPENDENCE	DEDICATED I/O	ASSIGNMENT DEADLINE	IDENTITY OF PROCESSOR	FREE RUNNING TIME	TERMINAL TASK	TASK SYSTEM COLUMN
	5	10	32	2	10	3	5	5	4
	F	V	V	F	F	V	V	F	F

	19	20	21	22	23	24	25
	EXECUTION	ACTIVE MODES	OLD MODE	RESOURCE NUMBER IN MODE	STATUS	PRIORITY	DEDICATED PROCESSOR
	1	10	1	5	2	5	1
	V	F	V	F	V	F	F

Figure 6.2. Program Module Identification Word

6.2.6.3 Ordering of Tasks in a Mode

Due to the limited size of the task memory (assumed 4096 words) many programs will not fit into a processor. Therefore, all programs have to be segmented into a set of program modules that are to be executed in predetermined order: however, in order to save BORAM storage, some PMs can appear as parts of several overall programs. In the example to follow three different types of programs are to be processed. These are iterative programs, programs initiated by an external interrupt, and programs which are to be run once and only once in a mode.

To ensure that all tasks in a mode are completed on schedule, an ordered assignment of tasks must be made, using a procedure somewhat similar to that used in critical path (or PERT) problems. The important information for each task is: the iteration period/completion deadline, task dependence, assignment deadline, execution time and whether it is a terminal task. The critical path solution will determine the ordering or tasks, the first assignment time and the assignment deadline. This information is then stored in the PMID words.

Figure 6.3 shows a typical job stream consisting of four programs which have been segmented into ten PMs. Program one is an iterative program consisting of PMs 1, 5, 2, 3, 4; program two is an iterative program consisting of PMs 1, 5, 2, 6, 7; program three is an externally enabled program consisting of PMs 8 and 9; and program four is a one time run program consisting of PM 10. PMs 1, 5, and 2 are common to programs 1 and 2.

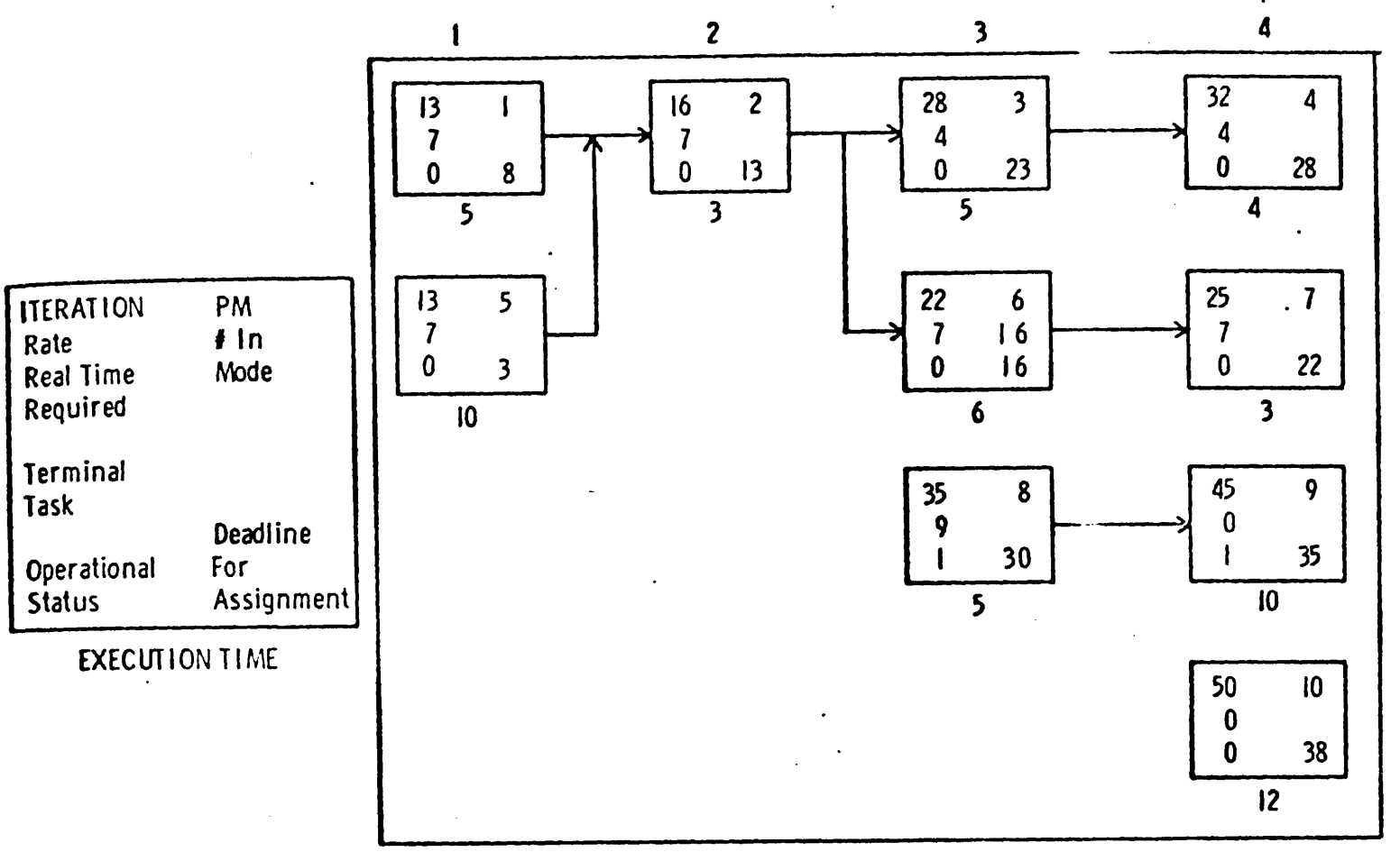


Figure 6.3. A Typical Job Stream Chart

For each mode of operation, a job stream such as shown in Figure 6.3 is drawn so that values can be calculated and stored in the appropriate fields of each PMID word. All terminal tasks (PMs which have no successor PMs) are placed in the rightmost column of the job stream chart. The iteration rate of iterative programs, the real time requirements of non-iterative tasks, is entered on the chart. From these values assignment deadlines are determined for all predecessor PMs with the assignment deadlines being the minimum difference between the assignment deadline of any successor PM and the execution time of the PM in question. Thus PM two has its assignment deadline determined by PM six instead of PM three and, therefore, its terminal task is PM seven rather than PM four.

Because it is assumed that there are enough system resources to successfully complete a job stream within the real time requirements of the system, no dynamic recomputation of critical path or complex scheduling algorithm is necessary. We have computed prior to the mission the assignment schedule of the PMs. Whenever a processor becomes available, a search will be made over all PMID words in the mode for the minimum value of assignment deadline and a zero in the operational status fields. (A zero in the operational status field represents PMs which have not been executed yet this cycle.) This PM is then assigned to a PE.

All PMs have the digital value representing iteration rate (real time required) decremented at periodic time intervals. If this value goes to zero and the operational status is still zero, it is an indication of insufficient resources. If the operational status of the PM is one and the PM is a terminal task the entire program is reinitialized. This ensures that the PMs are run once and only once per cycle. Note from Figure 6.3 that the operational status of all PMs in iterative programs (PMs 1-7) is set to zero. This in effect enables them for assignment. Program three (PMs 8 and 9) is not to be run until an external interrupt is received, thus the operational status of the PMs in this program is initially set to one. This in effect disables these PMs until the external interrupt is received and sets these fields to zero. Since PM nine has no terminal task (terminal task field equal to zero) program three is not reinitialized after completion. Program four (PM ten) is to be run once and only once. This is accomplished by initially setting the operational status to zero and setting the terminal task to zero. Thus the program once run will not be reinitialized again.

In the event of processor failure the number of computations that can be accomplished in a given time is diminished. This is noted by MEC when PMs time out (iteration field counts down to zero while operational status is zero). When this occurs the task load is decreased. The necessary information to accomplish this MEC function has been restored in MEC. It allows the MEC to selectively eliminate and/or halve the execution rate (double the iteration period) of programs on a priority basis.

6.2.7 Description of Hardware Executive

A special purpose hardware Master Executive Control - the recommended Executive for the Baseline and MMM systems - consists of three basic components:

1. A read-only program memory (ROM) to store the entire MEC program, resource identification words, and program module identification words for each mode of operation. This memory is estimated at 5122 32-bit words.

2. An associative memory (AM) - the heart of the MEC - which contains the PMID words for all active tasks (those in the present mode) as well as all hardware resource ID words for the system and the list of all uncompleted MEC tasks (LIST). The AM is estimated at 11,600 bits or 362 words.

3. A logic and control unit (CU) to recognize interrupts, save register data, control the associative memory, transmit executive interrupts and execute the Executive program. This unit is estimated at equivalent to 4000 logic gates.

Upon mode change the PMID words for the new mode are loaded from the read-only memory into the associative memory, while the old mode PMID words are still present. (i.e., there is room for two sets of PMID words). The AM with its ability to search simultaneously over all words in parallel provides the necessary high speed operation for the MEC.

6.7.1 Summary of Hardware MEC Operation

A summary of the operation of the hardware MEC is shown in a simplified form in Figure 6.4 and a slightly more detailed form in Figure 6.5. Whenever the system is initialized, an interrupt is received, or an executive task has been completed, a jump is made in the executive program to "start". The MEC then determines if there is presently an interrupt on the executive line. If no interrupt is present the highest priority class of interrupts is searched for the oldest entry, and this entry is then selected and processed. This allows first-in first-out operation within a given priority. The list of uncompleted tasks will never be empty since it contains household tasks which are not removed upon assignment, as well as interrupts which are removed.

If an interrupt is present on the executive bus, it is simply placed on the list of uncompleted tasks in with the proper time and priority designation. The MEC then returns to the task it was processing at the time the interrupt occurred. Thus, an interrupt is not processed immediately but it is placed on the list of uncompleted tasks.

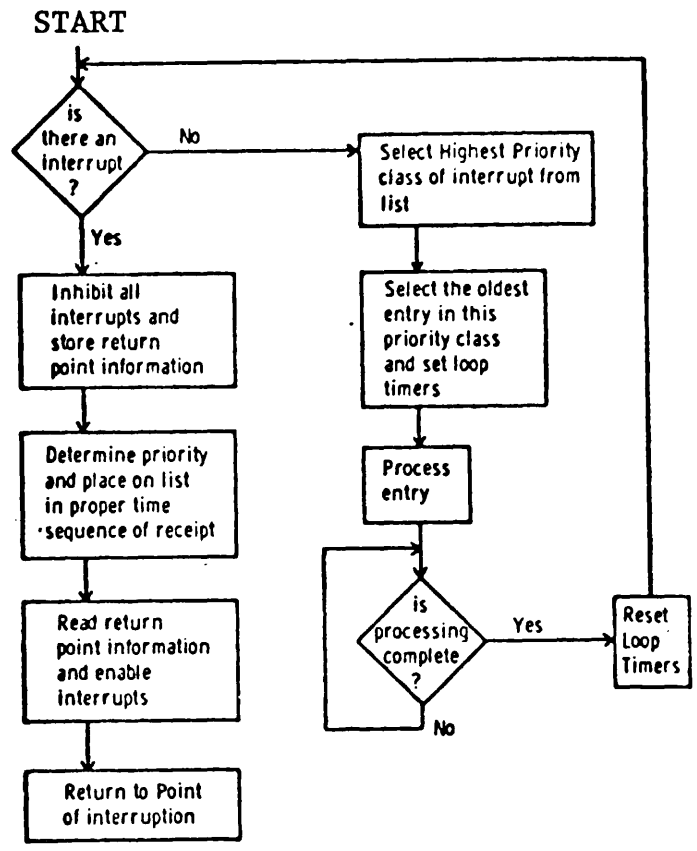


Figure 6.4. Simplified Flow Chart of MEC Operation

6.2.7.2 List of Interrupts and Routines

In order to accomplish the MEC functions of Subsection 6.2.4, a series of executive programs have been defined in flow chart form. These can be seen in detail in [6.1]. In order to keep execution time down, the basic programs (interrupts) have been segmented into a series of routines. A routine is placed on the LIST as a result of a decision made during processing of an interrupt. Those interrupts and routines defined are shown in Table 6.1. A detailed flow chart of two sample interrupts are shown in the next section. One entry in Table 6.1 is designated as both an interrupt and a routine. This is because this program can be initiated by both an interrupt and by means of a decision made while processing another interrupt.

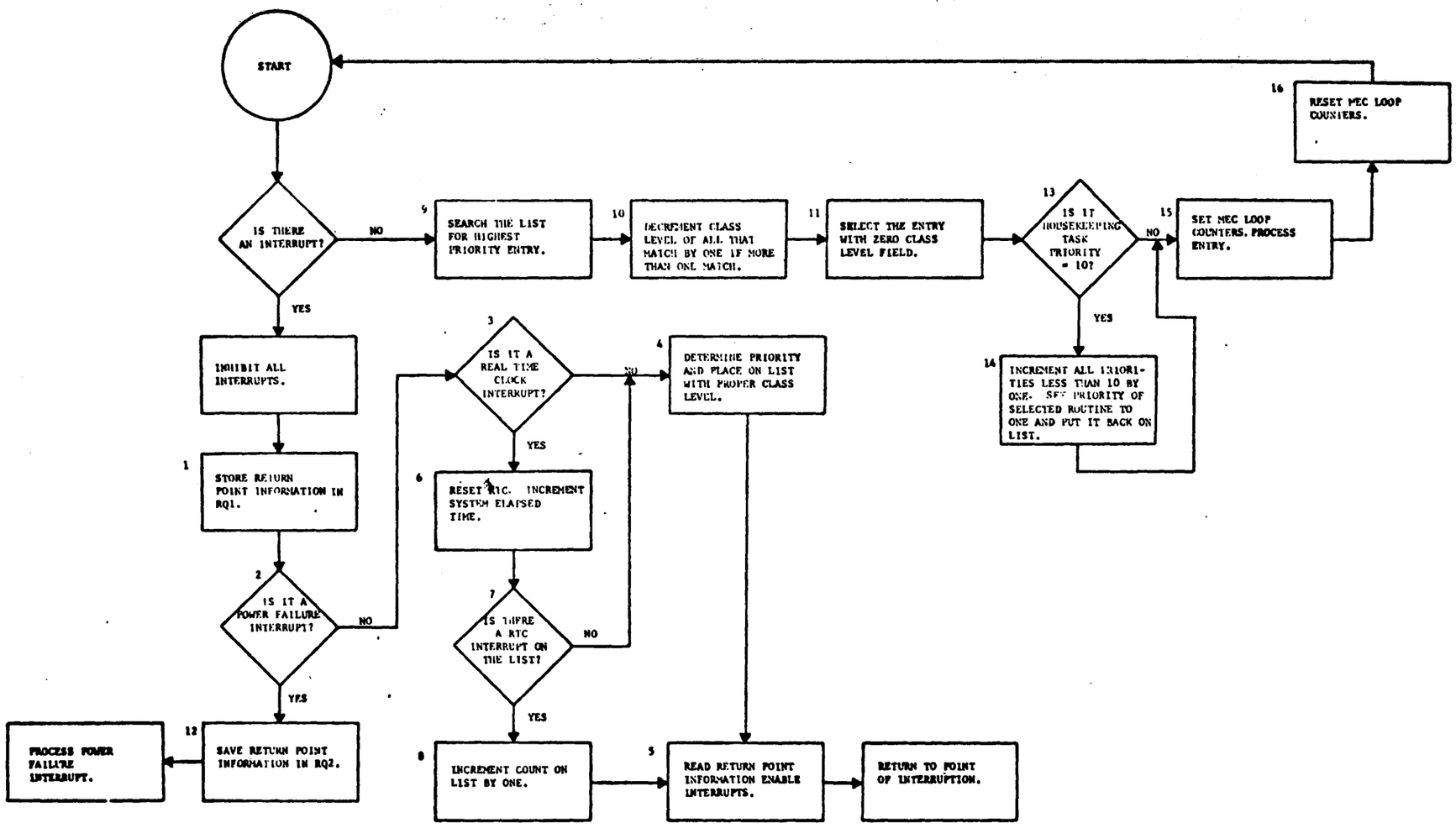


Figure 6.5. Summary Flow Chart for MEC Operation

Table 6.1. Types of LIST Entries for MEC Interrupt/Routines

PRIORITY	TITLE	INTERRUPT/ ROUTINE
31	Power Failure	Interrupt
30	Real Time Clock Failure	Interrupt
29	Master Executive Control Failure	Interrupt/Routin
28	Error	Interrupt
27	Loop Counter	Interrupt
26	Program Module Complete	Interrupt
25	External Program Module Enable	Interrupt
24	Mode Change	Interrupt
23	Real Time Clock	Interrupt
22	Channel Selector Switch Assignment	Routine
21	BORAM Test	Routine
20	RAMM Test	Routine
19	Bus Test	Routine
18	Processor Test	Routine
17	Data Transfer	Interrupt
16	Data Transfer Error	Routine
15	Memory Address Error	Routine
14	Program Module Address Error	Routine
13	Data Transfer Request	Routine
12	Program Module Reinitialization	Routine
11	Program Module Assignment	Routine
10 through 1	Miscellaneous Housekeeping	Routines

Except in the case of the data transfer interrupt, all interrupts are of higher priority than any routine. This is because routines are placed on the LIST as the result of processing an interrupt. The exception is made in the case of the data transfer interrupt because the transfer of data requires that the sender and receiver be operable. If requests to test hardware units are on the list, they should be processed before an attempt is made to transfer data. The channel selector switch assignment is of higher priority to preclude the possibility of transferring a PM to a processor and/or processing to begin before a dedicated I/O unit can be assigned.

In a hardware executive, the LIST is stored in the associative memory. A discussion of this LIST and the entries which are made are presented here for clarity.

When an interrupt or routine request is made, a request 36-bit word is stored in the associative memory. This word must store all the information that is required for MEC to process the request. The general form of the request word is shown as follows. Many of the fields shown are not used for most of the request words, but the form shown allows processing of all requests.

PRIORITY CLASS	CLASS LEVEL	IDENTITY	SOURCE	MEMORY	ADDRESS	NUMBER WORDS	BUS	PROCESSING STATUS BITS P_0, P_1, P_2, P_3
5	3	3	3	1	10	6	1	4 bits

Table 6.2 shows the contents of the fields for the requested words for each interrupt and routine. As can be seen many fields are blank.

Table 6.2 Request Words for MEC Interrupts/Routines

NAME	PRIORITY CLASS	CLASS LEVEL	OTHER NON-BLANK FIELDS IN REQUEST WORD
Power Failure Int.	31		No Request Word Generated
Real Time Clock Failure	30		
MEC Failure Int.	29		
Error Interrupt	28	0-7	SOURCE: RAMM, BORAM or PE#.
Loop Counter Int.	27	0-7	SOURCE: LOOP COUNTER in MEC or PE.
PM Complete Int.	26	0-7	SOURCE: PE#.
External PM Enable Int.	25	0-7	ADDRESS: Absolute PM# of a terminal task.
Mode Change Int.	24	0-7	ADDRESS: Address in ROM of the new mode's first PMID word; #WORDS: # of PMID words in new mode.
Real Time Clock Int.	23	0-7	IDENTITY: Records number of RTC interrupts received before one is processed.
Channel Selector Switch Assignment Routine	22	0-7	IDENTITY: Dedicated I/O Unit #. SOURCE: PE#.
BORAM Test	21		
RAMM Test	20		
Bus Test Routine	19		BUS: PM Bus or Data Bus
Processor Test	18	0-7	SOURCE: # of PE to be tested.
Data Transfer Interrupt	17	0-7	IDENTITY: In or out, error or successful; SOURCE: PE #; MEMORY: RAMM or BORAM; ADDRESS: First Memory Address; # WORDS: # words or blocks; BUS: Data bus or PM bus; PO: Set to 1 if data transfer must be retried.
Data Transfer Error Routine	16	0-7	SOURCE: # of PE to be tested; MEMORY: RAMM or BORAM; BUS: Data bus or PM bus; PO = 1 means bus tested; P1 = 1 means memory tested; P2 = 1 means processor has been tested; P3 = 1 means request active
Memory Address Error	15	0-7	SOURCE: # of PE causing error.
PM Address Error	14	0-7	SOURCE: # of PE containing PM
Data Transfer Request	13	0-7	Same as Data Transfer Interrupt.
PM Reinitialization	12		
PM Assignment	11	0-7	SOURCE: # of PE which made request.

Thus, the proposed request word organization would result in a very poorly utilized associative memory, unless most interrupts and routines have circuitry for only a few appropriate fields in the request word. In fact, five interrupts/routines have only the priority field, another ten have three fields or less and only three interrupt/routine come close to using the full 36 bits. These are the data transfer interrupt and the data transfer request routine with 33 bits each, and the data transfer error routine with 23 bits.

Thus, there are a total of 10 interrupts and 11 routines excluding housekeeping tasks. These 21 program segments along with the summary flow chart and housekeeping tasks comprise the entire executive program.

6.2.7.3 Reference to Flow Charts of MEC Implementation

The implementation of the hardware MEC is considered beyond the scope of this report at this time, but an outline of the appropriate sections of Reference 6.1 will be given. (It is planned to include this is a later version; see Subsection 6.7.2.)

The Section 2.1.5.5.1 [6.1, pages 40-45] describes the use of the executive bus which transmits all interrupts from other AADC resources to the MEC and which initiates data transfers between units of the system other than I/O data. The section describes the use of the active line, the acknowledge line and the reject line, and shows the sequence of words on the executive bus for three types of data transfers.

Sections 2.1.5.5.2 and 2.1.5.5.3 [6.1, pages 45-85] describe the implementation and English language flow charts of all the MEC interrupt handlers and MEC routines presented in the Tables 6.1 and 6.2 of the previous Subsection 6.2.7.2. Thus, there are 41 pages of flow charts and description of the MEC

implementation. Two sample interrupt flow charts are shown in Figures 6.6 and 6.7 as fairly typical examples. The PM Complete Interrupt flow chart is chosen because of its significance in changing PMs and modes. The Data Transfer Interrupt flow chart is chosen to show the complexity of data transfer handling. Some other flow charts are simpler; others are more complex. Another sample interrupt flow chart is shown in [6.5]. For a complete set see [6.1, pages 45 - 85].

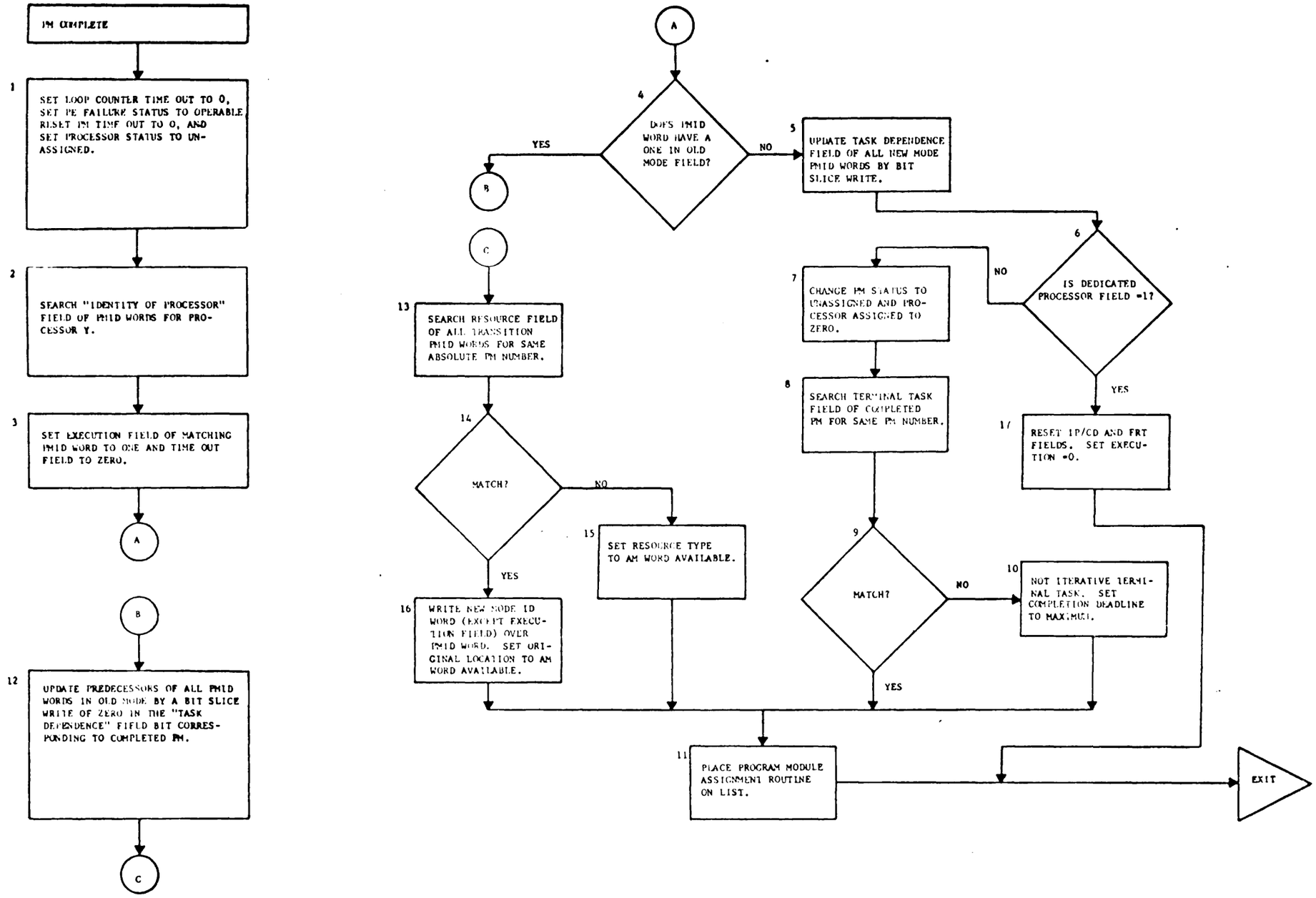
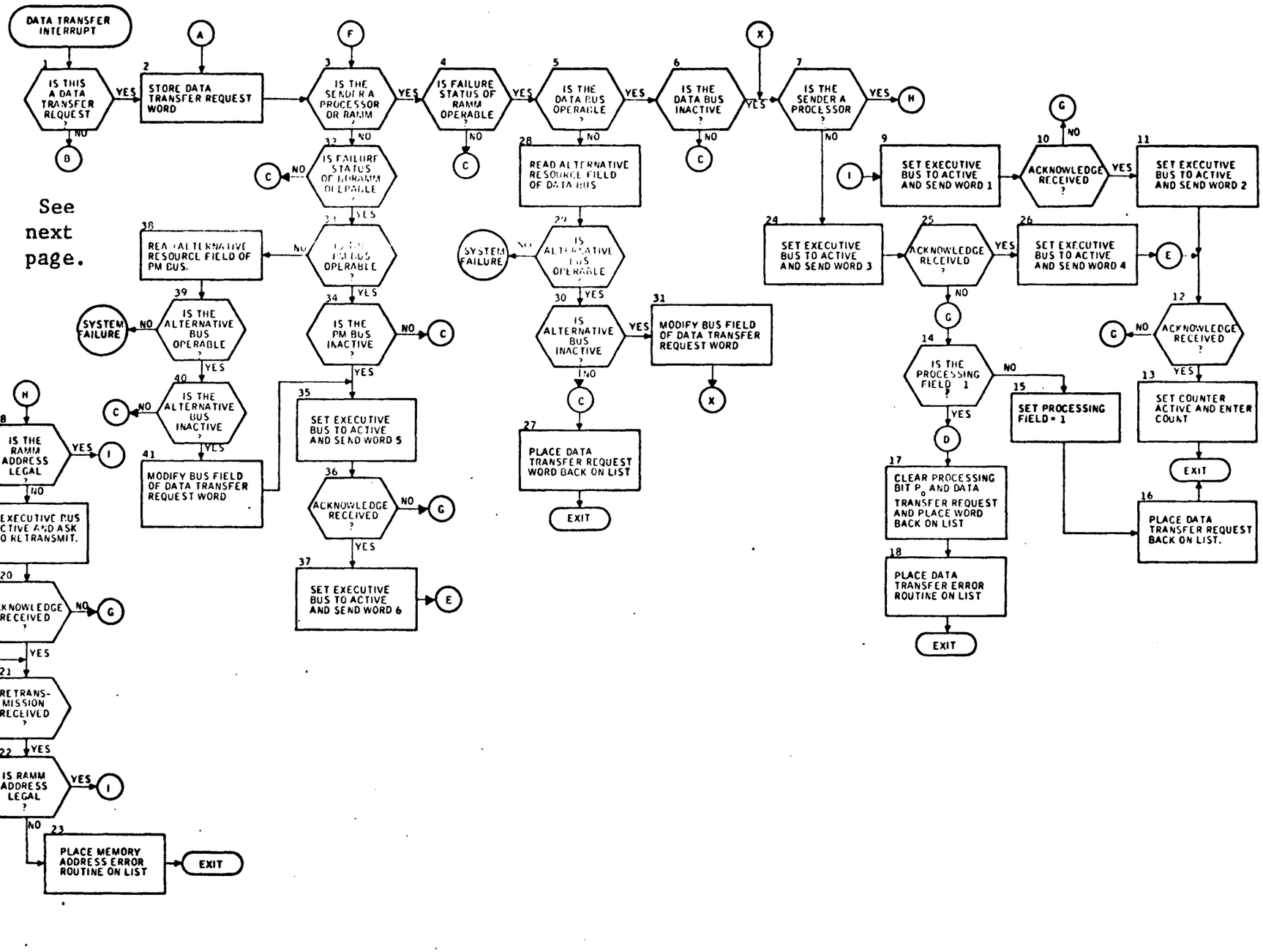


Figure 6.6. Program Module Complete Interrupt Flow Chart



See next page.

Figure 6.7(a). Data Transfer Interrupt Flow Chart

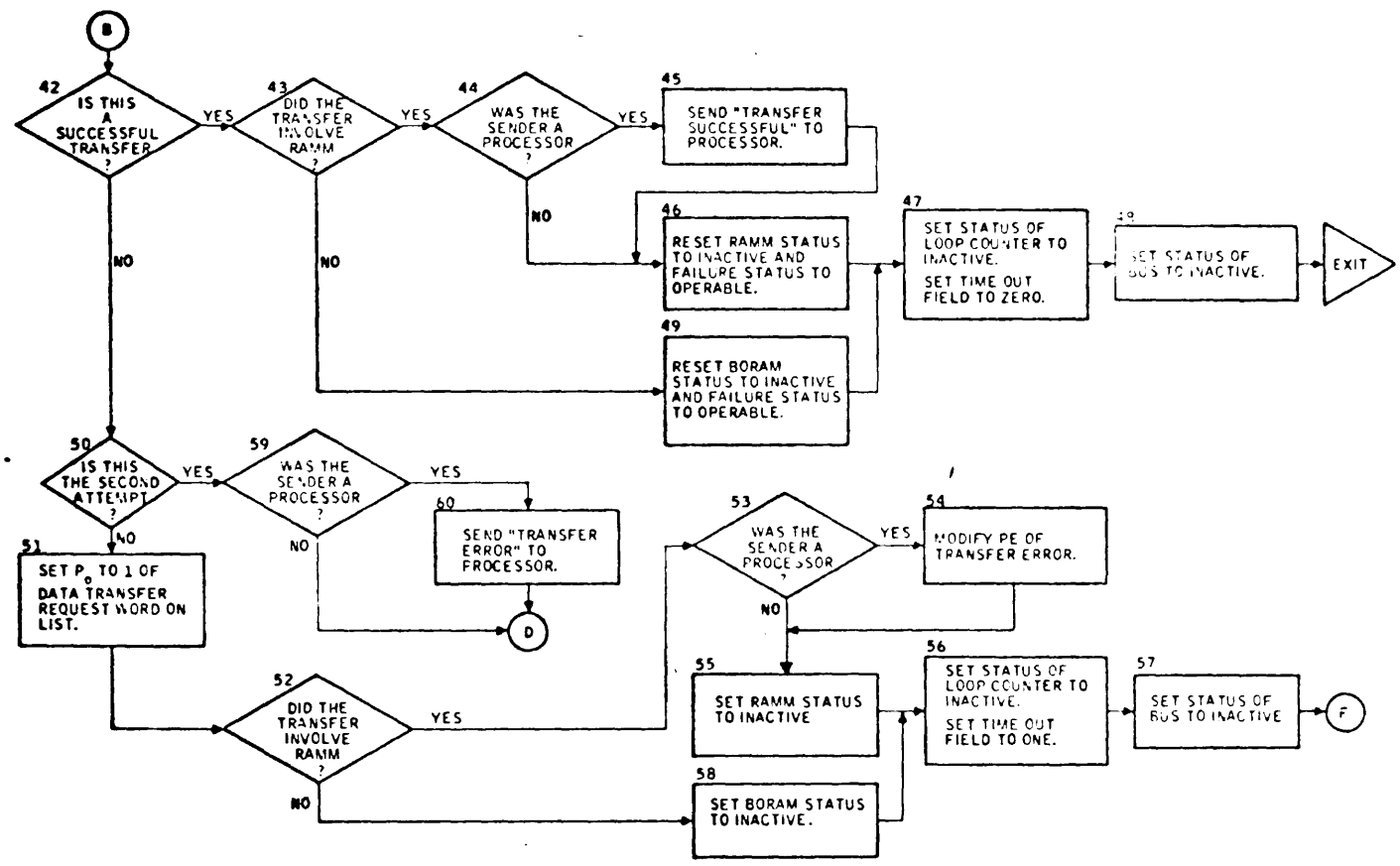


Figure 6.7(b). Data Transfer Interrupt Flow Chart (Concluded)

6.2.8 Summary and Preliminary Evaluation of the Hardware MEC

The hardware executive has several apparent advantages over the two other types of executives which will be studies for the baseline system. These are the following: high throughput, no constraints are imposed on PMs, does not require an AADC processor, and can be designed to be highly reliable. This executive has very simple software and is not very complex compared to the other executives.

Because the hardware executive has an associative memory and, therefore, can address data based upon some property of the data, the time consumed to do the searching required in executive functions has been significantly reduced. This has singificantly enhanced the throughput of the executive system. Also, since the executive does a particular function and not general calculations, the arithmetic capabilities required and the use of read-only memory to store the executive program allow the executive to be more reliable than any of the system's processors, thereby enhancing reliability. Since the executive function is performing in a special purpose computing element, executive design has no effect on the PMs and all of the AADC system's processors are available for processing of PMs. It is also expected that the overhead of the hardware executive will be minimal because the executive is always active (there is no dormant state). Also, the individual processors can be simplified because they do not need to process the "special" executive functions.

There are several disadvantages of the hardware executive such as complex "graceful degradation" and the requirement for special purpose hardware. In order to have "graceful degradation" of the executive, it must be able to switch to an executive operated on a system processor if it fails. This requires

development and storage of a back-up executive that could be significantly different than the original hardware executive; for example, the floating software MEC. Another disadvantage is the cost of design and development of the special purpose hardware executive. Also, the use of the special purpose hardware executive will be a viable system with the best processing capabilities for executive functions since it will be optimized to perform the executive function.

As discussed previously, the hardware MEC consists of three main elements. These elements and estimates of their complexity are presented below:

1. An associative memory which contains the following information. In each case the maximum storage requirement is shown.

   - Current-mode PMID words which are to be run once more (32 x 148 bits)
   - Old-mode PMID words which are to be run once more (32 x 148 bits)
   - List of uncompleted interrupts and routines (20 entries) (720 bits)
   - Other resource words
     Four processor (100 bits)
     Four busses (64 bits)
     Four dedicated I/O units (56 bits)
     64 words BORAM ID (896 bits)
     16 words RAMM ID (224 bits)
     MEC ID (11 bits)

   Total associative memory required = 11,543 bits

2. A read-only MEMORY which has the following information (In each case the maximum storage requirement is shown.):

   - The entire hardware executive program implementing the flow charts of interrupts and routines. (3109 words)
   - ALL PMID words for all modes of operation. Assume 32 PMs/MODE, 10 Modes; 5 ROM words per PMID word. (1600 words)
   - Memory address error test (20 words)
   - All resource words (93 words)
   - All mask patterns required for AM searches (30 words)
   - Priority and two Importance Criteria threshold for each mode (30 words)
   - Bus test (20 words)
   - Constants (20 words)
   - Associative memory micro instructions (200 words)

   Total ROM required = 5122 words x 32 bits.

3. A logic and control unit to consist of the following elements:

   - Five loop counters
   - A comparison counter
   - 20 thirty-two bit registers (RAM)
   - Macro program store counter
   - Micro program store counter
   - A shiftable argument register
   - A non-shiftable argument register

- Adder - Subtractor
- Real time clock
- A search results register
- A shiftable mask register
- A word select register
- Various control circuitry

Total estimated complexity in equivalent logic gates = 4000 logic gates.

Using the higher of the two costs given on page 12 of NASC progress report number 6, task memory is estimated to cost 5 cents per bit. If we assume twice that cost per bit for the associative memory and half that cost for the read-only memory, the cost of memory for the hardware MEC is $1154.30 + $4097.00 or a total of $5251.30.

Through discussions with the contract monitor it can be estimated that a processor arithmetic and control unit will consist of about 14,000 logic gates and 750 words of micro-program control memory. In logic gates the PE is 3.5 times as complex as the logic and control for the hardware MEC. If we assume $3000 is the cost of the PE, the logic and control of the MEC should cost approximately $860 ignoring the memory in the PE.

Thus, a hardware implementation of the MEC should come to a total of $6,111. This compares to a cost of $9,400 for a processor with a 4K task memory. Thus, a hardware MEC should cost about 65 percent of the cost of a processor.

The time required to compute all the interrupts and routines are shown in detail in Appendix F of [6.1]. As a measure of speed, the total nominal time required to run all interrupts and routines with the assumed MEC is 392 usec. This data is all taken from [6.1].

For all implementations in this report, the following requirements are placed on each processor (or PE) in the system.

1. Each processor must have a register which will recognize its own code when it appears on the MEC bus.

2. If a data transfer error is received by a processor, it must either save the data it was sending for transmission later or not attempt to use the received data if it was the receiver.

3. Each processor must generate a PM done interrupt at the completion of each PM.

In the report [6.1], it was shown that a special purpose hardware MEC could be built with the following advantages being obtained over software approaches for the AADC Baseline system:

1. Low cost (65 percent of a system Processor)

2. More reliability

3. Higher speed (4 to 11 times as fast)

4. Can take advantage of new hardware technology such as LSI and associative memories

5. Low overhead

The basic element of the MEC is a semiconductor associative memory. The use of a semiconductor approach to the associative memory* allows logic to be placed at every bit position (which allows full parallel output and equality searches) and construction of the memory LSI techniques. Because there is logic at every bit position of the associative memory, extremely fast equality searches can be made, thus, resulting in fast methods of determining the status of system resources and then allocating these resources. The use of the associative memory in the executive system enhances the MEC's speed significantly.

*i.e., the memory elements in the associative memory are also semiconductor LSI circuits.

In the future as executive systems become more and more complex, software approaches will approach or exceed their capabilities. A special purpose executive system utilizing an associative memory provides a high speed alternative to a software executive system. Since the hardware MEC has a high throughput capability, it will be able to accept a large increase in executive load and still provide the computational capabilities necessary to insure proper system operation without degrading system performance and reliability. In fact, the special purpose hardware MEC for the AADC system will do all of the above, and at the same time reduce costs.*

*Of course the hardware MEC has the largest design cost and therefore this actually assumes that there are sufficient number of hardware MECs produced so that the total cost per MEC is less than the cost for a Data Processing Element.

6.3 BACKUP MEC FOR BASELINE SYSTEMS

Since the Time Division Multiplexed Block Transfer Multiprocessor system is essentially a AADC Baseline system with a failed hardware MEC, this section can be considered either the backup Executive Control for the AADC Baseline System or the Floating Software MEC for the TDM Block Transfer Multiprocessor System. This report has chosen the backup MEC interpretation although the reference from which the material is taken [6.1] chooses the other interpretation. In fact, Honeywell's recommended Executive Control for the TDM Block Transfer System is the Floating Software MEC which is described in this section.

6.3.1 Applicable AADC Configurations

Figure 6.1 of Section 6.2 is the block diagram for the AADC Baseline System, and is essentially the block diagram for the TDM Block Transfer Multiprocessor System except that the separate hardware MEC is not available and a software MEC must operate from one of the PEs. However, the following changes must be made to the system (and to the corresponding section, 6.2.1):

1. BORAM. As well as all the operational PM, the BORAM contains all the Program Module Identification (PMID) words for each mode and all segments of the MEC software programs.

2. RAMM. As well as mode independent data and I/O buffers, the RAMM contains all resource words, current mode PMID words and MEC scratch pad areas. Since it is assumed that an Associative Memory is not available to the floating software executive system, the PMID words must be shuttled between the RAMM and the MEC task memory in lieu of AM search operations.

3. PE. All the PE must be capable of executing all the MEC functions. At any given time, one of these processors contains the entire MEC resident program or its transient active kernel.

4. Task Memory. The TM must be non-volatile; otherwise, the MEC task LIST would have to be stored in RAMM occasionally to privide a rerun point for restart after a power failure.

5. MEC. The floating software Master Executive Control routine will be able to operate from the task memory of any processor in either its fully active or partially active phase. The main duties of the MEC are the same as those listed in the second paragraph of Section 6.2; however, a categorization of these into the following four areas is useful: (a) input/output, (b) address translation or binding, (c) interrupt servicing, and (d) job initiation.

   These categories will be referred to throughout Section 6.3 on the floating software MEC.

The remainder of the system is the same as before. This includes the Matrix Parallel Processor, the high speed multiplexed digital filter, dedicated I/O units and channel selector switch.

The switch bussing, the system operation and the description of the MEC functions are the same as described previously in Subsections 6.2.2 to 6.2.4.

6.3.2 Implementation of Floating Software Executive

6.3.2.1 PMID and Hardware Resource Identification Words

Other than a change in the resource field for the Program Module Identification words, the PMID and Hardware Resource Identification words are the same as those for the hardware MEC. One other change is that the resource words are stored in the RAMM in three 256-word areas. One area is for active PMID which were previously stored in the AM. Another area is for the active initial PMID words that were previously stored in ROM. The third area is for the inactive PMID words which are used to assist in mode switching. Note the

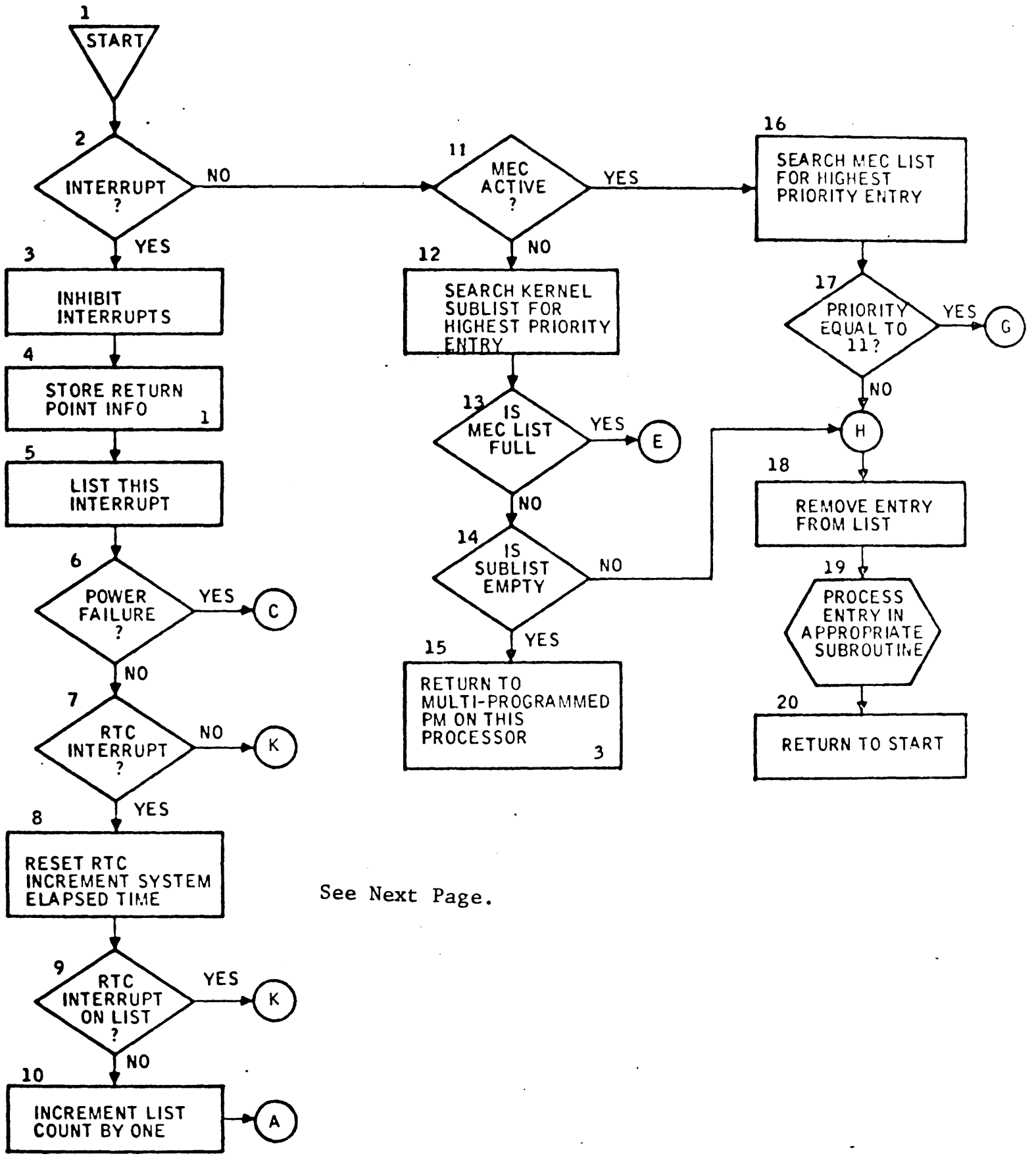
back up storage for the PMID words is the BORAM [6.1, pages 91, 92, 119].

6.3.2.2 Summary Flow Chart of the Backup Floating Software MEC

Again this section applies to either the Baseline system with a failed hardware MEC or the TDM Block Transfer Multiprocessor System.

The floating software MEC will pursue essentially the same logic flow as is given in Subsection 6.2.7.1. However, its logic must be partitioned into two phases. Some MEC operations, such as handling internal data transfers, loop counting, and real time clock interrupts must be performed by the always active kernel. Other operations must be performed by the entire MEC and when called for by an interrupt or PM, they must be listed if the MEC is not active, or they may either permit or force a MEC load. The PM complete interrupt permits the MEC to be loaded in the recently freed processor, but a power failure interrupt would force a MEC load. The mode change and external PM enable interrupts may also force a MEC load. The ensuing discussion will refer to the always active portion of the MEC as the "kernel" and to the entire routine as the "MEC". The MEC flow chart is given in Figure 6.8.

The same interrupt handling philosophy assumed in Section 6.2.7.1 is assumed here; any interrupt will be recognized and listed for later processing depending on its priority, but control will be returned immediately to the point of interruption. In the case the MEC is in its active phase, this situation is the same as the dedicated software in MEC case to be discussed later in Subsection 6.4.4; however, if only the kernel is active, then the point in interruption may be either in the PM or the kernel. Since the kernel is able to handle



See Next Page.

Figure 6.8 (a). Summary Flow Chart of Floating Software MEC Baseline AADC System

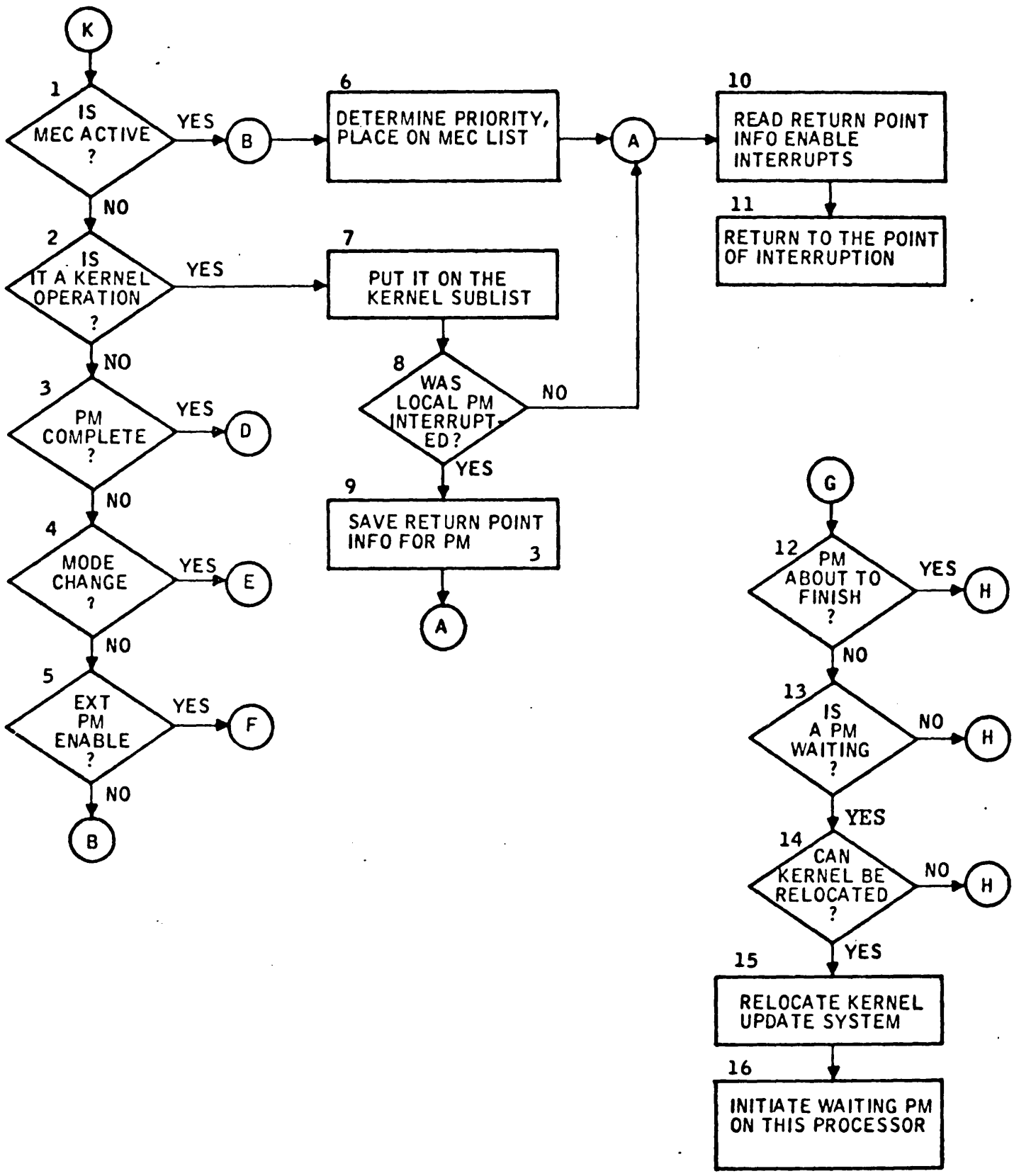


Figure 6.8 (b). Summary Flow Chart of Floating Software MEC Baseline AADC System

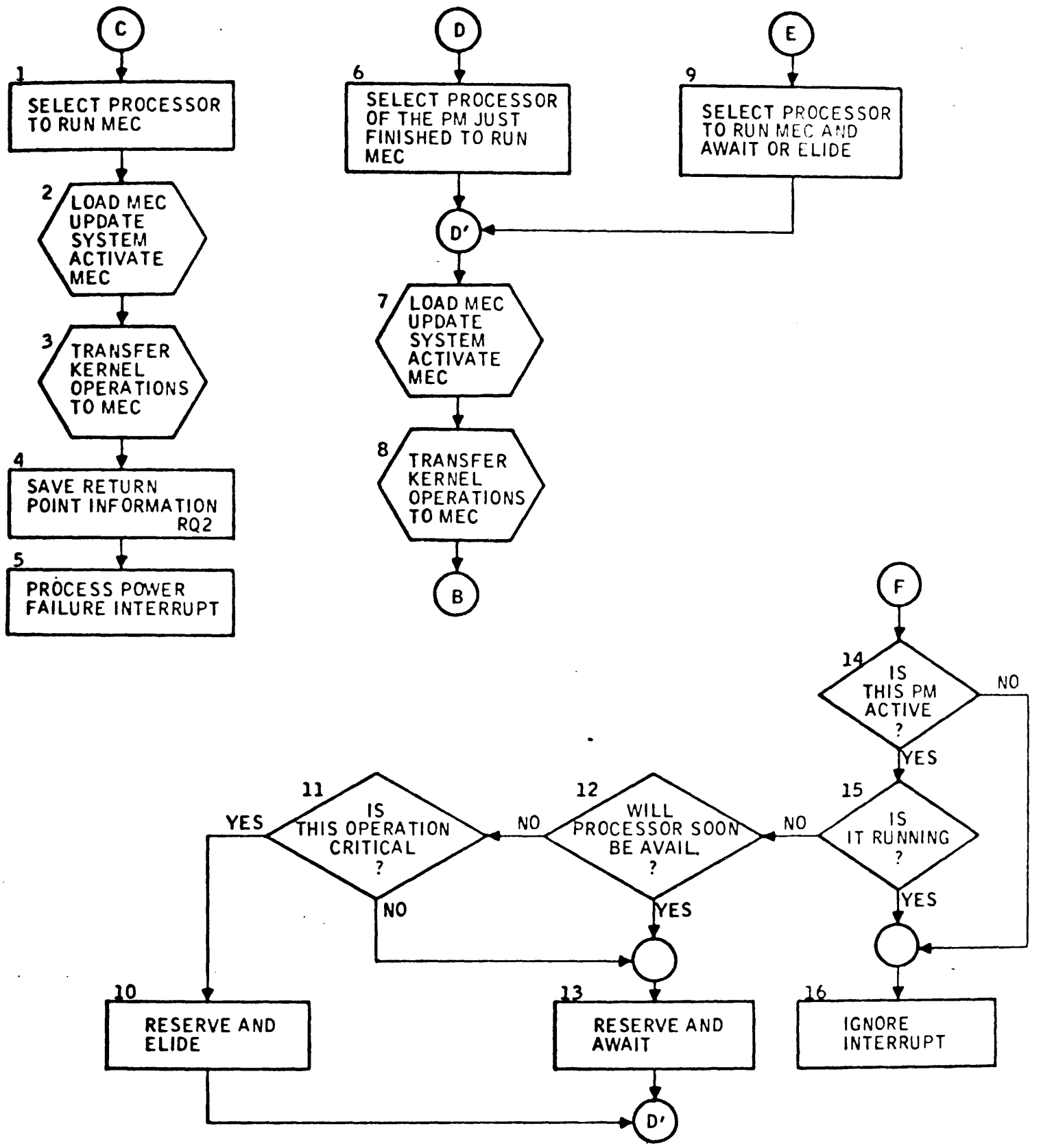


Figure 6.8 (c). Summary Flow Chart of Floating Software MEC Baseline AADC System (Concluded)

only the loop counter, real time clock and data transfer interrupts, it lists all others for later handling by the entire MEC in its next active phase but puts those it can handle in a special sublist (Table 6.3) for its own more immediate attention. The MEC task LIST and kernel sublist are stored in task memory.

Table 6.3 also indicates MEC functional routines and interrupt handlers as being either resident or non-resident. Although a 4096 word task memory may be able to contain the full floating software MEC it is not desirable to read in seldom used code every time the MEC is loaded. Those routines and interrupt handlers flagged as "non-resident" in the table are suggestions of coding that could be left in BORAM until actually needed.

If the kernel decides an interrupt, such as a power failure interrupt, must be handled immediately, it forces a MEC load into one of the PE. If interrupt is less urgent, the kernel may wait for a PE to finiah executing a PM before reassigning it. The details of handling interrupt is quite complex and not included here [6.1, pages 125-128].

For further discussion of the operation when the MEC is already loaded, refer to the dedicated software MEC presented later in Subsection 6.4.4.

Table 6.3. Types of MEC List and KERNEL Sublist Entries

PRIORITY	TITLE	K	M	N	TYPE
31	Power Failure		X		interrupt
30	Real Time Clock Failure		X		"
29	MEC failure (Special Case of 28)		X		"
28	Error		X		"
27	Loop Counter	X			"
26	PM Complete		X		"
25	External PM Enable		X		"
24	Mode Change			X	"
23	Real Time Clock	X			"
22	Channel Selector Switch Assignment	X			routine
21	BORAM Test			X	"
20	RAMM Test			X	"
19	Bus Test	X			"
18	Processor Test			X	"
17	Data Transfer	X			interrupt
16	Data Transfer Error	X			routine
15	Memory Address Error			X	"
14	PM Address Error			X	"
13	Data Transfer Request	X			"
12	PM Reinitialization		X		"
11	PM Assignment		X		"
10 thru 1	Miscellaneous Houskeeping			X	"

K - Kernel

M - MEC Resident

N - MEC Non-Resident

6.3.2.3 Reference to Flow Charts for Floating Software MEC Implementation

The interrupt and routine flow charts for the floating software MEC implementation are sufficiently different from those for the hardware MEC so that five of the flow charts are redrawn. One of them - the real time clock interrupt flow chart - now takes 6 pages. [6.1, pages 128-134, and 96-115].

6.3.3 Summary and Preliminary Evaluation of Floating Software MEC

The floating software approach has advantages over the hardware and dedicated software cases primarily in reliability, graceful degradation, and, of course, the fact that it does not require a processor on a full-time basis.

The disadvantages of the floating software approach are greater MEC complexity and slower running times for some functions, each processor must have any special capabilities required by MEC, and constraints on PM size and operation for those PMs designed to run in multi-program fashion together with the kernel. In some cases this constraint may be severe since the kernel is estimated to be about 800 instructions.

It is not necessary that all PMs allow space for the kernel, nor is this ever desirable. If six such PMs were running simultaneously on a large AADC configuration, then the five which were not sharing a processor with the kernel would collectively waste memory equivalent to an entire task memory. The designer responsible for developing a program module set for a given mode would thus be constrained to lay out his design in such a way that, on the average, at least one PM capable of sharing a processor with the kernel is running at any given time if he wants the MEC to be in its dormant phase at that time. A number of strategies could be employed to minimize unnecessary kernel relocation or other thrashing in the floating software MEC. Perhaps the best one would be

to choose as kernel co-resident PMs, those which have high importance criteria and fairly long running times.

The advantage of availability of more processors on the average, due to occasional dormant status of the MEC, is partly offset by the greater inefficiency of the MEC and of those PMs which are designed to run together with the kernel. If the density of interrupts becomes very high, this advantage would disappear altogether. The chief advantage of the floating software MEC is its ability to run on any processor and to switch freely between processors. If interrupt densities are high, then a dynamically relocatable "dedicated" software MEC would be preferable to the floating case as it is described in this section.

Table 6.4 gives an estimate of overhead (time spent in the floating software MEC master control operations). Most of the MEC functions would require the same time for either of the software approaches but the overhead times are higher in the floating case and significantly higher for the case that the MEC is required but not loaded. The overhead times in Table 6.4 could be used to determine processor time lost in overhead if a typical job stream was analyzed to determine its loading on the MEC in terms of types of requests and their frequency. The overhead times given in the figure assume that all elements of the floating software MEC are resident and, thus, are all loaded whenever the MEC is loaded.

Table 6.4. Floating Software Overhead Estimates in Microseconds
(From Figure 2-41, pages 1, 2, 3)

MEC Function	Average Time (μsec)	Overhead	
		MEC	Kernel
Power Failure	200	20	580
Real Time Clock Failure	6	4	---**
MEC Failure	590	4	9
Error	18	4	---
Loop Counter*	8	4	9
Program Module Complete	15	4	590
External PM Enable	61	4	603
Mode Change	450	4	585
Real Time Clock*	130	4-9	4-11
Channel Selector Switch Asst.*	9	4	---
BORAM Test	75	4	---
RAMM Test	17	4	---
Bus Test*	11	4	9
Processor Test	200	4	---
Data Transfer*	16	4	9
Data Transfer Error*	8	4	---
Memory Address Error	36	4	---
PM Address Error	120	4	---
Data Transfer Request*	16	4	9
PM Reinitialization	---	4	---
Program Module Assignment	120	4-113	---
Housekeeping	---	13	---
Interrupt List Empty (Idle Loop)	---	13	4

* Kernel Operations

** Dash in lieu of value indicates the value not applicable.

As shown in Appendix F [6.1], the amount of memory required to store the kernel is 899 words. All of the interrupts and routines require 2509 words plus an estimated 1100 words for ID and other data storage, for a total MEC program of 3609 words. This is within the requirements of a 4K word task memory.

The time required to execute all of the interrupts and routines in the nominal case is 4.43 msec. This includes a total of 2.51 msec in overhead time. Overhead times were taken from Table 6.4 assuming that the following interrupts occur when only the kernel is loaded.

- Power Failure
- PM Complete
- External PM Enable
- Mode Change

This total is 2.75 times as large as that required by the Dedicated Software MEC. This is due primarily to the overhead involved in writing in the MEC when only the kernel is active. It is also 11 times longer than that for the hardware MEC.

6.4 DEDICATED SOFTWARE MEC FOR DUAL PROCESSOR

Unlike the case of the dedicated software executive for the Baseline and MMM AADC systems, the dedicated software executive for the dual processor system could be considered a hardware implementation. This is because a second processor, identical to the simplex processor, must be added to the system. Thus, one processor of this system will be dedicated to processing the executive program while the other processor executes program modules. Actually, Honeywell calls this system an Optimized Simplex system with a dedicated software MEC, but a dual processor system is more accurate.

6.4.1 Dual Processor System

Figure 6.9 is a block diagram of the dual processor for the dedicated software executive. With only one processor processing program modules, the added expense of an associative memory does not seem justifiable in lieu of the expected nominal savings in time. The dedicated MEC processor can run in parallel with the PM handling processor for the majority of executive functions.

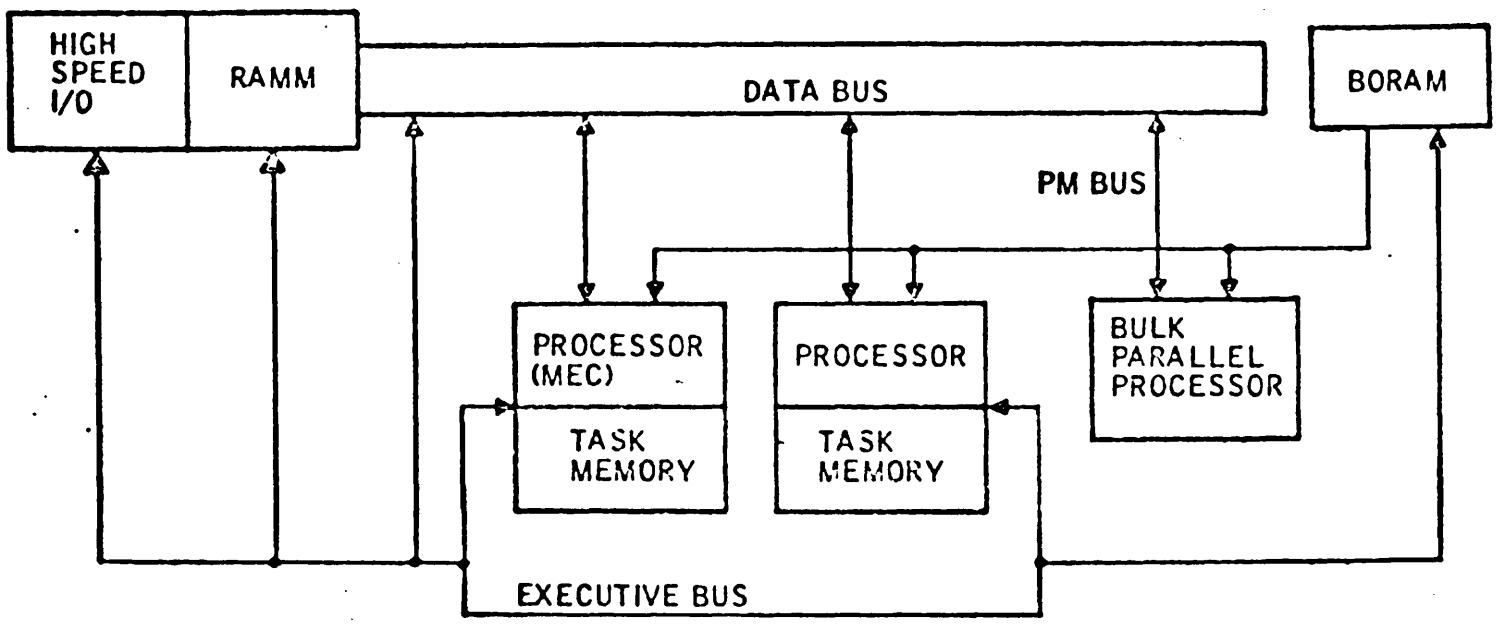


Figure 6.9. Dedicated Software Dual Processor AADC System

The BORAM, RAMM and Task Memories are all the same as for the Floating Software MEC in Subsection 6.3.1. The two PEs are identical except one is reserved for the sole use by the MEC. It must contain suitable microprogramming to handle the same associative memory functions (although no associative memory is available) as appear in the hardware MECs of the Baseline system. Thus, the PE must be enhanced to include some hardware MEC features. In actual fact, the other PE must also be able to handle all the MEC functions in case the MEC PE fails.*

The Matrix Parallel Processor and the high-speed multiplexed digital interface will be the same in the previous systems, but the dedicated I/O units and channel selector switch are not necessary in this simplified configuration.

6.4.2 System Bussing for Dual Processor System

Three distinct busses are used to transmit Program Modules, data and control signals throughout the system. They are the PM Transfer Bus, the Data Bus and the Executive Bus. The PM Transfer Bus is used to transfer PMs from BORAM to the simplex processor and MEC segments to the MEC processor. The Data Bus is used for data transfer between RAMM and the PEs. It is not required to be dual width as in the previous systems because there is only one PE executing PMs. The Executive Bus provides communication and controls between the MEC and all system resources as described previously. For further details see Section 6.2.2.

6.4.3 Operation of the Dual Processor System with Dedicated Software MEC

The operation of the dual processor system with a dedicated software MEC is the same as the Baseline system with a hardware MEC except for the following three simplifications:

*This is a very important point that has not been emphasized sufficiently in the design [6.1].

1. There is no need to consider the two types of PM requiring "special processing" because with only one PE it is inappropriate to consider a PM that overflows into another task memory or two PEs working on the same PM.

2. It is not possible to dedicate a PE to a given PM. In fact, a single PE must process several PMs at a given rate and in proper sequence in order to operate properly.

3. All PMID words are stored in BORAM and the active PMID words plus all other resource words are stored in the Task Memory of the MEC processor.

The MEC functions to be performed are the same as described previously in Section 6.2.4. One exception in the operation is that PMs will normally be allowed to continue executing until completion unless the postponing of handling of an interrupt endangers the mission.

6.4.4 Summary Flow Chart of MEC for Dual Processor

Logically, the summary flow chart shown in Figure 6.5 of Subsection 6.2.7.1 is satisfactory for use in this section. All of the processing requirements of the MEC are implied in the summary flow chart and will be shown to be satisfied as the result of processing the interrupts and routines from the list.

Normal MEC processing consists of interrogating the LIST for information that will direct the MEC to execute particular routines such as shown in Table 6.1. The channel selector switch assignment routine is not needed.

One method of implementing software-wise the list processing logic shown in the summary flow chart is to employ a system of processor flip-flops. Thirty-two hardware flip-flops are needed that can be set, cleared, and tested. These flip-flops will be associated with the MEC routines listed in Table 6.1

in such a manner that a set flip-flop is equivalent to having its associated routine "on the MEC LIST" and a cleared flip-flop equivalent to the routine's "absence from the MEC LIST".

The MEC can detect the necessity of executing a routine by testing its flip-flop. The testing can be done by 32 consecutive conditional jump-type instructions which will branch to the appropriate routine if its flip-flop is set. The first test instruction has the label START and tests for the presence of the highest priority rated routine. If the test fails, the next test instruction checks for the next lower priority rated routine, etc. When the housekeeping routines are reached, the MEC condition is that of an idle state, executing housekeeping routines until an interrupt occurs which places a routine of higher priority on the LIST.

Each housekeeping routine, when run, removes itself from the LIST, thus assuring that all such routines get run in sequence. The lowest priority routine must replace all other housekeeping routines on the LIST so the cycle can be repeated. When a test is successful and a routine is given control, the routine should clear its flip-flop before giving control back to START.

This implementation suggests the desirability of bit processing capability for the software. If this capability were present, flip-flops would not be necessary as a memory word or words could be used. In this case, the ability to set, clear, and test any bit in a word would be required. This has been included in the PE design and offers an alternate design to the 32 hardware flip-flops.

That part of the summary flow chart which discusses class levels would be implemented with actual linked lists in the form of queues. Each entry is a

queue would contain the necessary parameters for the routine associated with that queue. Each routine that required parameters would have a dedicated queue. Thus, each time a routine is placed on the LIST, its necessary parameters would be placed as an entry on the proper queue in a first-in first-out manner. When the routine is executed, it takes the top entry from its queue and processes it. (The bottom entry is the most recent entry.)

6.4.4.1 Internal and External Interrupts

Every interrupt occuring on the executive bus and interrupting a resource is denoted as an external interrupt if the originating resource is not the same as the destination resource. All other interrupts of a resource are denoted as internal.

Each processor contains a real time clock that can be reset by the program in the processor. When the clock interrupts the processor (not over the executive bus), control is routed to a fixed location in the task memory and the real time clock interrupt processing routine located there is executed. Of course, if the clock is not set, no interrupt will occur and no interrupt routine is necessary. The executive always sets its real time clock.

Other internal interrupts will occur on the executive bus for the MEC processor only. This means that non-MEC processor contain only one internal interrupt - its real time clock interrupt. However, the kinds of internal interrups that the MEC processor has, also exist in the other processor - i.e., parity, power failure, clock failure, etc. - but they will interrupt the MEC processor and not the processor in which they occured. Take, for example, a parity error. If a parity error occurs in the MEC processor, the hardware

generates an interrupt over the executive bus taking as the originating resource the MEC processor and, as the destination resource, also the MEC processor. (The first code is the processor ID code and the second is the MEC processor ID code - in this case, they are the same code.) If the parity error occurs in a non-MEC processor, the two codes will be different. Hence, a parity error interrupt will always interrupt the MEC processor, but will be considered an internal or external interrupt if the originating resource was the MEC processor or not.

Actually, it is immaterial whether interrupts are internal or external (except for the real time clock) because all executive bus interrupts to a particular processor are handled in the same way. When a processor (MEC or not) is in an interruptable state, the active line is set, and the processor's ID code matches the destination code on the executive bus, the processor is interrupted with control going to a predetermined location.

Therefore, each processor must have two locations reserved in the task memory for interrupt handling. These locations will be the same for all task memories [6.1, pages 199, 200, 93-96].

6.4.5 Summary and Preliminary Evaluation of the Dedicated Software MEC on Dual Processor System

This implementation offers total use of a single processor for running PMs, through the use of a second processor dedicated to MEC functions. A small degree of parallelism is also gained. Compatibility with the previous systems is easily maintained with practically no additional software cost. A floating software MEC must also be provided in case one PE fails. The total amount of memory required is 3373 words. This includes 1091 words of ID and miscellaneous

data and 2282 words for interrupt and routines as shown in Appendix F of [6.1]. The time required to process all the interrupts and routines in 1.253 msec.

In this implementation, one processor is always working on system tasks and its throughput should be maximum since it has an entire MEC dedication to assist it.

6.5 FLOATING SOFTWARE MEC FOR OPTIMIZED SIMPLEX PROCESSOR

The Optimized Simplex system is the same as the dual processor shown in Figure 6.9, except there is only one processor and the processor is allowed to execute programs from the RAMM as well as the Task Memory. In this way the Floating Software MEC is able to perform its functions without overwriting the currently executing PM. Although the Matrix Parallel Processor is included in this diagram, it is actually optional and may be deleted without affecting the MEC operation.

The bussing system in this case is the same as for the dual processor system (Section 6.4.2).

6.5.1 Operation of the Optimized Simplex Processor with a Floating Software MEC

The operation of the Optimized Simplex Processor with a Floating Executive Control is similar to the operation of the Dual Processor with a Dedicated Software MEC. The major difference is the RAMM must always contain the MEC kernel which is ready to handle interrupts. The MEC is normally in its dormant state and is reached via a kernel for urgent services and normally reactivated by a PM complete interrupt. When this interrupt occurs, the MEC will employ the now available processor to process its task LIST and then initiate that the waiting PM with the lowest assignment deadline. Since the kernel can execute instructions directly out of RAMM (where the MEC is stored) there is no need for the PM in the processor to be aborted. Time must be kept track of while the processor is executing the MEC to avoid a PM time out. In this configuration, the kernel consists of same interrupts and routines as in the Baseline case except that the channel selector switch assignment is not required. 780 words make up the kernel.

Since the PE can now execute MEC segments from RAMM and PMs from the Task Memory, an interrupt can either be handle when it occures or be listed for later execution, depending on the urgency. There are also other small difference in the operation of the dedicated software and floating software executives, but these do not seem significant [6.1, Section 4.2.3].

6.5.2 Summary Flow Chart of Floating Software MEC for the Optimized Simplex Processor

The summary flow chart for the floating software MEC for the optimized simplex system is shown in Figure 6.10. Although the logic is essentially that of the previous floating software cases, it is much simpler because the resource allocation problem is less complex. The three basic alternatives of MEC operation are: (a) handling an (external) interrupt either by executing the appropriate routine (power failure) or listing it for later processing; (b) handling a MEC call, internal interrupt or "pseudo-interrupt" (in this case, from the current PM either by executing a small kernel routine in task memory, a short MEC routine in RAMM, or by loading a MEC routine in task memory and processing the call); and (c) the case MEC activity is due neither to interrupt nor call, i.e., the MEC is fully active and is processing its task LIST. The latter alternative ends when a PM assignment operation is encountered, an external PM enable occurs, or a mode change is called for. If none of these occur, then the MEC begins processing miscellaneous houskeeping routines until external circumstances call for it to begin a new mode or initiate a PM.

The priority scheme, given in Table 6.3, applies to this case except that only the kernel is resident; all other routines and handlers are either segmented or executed out of RAMM.

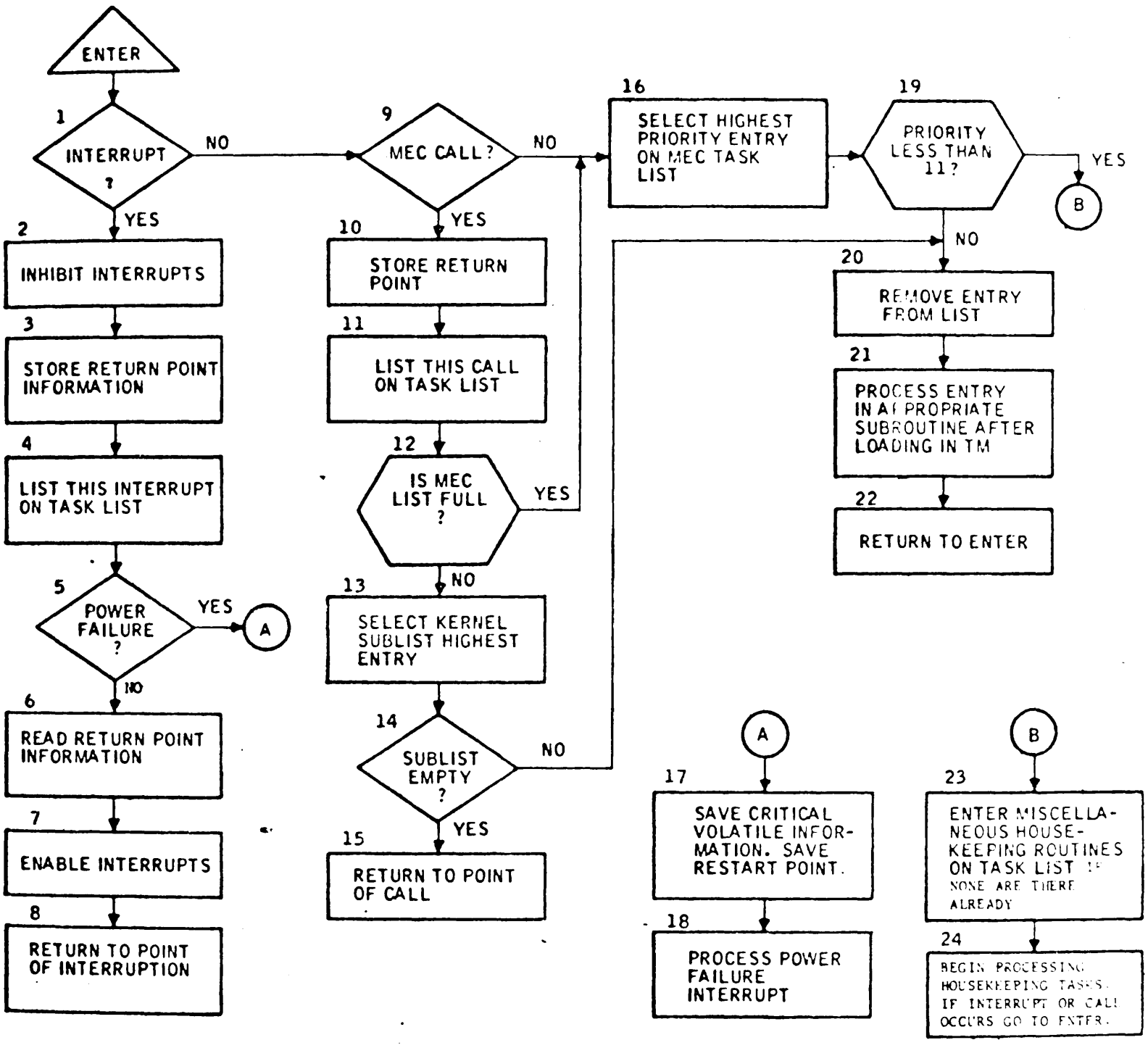


Figure 6.10. Summary Flow Chart of Floating Software MEC for Optimized Simplex System

6.5.3 Summary and Preliminary Evaluation of the Floating Software MEC on the Simplex System

The floating software approach on a simplex system may be considered as a minimal cost, low performance configuration or as a fall-back configuration reached when all processors but one of a more complex system fail. In the simplex configuration, the floating software approach for a MEC does not offer the enhanced reliability indicated in Section 6.3.3.

As compared to the software dedicated processor case in Section 6.4, the floating software requires one less processor since the MEC shares the unit processor with the PM currently in execution. It has the advantage of requiring one less processor and, naturally, the disadvantage of taking time away from the currently operational PM on the only processor available. This approach is, thus, advantageous only if the sum total of MEC functions plus overhead required considerably less than half the processing time of one processor. In this case, the cost/effectiveness of the system may be competitive even though the throughput is almost halved. If the total time needed for MEC functions required less than ten percent of a processor's attention, then the floating software approach would be advantageous for the optimized simplex processor system. Also, the simplex floating MEC must utilize part of the task memory to house the MEC kernel. Thus, all PMs must be 780 words shorter than in the dedicated software case.

Appendix F [6.1] shows the estimated memory requirements and execution times required by this MEC implementation. The interrupts and routines require 1877 words of memory. An additional 1100 words are used for ID words and other data. This yields a total memory requirement of 2977 words.

The time to process the entire set of interrupts and routines in the nominal case is 1.037 msec. The following functions will require a complete load of the MEC.

- Power Failure
- Error Interrupt
- External PM Enable
- Mode Change
- PM Complete

The estimated time it takes to load the MEC is assumed to be 580 μsec, thus, 2.9 msec must be added. This makes the total time 3.937 msec. This exceeds the dedicated implementation by a factor of 3.1.

6.6 EVALUATION AND RECOMMENDATIONS

6.6.1 Method of Evaluation

Three steps will be taken to evaluate the MEC implementations for each of the system configurations considered in this study. First, a set of evaluation parameters (system attributes) will be established. Then weights will be assigned to each attribute as a function of its importance. Then a table will be constructed for each of the four systems and each implementation will be a measure of its effectiveness for the system under consideration.

Each attribute will be assigned a weight of 10 or less. For each system, each MEC implementation will be evaluated against the attributes. Scoring will be on a ten (10) point must system with the MEC implementation that best exhibits the attribute receiving 10 points and others a proportionate amount. The points will be multiplied by the attribute's weight and the products summed for all attribute-point products. The implementation with the best score (highest) will be recommended. This will be done for each AADC configuration and will result in a "best" choice for each system. Based upon the results for each AADC configuration, an "optimal" implementation for all configurations will be recommended.

6.6.2 Evaluating the MEC Implementations

The attributes that were selected are: reliability, graceful degradation, speed of the MEC, constraints on the rest of the system, functional expandability of MEC (can it be enhanced without redesigning?), maintainability, hardware production cost, software production cost, volume, weight, power requirements, hardware and software developmental costs, flexibility (ability to perform other functions), simplicity, overhead, and computational suitability

(how well the MEC is optimized to perform its functions). Although the names of the attributes are fairly suggestive of their functions, further description of each can be obtained from [6.1, pp 226-229].

Table 6.5 summarizes some of the quantitive MEC performance attributes that are used in the comparative evaluation. Something seems suspicious in the time to process interrupts and routines for the floating software MECs; for Baseline system overhead in 2.51 msec, for MMM system overhead is 0.17 msec and for OS system the overhead is 2.9 msec.(The interrupt/routine processing times are 1.93, 1.94 and 1.04 respectively, which is reasonable.)

The ranking of the attributes is reflected in the ordering above with the most important attributes listed first. The exact weights assigned to each attribute is shown in parentheses after the attribute name in the tables following Table 6.5.

Table 6.5 Summary of Quantitative MEC Performance Attributes

Baseline System

MEC	COST	TIME TO PROCESS ALL INTERRUPTS/ ROUTINES (MILLISEC)	COMPLEXITY	BACKUP
Hardware	$6100 (=.65 x PE)	0.39	4000 logic gates 360 word AM, 5200 Word ROM	Software MEC
Dedicated Software	$9400	1.6	A special assigned PE, 3900 word TM	Another PE
Floating Software	$2000 (assuming 20% for MEC	4.44 (including 2.51 msec over-head)	899 word Kernel, PE part time (20%) with 3600 word TM, Some PMs restricted by 899 words	Built-in

Multiple Memory Multiprocessor

Hardware	$6100	0.55	Same as Hardware MEC above	Software MEC
Dedicated Software	$9400	1.85	A special assigned PE, 4025 word TM	Another PE
Floating Software	$7400 (.75 x PE)	2.11 (including 0.17 for capturing PE)	PE part time (20%) with 3450 word TM, PM not restricted	Built-in

Optimized Simplex

Dedicated Software	$9400	1.25	Dedicated PE With 3370 word TM	Floating Software
Floating Software	$2000 (assuming 20% for MEC)	3.9 (including 2.9 for loading MEC	PE part time with 2980 word TM all PM restricted by 780 words.	None

The scoring of the attributes for the AADC Baseline system and the total score is shown in Table 6.6. As shown the hardware MEC scores highest. The best backup for the hardware MEC is the Floating Software MEC.

The scoring for the Time Division Multiplexed Block Transfer Multiprocessor System is the same as for the Baseline System without the hardware MEC. Thus the Floating Software MEC is best for the TDM Block Transfer Multiprocessor.

The scoring of attributes for the MMM System is shown in Table 6.7. The hardware MEC again scores the best and even higher than for the Baseline System. The two software MECs score about the same for the MMM System.

The scoring of attributes for the Optimized Simplex System is shown in Figure 6.8. The Dedicated Software MEC scores the best for the OS System (Again this actually a violation of the simple processor Optimized Simplex concept.) Also see Subsection 6.6.4 for comment on these evaluations.

6.6.3 Recommend MEC Implementation Methods

As indicated in the evaluation, the best MEC implementation for each system is a function of the PM load of the system, the number of executive functions required per PM, the average run time of a PM, and the number of resources available in the system. These are all parameters which are not as yet well defined, and will probably not be before extensive simulation is complete.

Bearing this in mind, the following MEC implementations are recommended.

Table 6.6. Baseline AADC System Evaluation

		IMPLEMENTATION	
Attribute	Hardware	Software Dedicated Processor	Software Floating
Reliability (10)	10	6	8
Graceful Degradation(10)	6	6	10
Speed (10)	10	3	1
Constraints on rest of system (10)	10	8	4
Functional Expandability(10)	6	10	8
Maintainability(9)	6	10	8
Hardware Cost(8)	6	2	10
Software Cost(8)	10	8	6
Volume (5)	6	2	10
Weight (5)	6	2	10
Power (5)	6	2	10
Development Cost (4)	7	10	8
Flexibility (3)	10	7	7
Simplicity (2)	10	7	5
Overhead (2)	10	8	1
Computational Suitability (2)	10	4	4
Weighted Total 8	10	629	733

Table 6.7. Multiple Memory Multiprocessor Evaluation System

	IMPLEMENTATION		
Attribute	Hardware	Software Dedicated Processor	Software Floating
Reliability (10)	10	7	8
Graceful Degradation (10)	6	7	10
Speed (10)	10	3	2.5
Constraints on rest of System (10)	10	8	2
Functional Expandability (10)	6	10	9
Maintainability (9)	6	10	9
Hardware Cost (8)	10	6	9
Software Cost (8)	10	8	8
Volume (5)	10	6	9
Weight (5)	10	6	9
Power (5)	10	6	9
Development Cost (4)	7	10	9
Flexibility (3)	10	7	7
Simplicity (2)	10	7	6
Overhead(2)	10	8	6
Computational Suitability (2)	10	4	4
Weighted Total	902	741	756

Table 6.8. Optimized Simplex Processor Evaluation

	IMPLEMENTATION	
ATTRIBUTE	SOFTWARE Dedicated Processor	Software Floating
Reliability (10)	10	5
Graceful Degradation (10)	10	5
Speed (10)	10	3
Constraints on rest of System (10)	10	4
Functional Expandability (10)	10	7
Maintainability (9)	10	8
Hardware Cost (8)	2	10
Software Cost (8)	10	7
Volume (5)	2	10
Weight (5)	2	10
Power (5)	2	10
Development Cost (4)	10	8
Flexibility (3)	10	9
Simplicity (2)	10	7
Overhead (2)	10	6
Computational Suitability (2)	10	10
Weighted Total	852	603

SYSTEM	Recommended MEC Implementations
Baseline AADC System	Special Purpose Hardware
TDM Block Transfer System	Floating Software
Multiple Memory Multiprocessor	Special Purpose Hardware
Optimized Simplex Processor	Dedicated Software

These recommendations may change if the weighting factors assigned to the attributes are deemed to be inappropriate for the ultimate utilization of the AADC system.

In the baseline system, the hardware MEC wins primarily because of basic reliability, speed, lack of system constraints, and the fact it is designed specifically to handle the executive tasks.

In the TDM system the floating software MEC wins primarily due to graceful degradation, cost and the associated attributes of size, weight and power.

The MMM system operates best with a special prupose hardware MEC because of speed, lack of constraints on PMs, cost, size, weight, power, and because it is designed to perform executive tasks.

In the optimized simplex syste, the dedicated software gets the nod due to every attribute except cost, size, weight, and power.

6.6.4 Author's Comments on the Evaluations

There is one very obvious and very serious omission from the list of attributes which would probably change the results significantly; that is, the cost of providing adequate backup. For the hardware MEC the extra backup could be built into an ultra-reliable MEC but more likely would be a software

MEC. Thus the development and production cost of providing this software backup should be included in the cost for the hardware MEC. The cost of providing triple redundancy in the hardware MEC would probably eliminate it from contention.

Another obvious case where the results would probably be different is for the Optimized Simplex System. Since the only backup for a Dedicated Software MEC is a Floating Software MEC (if one PE fails), the cost of producing two sets of software is certainly larger than for producing only one. Thus the best system for the dual processor is probally the floating software.

From examining Honeywell's report and without doing any analysis, it would seem that the Dedicated Software MEC could be eliminated for considering if the need for adequate backup was included. It seems unbelievable that a report as comprehensive and as detailed as this could have skipped such an important avionic requirement as adequate backup. On the other hand, the inclusion of an adequate backup may have reduced the number of viable MEC alternates to only one - the floating software MEC, thereby reducing the complexity of the project significantly.

The statements in this subsection are the opinions of the author's and not that of the Navy or the NPS, and are not substainated by fact.

6.7 RECOMMENDED AREAS FOR FURTHER STUDY

6.7.1 Continued Development, Simulation and Implementation of MEC

Figure 6.11 shows eight studies recommended by Honeywell for design and implementation of the MEC and the appropriate time periods. The studies include:

1. The initial MEC implementation study as defined in this report and [6.1].

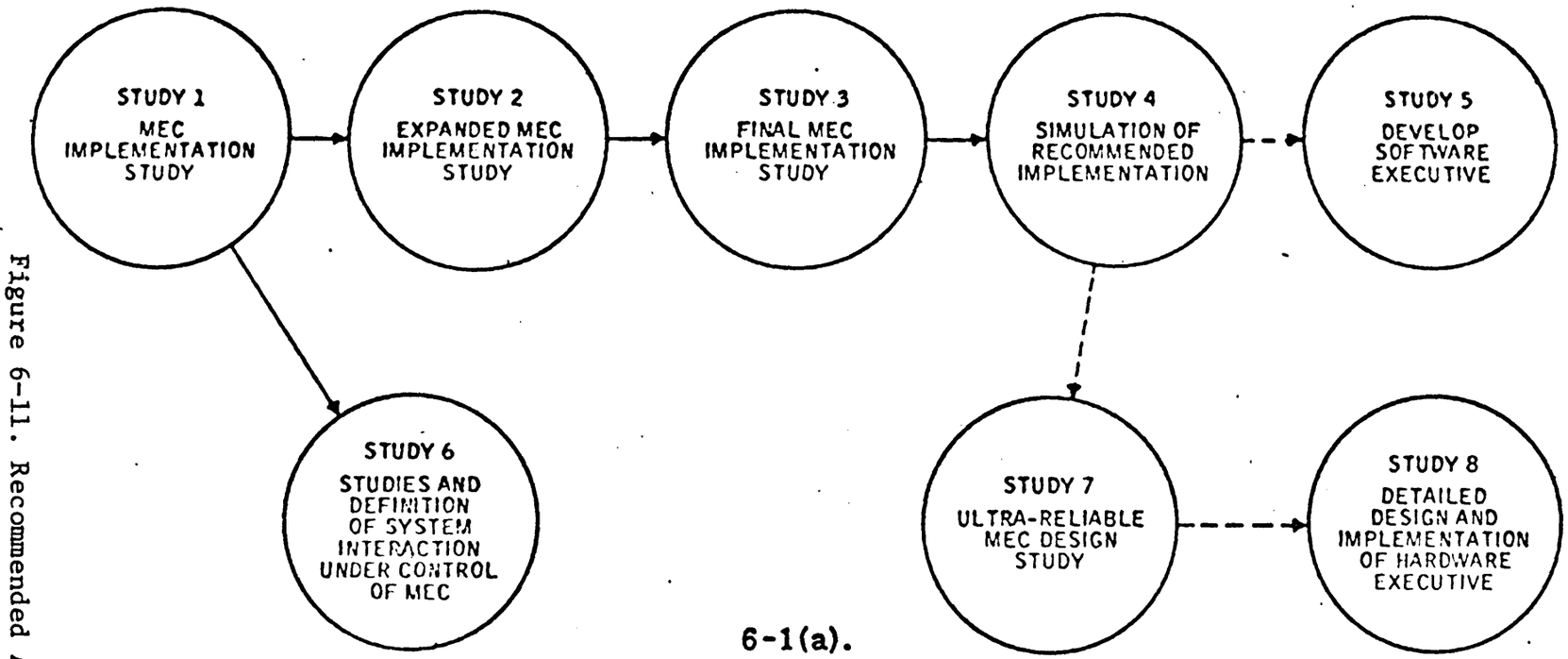
2. An expanded MEC implementation study to include two other executives system called the Dynamic Dedicated Software and the Dynamic Dedicated Software with Associative Memory. This would be equivalent effort to doing two of the three studies in [6.1].

3. A final MEC implementation study including an overview simulation of all the components in the AADC and their interactions (this would be a suitable thesis topic), and suitable expansion correction and detailing of English language flow charts.

4. Simulation of the recommended MEC implementations. This would be a detailed real-time simulation of all the MEC functions and the interaction with other AADC components.

5. Based on the results of Study 4, either a software or hardware (or both) MEC will be selected for implementation. If a software MEC is chosen, Study 5 would result in a coded and hopefully debugged software executive program capable of running on the Navy's AADC prototype system.

6. The sixth study would be the definition of system interaction under MEC control - to be run in parallel with the studies above. The areas for study are bussing techniques, digital interface designs, channel selector switch designs, and alternate routing in the case of component failures.



6-1(a).

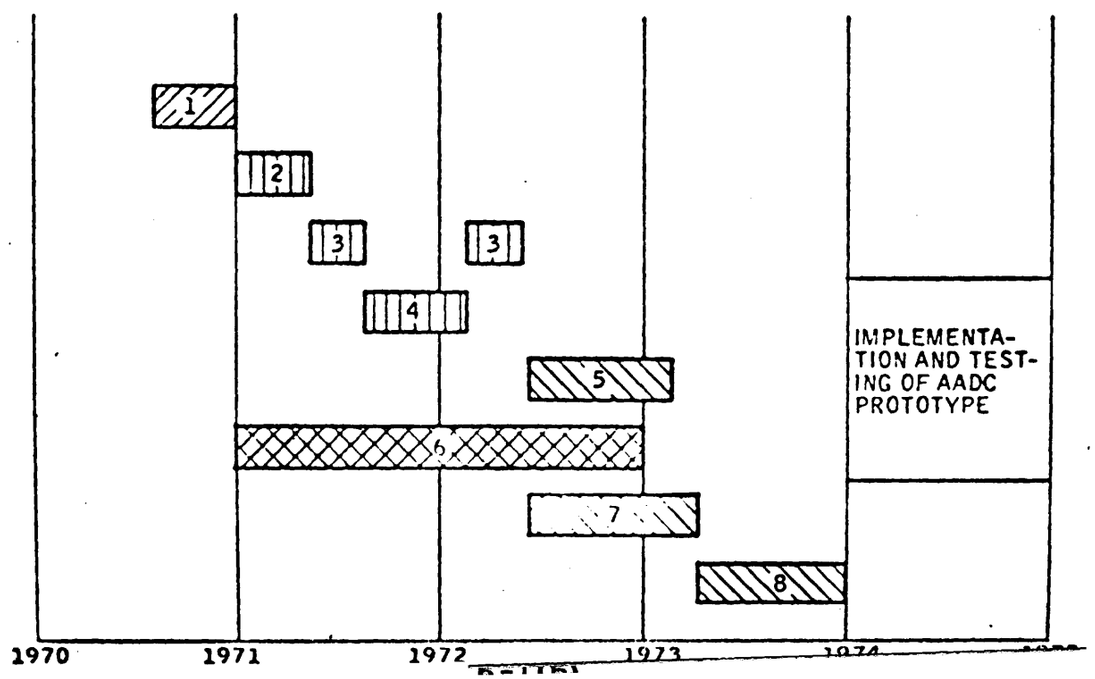


Figure 6-11. Recommended Areas for Further Study

7. An ultra-reliable hardware MEC design study is needed if only a hardware MEC is provided. The study would include locating critical portions of MEC, defining failure detection methods, choosing redundancy and error correction techniques and selecting a fail-safe and fail-soft design. This work is already in progress [6.6]. (This would make a good thesis topic).

8. Detailed design of a hardware MEC including the design of the AM, ROM, algorithms, logic and control and MEC language.

There is an ongoing effort to improve the reliability and applicability of the AADC Master Executive Control (MEC). This work will permit the executive to reside in various versions of AADC configured to improve overall computer system reliability and problem solution confidence, as well as improve MEC response in the event of hardware failures. The improved MEC will also incorporate provisions for demand paging of both procedure and data, event posting and process scheduling, as well as more efficient distribution of function [6.7].

In addition to the above, there is also an effort to develop a rudimentary OS/AADC which can be used to interface AADC with non-avionic peripherals. This is in keeping with the expanded role of AADC.

6.7.2 Continued Development of MEC Course Material.

Although this chapter has been shortened considerable over that in Reference 6.1, it is considered only a first draft and could still be shortened considerably. This section describes some of the ways in which this chapter on the MEC can be improved in the next version.

For teaching purposes, I think Sections 6.3 and 6.4 (the backup floating software MEC for the Baseline System and the dedicated software MEC for the dual processor system) should be eliminated. Also the floating software MEC for the

simplex system should be described first as the simplier system, before the more elaborate Hardware MEC. Also the descriptive verbage should be reduced and made more concise.

These reductions in the design philosophy portions would allow the inclusion of more detail on the actual design implementation. In particular, more of the interrupt and MEC routines could be included, with English language flow charts. These could be organized as follows:

1. MEC interrupt/routines common to both the floating software and hardware MEC,

2. MEC interrupt/routines unique to the floating software MEC on a OS system,

3. MEC interrupt/routines unique to the hardware MEC on an AADC baseline system.

In summary, this chapter presents the design philosophy, various hardware and software configurations, design tradeoffs, capabilities and operating characteristics of the major control component of the AADC system - the Master Executive Control.

6.7.3 Current Status of MEC Developments.

Since this chapter was written, Honeywell has produced another voluminous report containing four volumes. Volume I contains a summary of the technical results of the report. Volume II is the technical volume and contains the results and tradeoffs of the demand paging and virtual memory performance for AADC, the functional analysis of MEC, the internal communication and bussing system and the functional description of the hardware, software and hybird executives for AADC. Volume III contains all support data and information while Volume IV contains the detailed timing and evaluations of the three different implementations of MEC.

Of this report, Sections 3 and 5 of Volume II, on the descriptions of the functional analysis and functional descriptions of the hardware, software and hybird MECs, respectively, are of the most interest here. (Section 2 of Volume II on demand paging and virtual memory pertains to Chapter 3 while Section 4 of Volume II on internal bussing is discussed in Chapter 2).

The three versions of MEC considered in this report are the dedicated hardware MEC (similar to the one in Section 6.2 above), the floating software MEC (similar to the one in Section 6.3) and a hybird executive which consists of a software executive with an associate memory assist. The flow charts of these executives are shown in Volume II, Section 5 [6.7].

Apparently in Volume V of Honeywell's report, an Optimized Simplex MEC is defined which uses fixed priority scheduling rather than time-driven scheduling, has a despatcher and an interrupt handler, allows pre-emptive scheduling by higher priority tasks and contains only seven modules and three system tables instead of the 17 modules for the other MECs [7.8 and 7.9].

Question on the MEC

For problems concerning the MEC development see Section 6.7. Specific questions will be included in the next report.

6.1 In Section 6.3.3, what are some of the problems of trying to defind a set of kernel co-residents Program Modules? Can you guarantee one of these PMs is always present?

6.2 Use Table 6.4 to estimate the overhead for a particular set of PMs in a particular mode. Try to obtain realistic usage data for a particular aircraft.

6.3 See Section 6.7, particularily items 3 and 7, for a group term project or thesis topic.

References for Master Executive Control (MEC)

6.1 AADC Master Executive Control, System Analysis Design Study; Final Report; Honeywell Inc., Report No. 12234-FR; December 1970; NAVAIRDEVCEN Contract N62269-70-C-0314; Unclassified-NOFORN; AD-800-635 (Vol.1, Basic Document); AD-800-637 (Vol. 2, Appendices); (43, Vol. 1 only at NPS).*

6.2 Master Executive Control Techniques for AADC System Final Report; Honeywell Inc., No. 14206-FR; July 1969; NAVAIRSYSCOM Contract AIR-5333-69-1; Unclassified; (10).

6.3 AADC Master Executive Control, Baseline Definition; R. S. Entner, NAVAIRSYSCOM and J. Stepenosky NAVAIRDEVCEN; 22 December 1969; Unclassified-NOFORN; (19).

6.4 An Associative Memory for Executive Control Functions in an Advanced Avionics Computer System; R. Berg and M. Johnson; Proceedings of the 1970 IEEE Internation Computer Group Conference; June 16-18, 1970; pp. 336-342; Unclassified; (31).

6.5 A Hardware Executive Control for the Advanced Avionic Digital Computer System; R. O. Berg and K. L. Thurber: NAECON '71 Record; May 1971; pp 206-213; Published by the IEEE Transactions on Aerospace and Electronic Systems; Available on special order only from the IEEE Order Department, 345 East 47th Street, New York, N. Y. 10017; Reference publication 71-C-24 AES; (54, NPS).

6.6 Operating System Reliability for the Navy AADC: R. S. Entner and E. H. Bersoff; IEEE Transactions on Aerospace and Electronic Systems; January 1971; pp 67-72; (47, NPS).

* AADC Bibliography number and available at the Naval Postgraduate School.

6.7 Master Executive Control for AADC - Interim Report; Honeywell, Inc., Report Z9506-3018; NAVAIRDEVCEN Contract No. 62269-72-C-0051; June 1972; Volume I to IV; (77, NPS).

6.8 Master Executive Control for AADC - Final Report; Honeywell Inc., Report Z9506-3018; NAVAIRDEVCEN Contract No. 62269-72-C-0051; Oct. 1972; (NPS).

6.9 All Applications Digital Computer 1973 Symposium; Orlando, Florida; Jan. 23-25, 1973; Proceedings are not yet available.

Chapter 7

SIGNAL PROCESSING ELEMENT

Table of Contents for the Signal Processing Element

List of Figures

Glossory of Terms for Parallel Processor

AP	- Associative Processor: first version of the AADC Parallel Processor
APE	- Array Processor Elements: a general purpose sequential processor with limited control and data management capabilities. One of many processors in the GPAP.
A&C	- Arithmetic and Control Unit of DPE. A&C plus Task Memory makes a sequential Data Processor Element.
BPP	- Bulk Parallel Processor: another name for a general parallel processor.
DPE	- Data Processing Element for sequential Processing (Chapter 5)
GPAP	- General Purpose Array Processor: the third version of the AADC Parallel Processor.
ILLIAC IV	- A very large matrix parallel processor with 64 processor elements in parallel under one instruction interpretor (controller) installed at NASA Ames in San Jose, California.
MCU	- Microprogrammed Control Unit: the main control unit for SPE.
MPP	- Matrix Parallel Processor: second version of the parallel processor.
PEPE	- Parallel Element Processing Ensemble: a special parallel processor with several identical PEs each one for tracking its own radar target under a single pair of control units [7.4].
PMU	- Program Management Unit for Data Processing Element - function similar to MCU, (Chapter 5).
SCU	- Storage Control Unit: control for the SPE Buffer Memories.
SPAU	- Signal Processor Arithmetic Unit: the arithmetic, logic and shift unit for SPE.
SPE	- Signal Processing Element: the latest version of the AADC Parallel Processor.

Chapter 7

PARALLEL PROCESSOR

7.1 INTRODUCTION AND SUMMARY

Whereas the PE described in Chapter 5 is designed to fulfill all the sequential processing requirements, the parallel processor is designed to handle all the parallel processing requirements for AADC. The avionic parallel processing requirements include signal processing, radar processing, multiple tracking, pattern recognition, table look-up, optimal filtering signal correlation, Fourier analysis and synthesis, analog test function generation, voice command interface, etc. Parallel processing requirements are for 70 to 133 MIPS and 32K to 100K words of memory [7.1].

Although the parallel processor was one of the first AADC areas of concern, it has undergone more changes in design concept than any other AADC module, it still is the module whose design is the least firm and the most likely to be changed. Already the parallel processor has been referred to as the Bulk Parallel Processor (BPP), Matrix Parallel Processor (MPP), Associative Processor (AP), General Purpose Array Processor (GPAP), and the Signal Processing Element (SPE).

The feasibility of constructing a parallel processor capable of 150 MIPS throughput is not in doubt, but what will it cost, and how should it be designed to maximize the throughput, maximize the flexibility and minimize the cost? ILLIAC IV and PEPE are examples of very powerful parallel processors that are already in operation at NASA, Ames, San Jose,

California and the Ballistic Missile Defence Agency, Huntsville, Alabama, respectively. The ILLIAC IV interprets instructions sequentially, controls 64 arithmetic processors in parallel, has inter processor communications and is specifically designed for array processing. The PEPE is similar to ILLIAC IV except it interprets 2 instruction steams simultaneously, contains any number of arithmetic processors, has no interprocessor communications and is specifically designed for radar-like signal processing. Both these systems are very complex and costly [7.2-7.4]. The SPE is intended to perform both these functions but with a cost reduction of at least two orders of magnetude.

7.2 HISTORICAL DEVELOPMENTS

7.2.1 Associative Processor

One of the first concepts promulgated by AADC has been the incorporation of an optional, integrated, array processing capability within the computer mainframe. This hardware function would permit general purpose processing of radar, acoustic and/or video signals by means of some combination of domain transformation (frequency to time domain or vice versa), convolution (a method of correlating two signals) and high speed, associative list processing (searching on an attribute in a file rather than on an index*).

It was initially believed that a "simple" Associative Processor (AP) would suffice to handle all bodies of data which are amenable to bulk processing. It was believed that the AP could be used to maintain a multiple target track file, perform various filter operations or provide a means to correlate target signature information [7.5, 7.6 and 7.7]. Shortly thereafter it was realized that certain tasks, primarily those which require domain transformations, could not be adequately processed in a simple associative processor. It was also recognized that the cost of the AP could be prohibitive if storage requirements grew beyond moderation.

Some more recent work at NRL on a new associative processor, including a simulation, is presented in [7.8 and 7.9].

7.2.2 Matrix Parallel Processor

To contend with these problems, two further elements were added to this subsystem within a subsystem (the Associative Processor within the Parallel Processor). The new elements, the Fast Fourier Processor and the Pseudo-Associative Memory were conceived as individual, extensible building blocks which could be fitted together with the Associative Processor and then put under the supervision of either a private operating system or the AADC MEC* *.

---

*like finding the name of the person from the address in the phone book.
**Master Executive Control.

The AP, however, underwent certain conceptual changes based on its modified role, as well as a better appreciation of its operating environment. Among these changes were:

- the inclusion of a full adder in each memory cell;
- independent, simultaneous, multifield operations;
- vertical, as well as horizontal neighbor communications and control; and,
- a variable clock, which would keep system operation in step with variable settling times.

Two fundamental problems still existed in the new Matrix-Parallel Processor (MPP), as the combination of elements was called. First, there are domain transformations other than the Fast Fourier Transform (FFT) which are useful, and in some cases superior, for airborne data processing applications. These transformations (Walsh-Hadamard, Haar, etc.) require special processing, not necessarily compatible with a hardwired FFT.

Second, while the Associative Processor design changes improved the matrix and vector operations, they still did not address the issue of data movement within the processor, as would be encountered in a matrix inversion. It also did not address the problem of hardware inefficiency resulting from the fact that the size of most matrices may not, and usually won't, correspond to the physical dimensions of the hardware. Further information on the Matrix Parallel Processor can be found in [7.10].

7.2.3 General Purpose Array Processor

Figure 7.1 illustrates a General Purpose Array Processor (GPAP), the third version of the parallel processor. This design is indicative of a class of ensemble processors such as the ILLIAC IV and PEPE [7.2 to 7.4], which utiliz

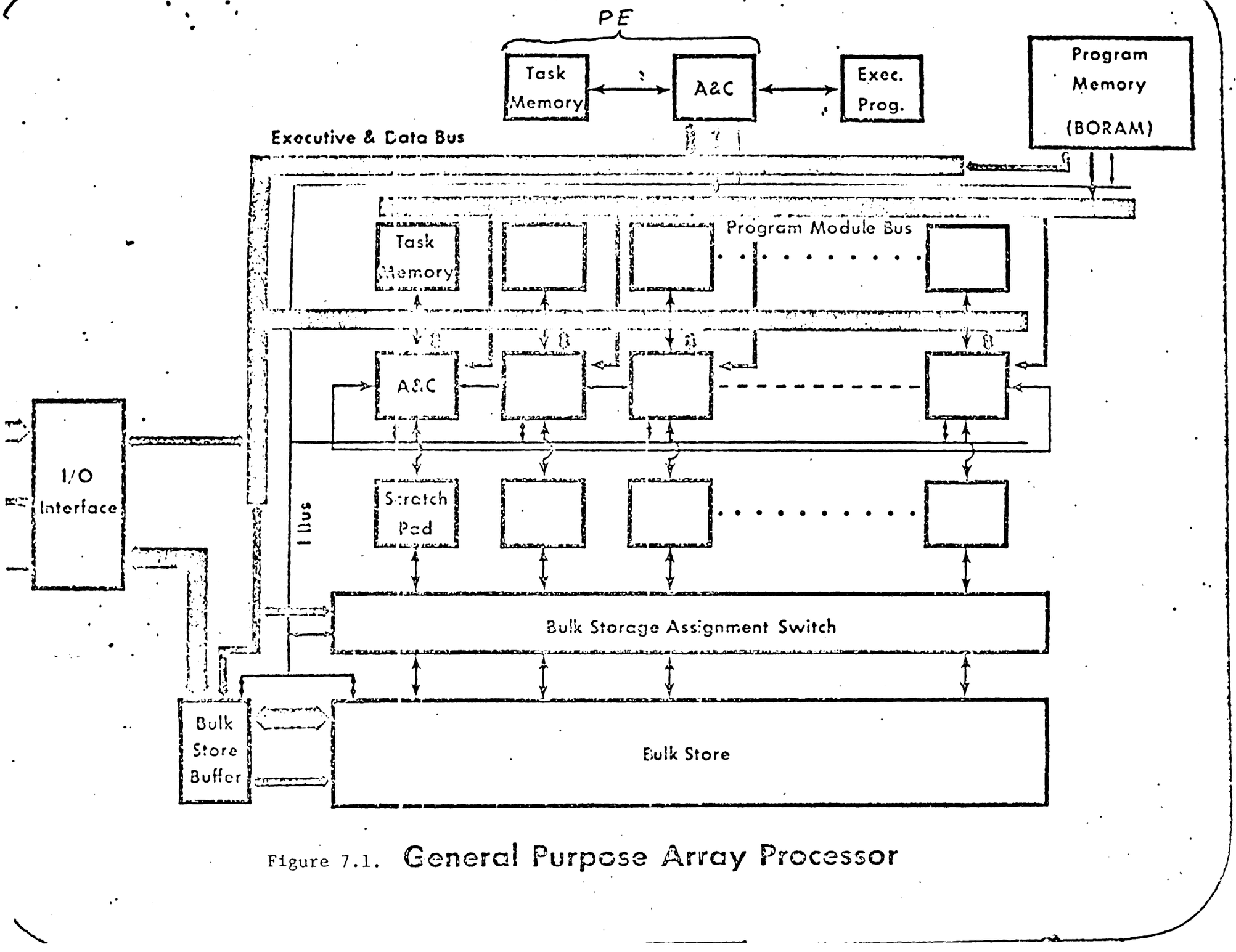


Figure 7.1. General Purpose Array Processor

reasonably competent Array Processing Elements (APEs) in place of the processor/memory cells found in the Associative Processor. In addition to the improved arithmetic and logic capability provided by this organization, each APE is provided with sufficient quantities of procedure and data storage to manage computational problems of moderate complexity. Where the ensemble processor differs from the sequential multiprocessor are in the areas of control and data management. In the GPAP, each APE may be slaved to a global controller (a supervisory DPE) thereby allowing common management of extensive, distributed computation. In essence, the ensemble processor accepts a large body of data, partitions the data into digestible sets, then operates on these data sets in parallel. As such, the Bulk Memory, which appears at the bottom of the GPAP diagram, might reasonably be considered a signal converter, since it converts very high frequency (100 M bits per sec) sequential signal into several lower frequency parallel signals (i.e., into twenty 5 M bits per sec signals) for processing.

If the distributed operations are identical and synchronous, then the overall process may be controlled by the global controller. If the processes are asynchronous or nonidentical, then control is passed to the program stored in local Task Memory. In such instances, the global controller is only used to supervise system program and data transfers, and manage system interrupts.

Among the advantages of this type of structure over the Associative and Fast Fourier Processor are its ability to perform matrix and vector operations efficiently, its ability to handle the signal processing requirement for synthetic aperature radar mapping and other "holographic"-like* functions,

---

* i.e. transformations on video signals.

and its ability to perform domain transformations (i.e., frequency to time domain). All these functions can be programmed into the array, and optimized using conventional software and programming tools, such as high level language compilers. Further description of the General Purpose Array Processor can be found in [7.11 to 7.14]. The current parallel processor design will be described in the next section.

7.3 CURRENT SIGNAL PROCESSING ELEMENT

7.3.1. Introduction

The current Signal Processing Element (SPE)* is a high-performance signal processing facility for radar, sonar, and communcation systems. The design of the SPE provides for efficient, flexible solutions to problems suited to digital signal processing machines. The SPE is intended to be compatible with the Navy All Application Digital Computer (AADC) system now under development, but is also intended as a stand-alone signal processor.

The SPE consists of the following elements:

Microprogrammed Control Unit (MCU)

Signal Processing Arithmetic Unit (SPAU)

Buffer memories or Buffer Store

Storage Control Unit (SCU)

Input/output system.

The SPE elements for the Advanced Development Model are to be implemented with "off-the-shelf" components. Bipolar monolithic storage devices and TTL Schottky family logic are to be used. Performance specifications include:

MCU basic microinstruction	150 nsec
Buffer memory cycle	150 nsec
SPAU-equivalent complex operation	300 nsec
(four multiplications and six adds)	

Performance is compatible with projected AADC technology, and efficient operation can be expected under stand-alone or system-integrated conditions. The material in this section is a summary of [7.17-7.20].

---

*the current SPE was developed at NRL (Navy Research Lab, Washington, D.C.).

7.3.1.1. Functional Description

The SPE is designed as a tool for processing digital data streams. The heart of the SPE is the Microprogrammed Control Unit which serves as system superviser and data organizer for the Signal Processing Arithmetic Unit and other I/O devices in the system. Microprogrammed operations in the MCU process 16-bit-wide data accessed from 32-bit-wide buffer memories and control buffered and unbuffered I/O operations to and from SPE devices.

The Signal Processing Arithmetic Unit performs special data processing operations such as Fast Fourier Transforms, recursive filtering, and correlation under direction of the MCU. Parallel organization of fast multiply and add logic units allow for high-speed execution of these functions. Interfacing between the SPAU and MCU is via buffer memories and the I/O system.

It is the responsibility of the Storage Control Unit to manage accesses to buffer memories by the elements of the SPE. The MCU, SPAU, and other buffered devices in the system access buffer memories independently under their own control, and the SCU resolves conflicts for buffer cycles on a priority basis.

The I/O system is designed to allow expansion of the SPE so that multiple MCU's and SPAU's can communicate and coordinate processing of increased data bandwidths.

Figure 7.2 is a block diagram of the SPE.

7.3.2. SPE's Microprogrammed Control Unit (MCU)

The Microprogrammed Control Unit is a high-speed, executive, input-output processor and interrupt handler for the NRL SPE. Since the MCU is the microprogrammable executive for the SPE, users will write microprograms (or have them written) which will direct and control all elements of the SPE. It

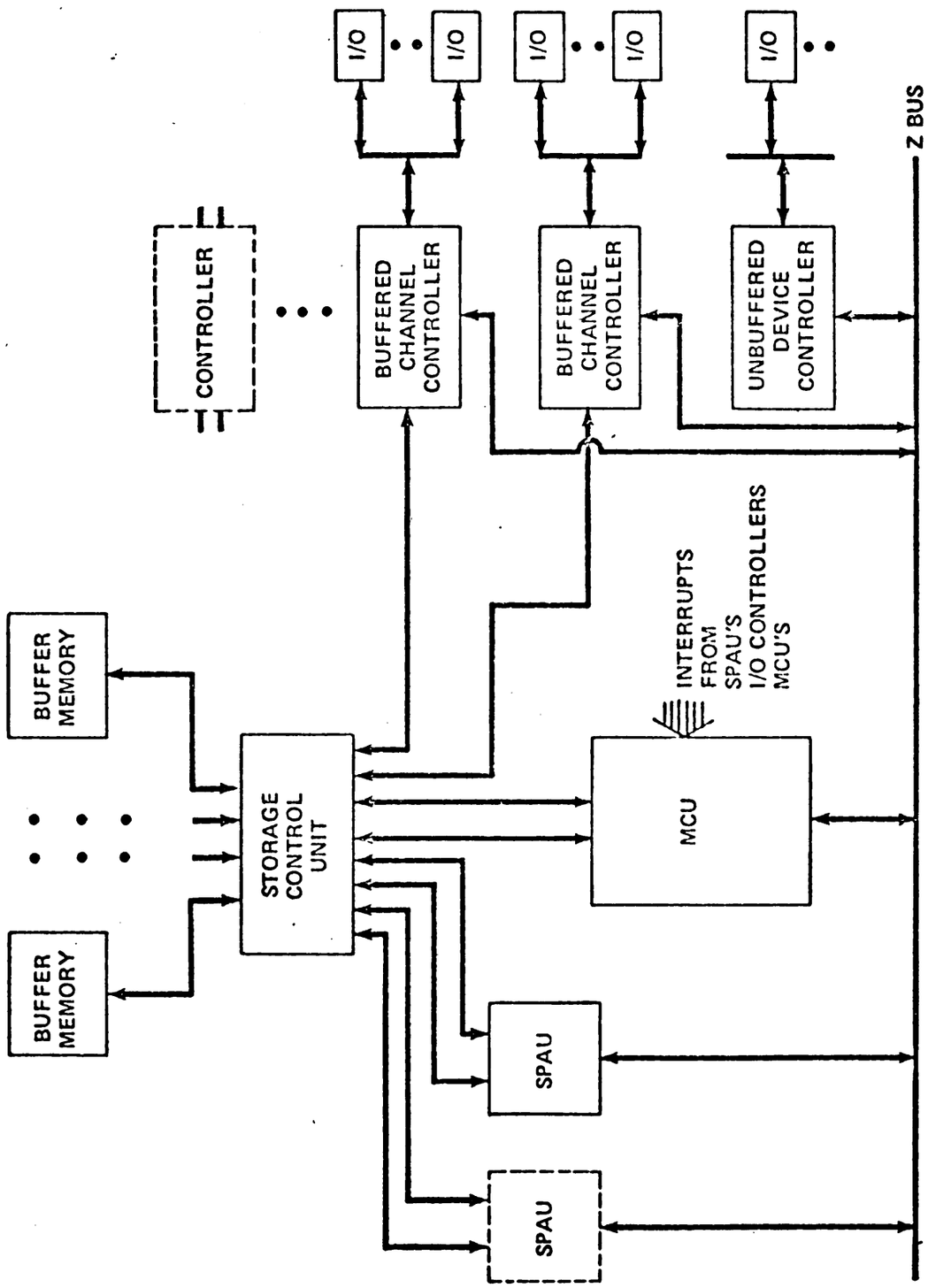


Figure 7.2 - SPE system

is the responsibility of the MCU to initiate and keep records of all I/O operations. Concurrently, the MCU may be doing preprocessing on a block of data before requesting action from the SPAU. Similarly, it may have to do postprocessing of SPAU output before outputting the results or sending them back to the SPAU for yet another operation. In addition to these functions, the MCU must service the interrupts from the SPAU, Buffered Channel Controllers, Unbuffered Device Controllers, and other MCU's, if any. To handle all of these responsibilities, it is necessary for the MCU to do many things at a very fast rate. As a result, the MCU operates at a 150-nsec clock cycle time, with the ability to do all operations, including buffer memory accesses, within one cycle. To achieve this high rate of control, the MCU operates from a single-format, 64-bit-wide, microprogram control word. From this wide control word, it is possible to achieve benefits such as increased speed due to the highly decoded fields and high hardware utilization (and, therefore, performance improvement) from the ability to control all of the registers and gates during each cycle.

Thus the MCU serves as system supervisor and data organizer for the SPAU and other I/O devices. The MCU includes a 64-bit Control Store, two local stores, an arithmetic element, two busses to buffer memory, an unbuffered byte channel, and a priority interrupt system. The next subsection will examine the MCU architecture and operation.

7.3.2.1. MCU Architecture and Operation

To obtain the basic clock rates, the MCU must be simple, but to do the required work it must have considerable parallel-operating hardware. These requirements dictate the design shown in Figure 7.3 and the overlapping operations and timing shown in Figure 7.4.

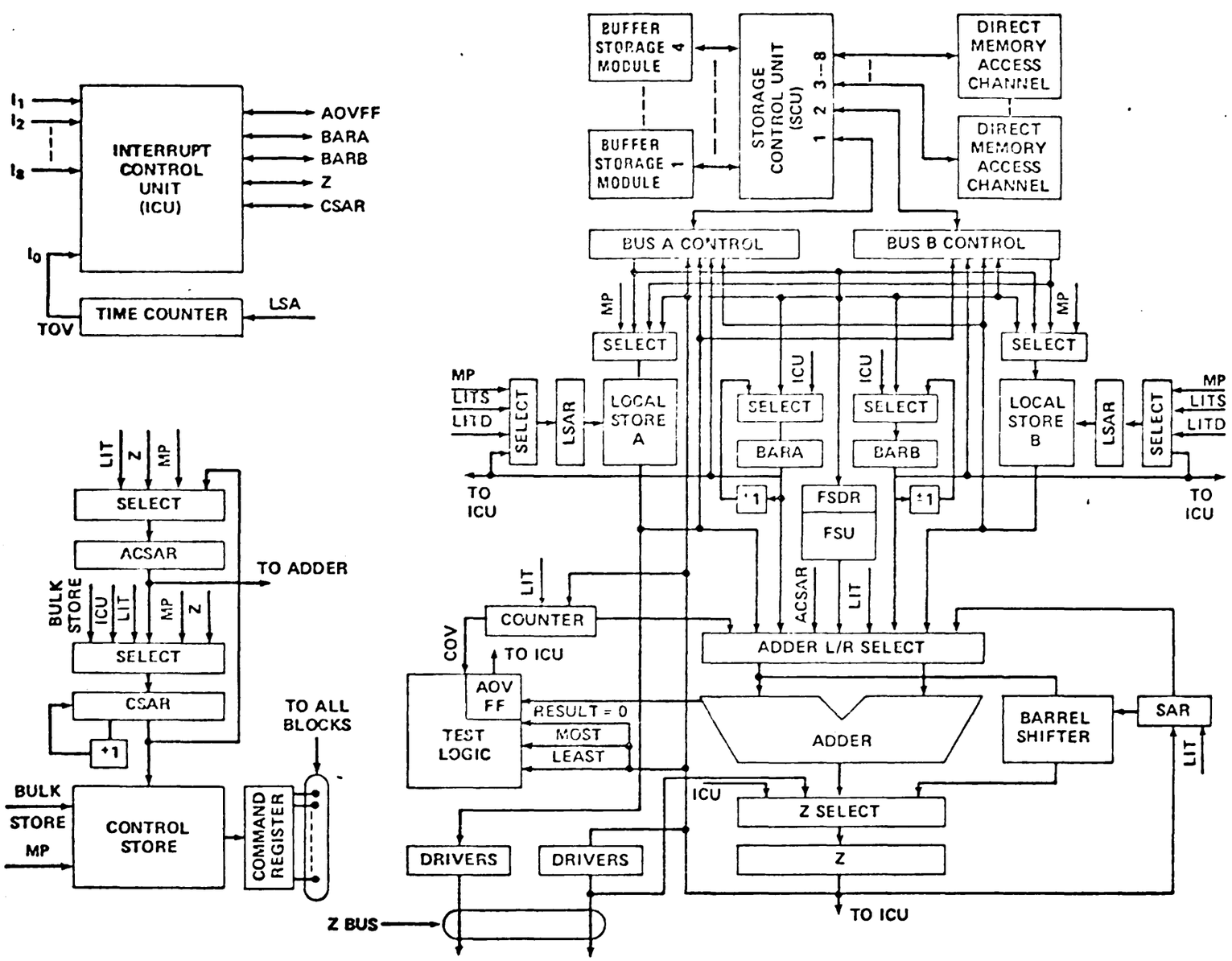


Figure 7.3 Microprogrammed Control Unit for SPE

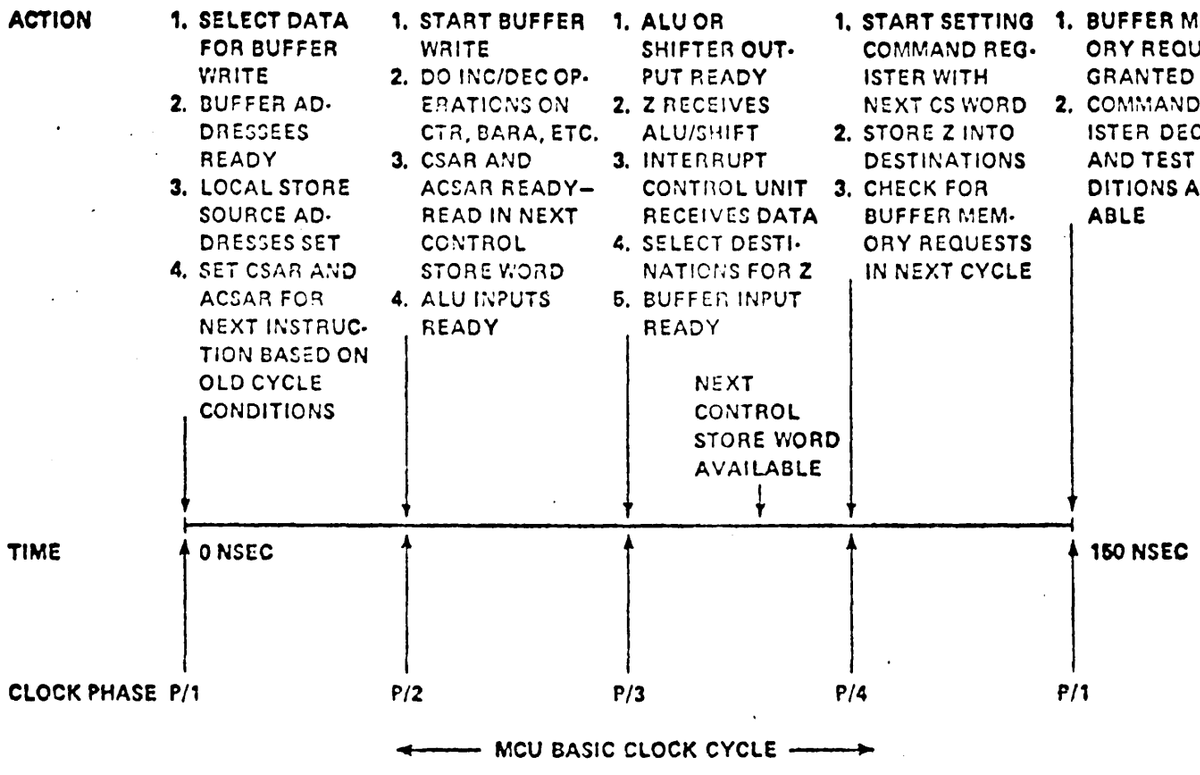


Figure 7.4 MCU Timing Diagram

All data entering or leaving the MCU must travel over one of two channels, Bus A or Bus B, via the Storage Control Unit to the buffer memory. Each channel can be used for one memory operation during every MCU cycle due to the matched speed of the MCU and the buffer memories. See Figure 7.3.

Associated with each 32 bit bus is an address register. Bus A Address Register (BARA) goes with Bus A and, similarly, Bus B Address Register (BARB) goes with Bus B. Each register has 16 bits, composed of 12 bits of word address, 3 bits of buffer address, and 1 bit for half-word addressing since each buffer memory word read out and transferred is 32 bits wide. To aid in buffer memory addressing, each address register has an incrementer and decrementer associated with it.

To store data from these busses or intermediate results generated by the MCU, there are two small very fast (30 nsec access) memories, Local Store A and Local Store B. Each is 16 words by 16 bits with the capability of being both read out and stored into during the same cycle. Double addressing capabilities are associated with each local store whereby different addresses can be specified for read and write operations in the same instruction cycle.

For indirect (computable) addressing, a default scheme is incorporated which allows the least significant four bits of the adjacent bus address register (BARA with Local Store A and BARB with Local Store B) to supply the local store address. This occurs whenever the control store field address is zero.

To perform basic arithmetic and logical operations, the MCU has both an added and a shifter (sometimes called an Arithmetic and Logic Unit).

The adder can perform 16 basic operations including add, subtract, and full Boolean operations. Binary operations are performed on two 16-bit words, one on the left input (L) and one on the right input (R). The output is delivered to the 16-bit Z register for gating to other MCU registers. The shifter is a barrel switch which allows shifting of the adder output by any number of bits within 20 nsec. The number of bits to shift is specified by the Shift Amount Register (SAR). Output from the adder/shifter goes via the 16-bit Z register into local stores, buffer address registers, and other registers in the same cycle.

One support register, the Shift Amount Register, has already been mentioned. Another is the Counter (CTR) which can be loaded with a literal value and counted up to overflow which can be checked and thus cause appropriate action. Other conditions that can be checked are based on results of the last adder operation and include adder overflow, result equal to zero, Z register most significant bit set (sign), and X register least significant bit set (odd or even, flag, etc.).

Two registers are provided for control store address selection. The Control Store Address Register (CSAR) is the only one which addresses the writable Control Store. It can be set from the other address selection register, the Alternate Control Store Address Register (ACSAR), the literal field of the control word, or from its incrementer. In addition to these, the Interrupt Control Unit (ICU) can set the CSAR to allow for interrupt handling. At the beginning of each cycle, the CSAR contains the address of the currently executing control word. Under direction of the new control word, the CSAR and ACSAR are selectively altered from one of eight choices. For example, in normal sequential program stepping, the CSAR is

incremented during each clock cycle and the ACSAR is unchanged. For subroutine calls the ACSAR retains the return address (the old CSAR +1) and the CSAR holds the address of the subroutine.

The Interrupt Control Unit (ICU) mentioned earlier contains no programmable elements. Upon receipt of an interrupt of higher priority than the current level executing in the MCU, the MCU operations are suspended, all necessary registers are saved, and the appropriate interrupt handling routine address is passed to the CSAR. This routine executes then restores the MCU to its preinterrupt status. The user will be unaware of this action except for deviations in expected execution times.

I/O action is initiated by the MCU by sending out an I/O command over the Z bus. The programmer must select the proper command operation code, count, device address, buffer address, etc., to be sent out on the Z bus. This process will be discussed further in Subsection 7.3.5.

The last element of the MCU is the Field Select Unit (FSU). This device allows the programmer to address fields within a word. As data are brought in over bus A, the programmer may specify that during any transfer the 32 bits of data also be put into the Field Select Data Register (FSDR). In subsequent cycles after this operation, the user may select one of seven predefined fields [7.17, page 6] from the FSDR as an operand for the adder. The output will be a 16-bit value with the selected field right justified with leading zeroes.

The two most important attributes of the MCU is its speed and flexibility. The speed is obtained largely from the overlapping of operations. Figure 7.4 shows the MCU's 150 nsec cycle time broken down into four subcycles with an average of four events being performed in each sub-

cycle. This gives some idea of the amount of concurrency allowed by this architecture.

Much of this subsection is taken from [7.17]. A more detailed explaination of the MCU architecture and operation is available in [7.18, pp 2-9] or [7.19, pp 2-17]. A description of the 64-bit control field is available in one of [7.17, pp 7, 8; 7.18, pp 22-31; or 7.19, pp 27-34]. For a discussion of the MCU programming language, see Subsection 7.3.6.

7.3.3 Signal Processing Arithmetic Unit (SPAU)

The Signal Processing Arithmetic Unit operates under direction of the Microprogrammed Control Unit. It is a special-purpose hardware device designed to provide very high-speed processing of Fast Fourier Transforms, recursive filter, and other signal processing algorithms. Its performance is indicated by a time of 300 nsec (two MCU cycles) to complete an SPAU-equivalent complex operation (four multiplications and six additions).

Two major sections provide the processing functions of the SPAU. These are the Arithmetic and Control Section (ACS) and the Address Generator and Control Section (AGCS). Both sections operate under microprogram control from read-only or read-mostly memories. These two major sections are later subdivided in 5 smaller sections.

The Arithmetic and Control Section contains four high-speed multipliers (185 nsec) and six high-speed adders (25 nsec) which can operate in various parallel or serial configurations as governed by the microprogram control. Direct access to SPE buffer memories is provided via two buffered data channels allowing high data throughput in the SPAU.

The Address Generator and Control Section contains adders, counters, and other logic elements and provides the function of computing addresses

needed by the Arithmetic and Control Section to access buffers and internal stores containing data used by the signal processing operations.

7.3.3.1 Design Objectives of SPAU

The SPAU has been designed to attain two primary objectives, high speed and efficiency, in the execution of signal processing algorithms. The former has been accomplished by using four parallel hardware multipliers and four adders in the section which performs arithmetic operations on the input data, and by concurrently generating memory addresses in a separate section which uses three parallel adders and three counters. High efficiency, that is the ability to keep most of the hardware busy most of the time, is accomplished by providing many data transfer options to the multipliers and adders.

During the design process, major emphasis has been placed on two signal processing algorithms: the Fast Fourier Transform (FFT), and the second-order recursive filter. Another objective has been to provide flexibility for the efficient execution of other algorithms, such as data and spectrum weighting (Hanning) and vector and matrix operations. This overall flexibility has led to a wide control word (154 bits).

7.3.3.2 SPAU Architecture

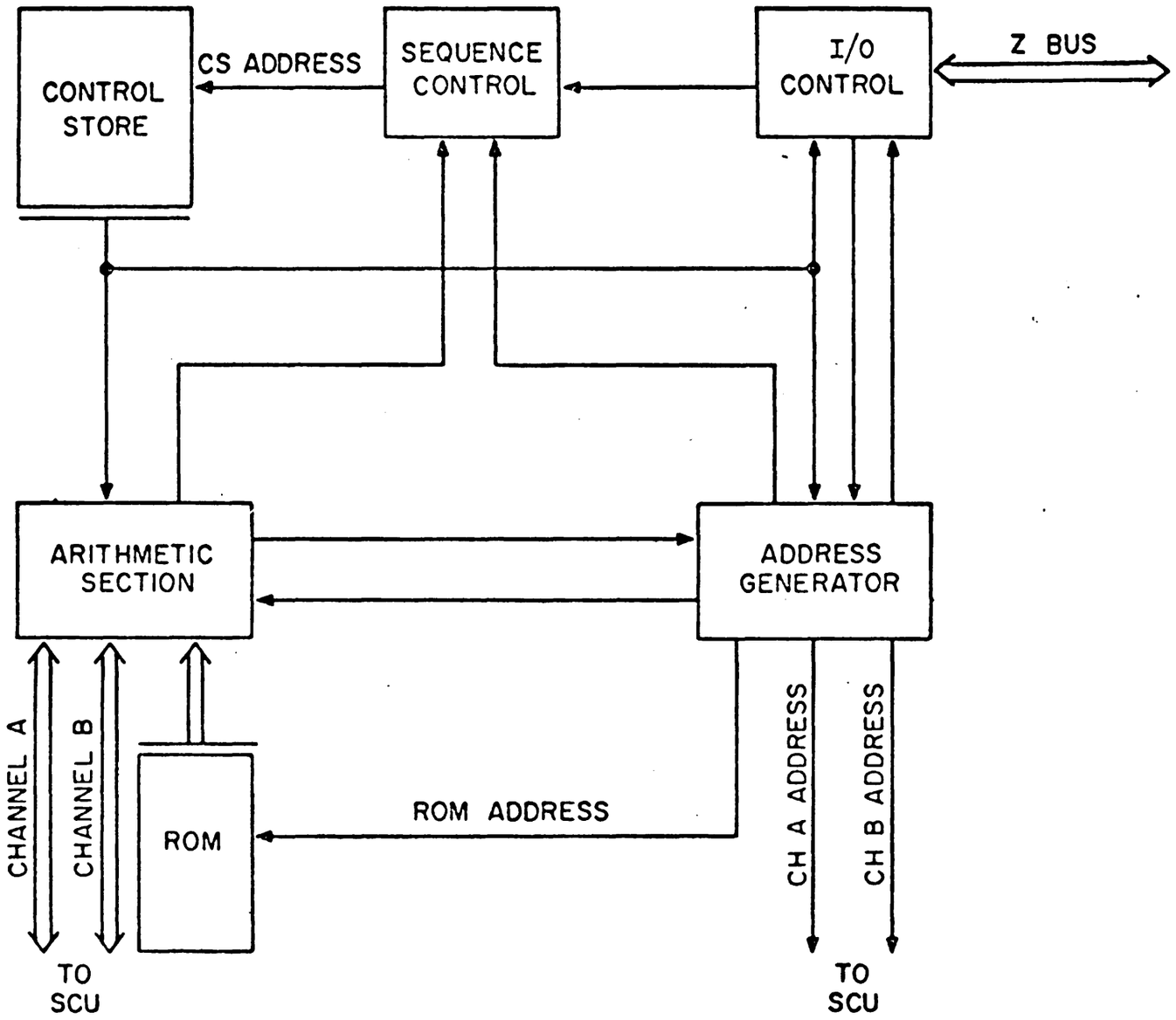
As shown previously, Figure 7.2 illustrates the relationship of the SPAU to the other elements in the SPE. The SPAU communicates with the Microprogrammed Control Units (MCU) by means of the Z bus and buffer memories. Input and output data areas residing in one or two buffer memories are assigned by an MCU each time the MCU issues a "macro" command to the SPAU. After receiving a macro, the SPAU operates in a stand-alone mode until it has finished the assigned task, then it sends an interrupt signal to the MCU which called it indicating that the macro has been completed.

In order to operate in this manner there are five functionally different sections combined within a SPAU, as illustrated in Figure 7.5. There are: the Arithmetic Section (AS), the Address Generator (AG), the Sequence Unit (SU), the Control Store (CS), and the Input/Output Control Unit (IOCU). The Arithmetic Section contains four high-speed (185 nsec) 16-bit multipliers; four high-speed (25 nsec) 16-bit adders (arithmetic logic units); four each of input, product, and result registers; and four 16-word 16-bit local stores which are "ganged" in pairs (the two stores in a pair have common Read and Write addresses). The Address Generator contains three adders, three counters, three output and three result registers, and a single 16-word 16-bit local store. Communication is provided between the AS and AG local stores in order to facilitate data dependent addressing. A read-only memory (ROM) contains 1025 sine and cosine coefficients, each 12 bits wide, for use in the FFT, plus often-used filter coefficients and other constants.

7.3.3.3 SPAU Operation

A SPAU operation may be initiated by an MCU sending an inquiry signal on the Z bus, and receiving a "not busy" reply from the SPAU. The MCU then sends a linkage message which includes the identity of the macro being requested, and its associated parameters. The message is transmitted via the IO Control Unit to the W store in the Address Generator and thence, as required, to the X and Y stores in the Arithmetic Section. The starting address of the particular macro in question is set up on the Sequence Unit, and operation of both the Arithmetic Section and the Address Generator begins.

The normal sequence of control is an unconditional step from one instruction to the next; however, this sequence can be altered by testing



SPAU–FUNCTIONAL UNIT BLOCK DIAGRAM

Figure 7.5

any one of fifteen other conditions in the AS and AG hardware, and transferring control to one of seven other successors. A new instruction is fetched every 150 nanoseconds (nsec) unless a buffer memory access is denied to the SPAU, in which case the unit idles, re-requesting the memory access. Data are transferred to and from buffer memories over two channels, denoted by A and B, of 32 bits width. The X and Y stores are each partitioned into 16-bit halves, X_1, X_2, and Y_1, Y_2, respectively, to operate with the 16-bit hardware of the Arithmetic Section.

Adder outputs may be loaded directly into result registers, R1 through R7, and multiplier outputs are always loaded into product registers, P1 through P4. There are also four input registers, denoted by Z1 through Z4, which may be loaded from the ROM or from the source that is otherwise indicated (in the control word) for X and Y. The inputs to the Arithmetic Section multipliers and adders are obtained from X, Y, the Z registers, the P registers, and R1 through R4.

In the Address Generator, the memory addresses are held in registers denoted by BARA, BARB, and RAR for channels A and B, and the ROM, respectively. Their contents are normally incremented by amounts contained in registers INCA, INCB, and INCR, respectively. The inputs to the Address Generator adders are obtained from the address registers, the INC registers, the W store, or the literal field of the control word. The literal field and W are 16 bits wide; only the 11 least significant bits and the sign bit (the most significant bit) are used in the 12-bit Address Generator hardware.

As stated in [7.20] the design of the SPAU is still in the preliminary stage and thus the material in this subsection may be subject to change. The final design is expected in March 1973.

7.3.4 Buffer Memories and Storage Control Unit

7.3.4.1 Buffer Memories

An SPE can have a maximum of eight buffer memories. Each buffer memory consists of up to 4096 words of 32 bits and has a separate 32-bit data port. The b uffer memories use static, bipolar, monolithic storage devices which are compatible with TTL logic. The read/write cycle time is 150 nsec. The memories are contained on printed circuit boards which are placed in 19-in. wide panel racks.

Each buffer memory is independently accessible through its own port. MCU's, SPAU's, and peripheral devices must contend for buffer memory access on a cycle-by-cycle basis. It is the responsibility of the Storage Control Unit (SCU) to resolve memory access conflicts.

7.3.4.2 Storage Control Unit (SCU)

All SPE devices (MCU's, SPAU's, peripherals) which require buffer memory access are interfaced to the memories through the Storage Control Unit. The SCU can interface up to eight data channels with up to eight buffer memories. Any channel may access any buffer memory at any time. Whether or not the buffer cycle which is requested is granted depends on the priority of the requesting channel and the state of the other channels. Channel priority is hard wired and determined by the physical location of the channel at the SCU.

Requests for a buffer cycle are made by a device raising a buffer request line along with the buffer address lines. The SCU records all buffer requests every clock cycle (150 nsec) and returns a Request Granted line to each device receiving its requested buffer cycle. If two or more devices request the same buffer on the same cycle, only the highest priority channel will receive the Request Granted line.

It is the responsibility of any device to remain idle pending a positive response by the SCU to its buffer cycle request. This is done automatically by the MCU and SPAU. A little more detail on the SCU can be obtained from [7.19, pg 17-18].

7.3.5 Input/Output System

The SPE Input/Output and internal communications are provided by Direct Memory Access buffered data channels, a single unbuffered byte channel, and a priority interrupt system. The unbuffered byte channel called the Z Bus communicates both data and control information to all I/O devices. Eight/sixteen buffered channels enable high-speed data transfer between buffer memories and system devices or MCU's. The Z bus allows direct communication under MCU control and on a word-by-word basis between the Z register of an MCU and all devices connected to the Z bus. The Z bus also enables direct MCU-to-MCU communication.

Figure 7.2 in Subsection 7.3.1 showed an SPE configuration with I/O system elements and interconnections.

Devices which access buffer memory over buffered channels are interfaced to the buffered channels by Selector Channel Controllers (SCC) (Not shown in Figure 7.2). SCC's also interface with the Z bus and are responsible for interpreting device requests coming over the Z bus from MCU's. These requests originate in the form of MCU I/O instructions and can call upon an SCC to initiate various device I/O operations over its buffered channel interface.

The SCC's are intended to be standard I/O elements interfacing between buffered channels and Device Controllers (DC). DC's interface between SCC's and I/O devices and must be tailored to meet the interface

requirements of a particular device type. DC's interface to SCC's over Z-bus-compatible connections. This allows a DC to connect directly to the Z bus for direct unbuffered communication with an MCU or to connect to an SCC for buffered channel communication.

SCC's and DC's can request MCU action via interrupt lines provided in the MCU's for such purposes. Separate SCC's or DC's sharing a single interrupt line must have hardware to resolve competition among the units for interrupt service.

An MCU generating an I/O request addressed to another MCU for the purpose of MCU-to-MCU communication causes the addressed MCU to raise an internal interrupt line. An I/O acknowledge instruction by the interrupted MCU completes the data transfer over the Z bus.

The Z bus consists of 30 lines, 16 of them are bidirectional data lines, 8 are bidirectional device address lines and 6 of them are other control lines. The maximum burst transfer rate over the Z bus, based upon an MCU cycle time of 150 nsec, is 2 MHz. The Z bus is used by the MCU to exchange commands and unbuffered data with I/O devices. The MCU is interface to the Z bus through the Z register and the Local Store A (See Subsection 7.3.2 and Figure 7.3).

Since three references [7.17, pp 19-25; 7.18, pp 9-15; and 7.19, pp 21-27] all contain (almost) identical detailed descriptions of the SPE Input/Output System and its operation, no further description will be presented here.

7.3.6 A Microprogramming Language (AMIL)

One of the most significant outcomes of the NRL SPE development program is the creation of a Fortran-like language for programming the SPE.

Since the control unit of the SPE is the Microprogrammed Control Unit, all programming would normally be done by specifying binary bit pattern which is a difficult and tedious task. For this reason, a new language called AMIL (A MIcroprogramming Language) has been created to allow users to write microprograms in a Fortran-like register transfer language as opposed to ones and zeroes. As a result, users will now be able to write AMIL programs and allow the AMIL translator to convert his program into MCU bit patterns. This translator has been developed and is currently operational on a time-sharing service available to NRL.

AMIL is syntactically described using Backus-Naur Form (BNF) with semantic descriptions in [7.17, pp 9-18]. A complete AMIL Syntax is given in [7.17, Appendix B] with key words listed in [7.17, appendix C]. Two sample programs and their output from the AMIL translator are shown in [7.17, appendix D] and a complete listing of error messages generated by the translator is shown in [7.17, appendix E]. In all, AMIL looks like a well defined and very useful language for controlling the microprogrammed SPE.

7.4 COMPARISON OF DPE AND SPE

Since the Data Processing Element (DPE) described in Chapter 5 and the Signal Processing Element (SPE) are substantially different processors, a brief comparison will be presented. The DPE is intended for all sequential processing, although it does have some matrix handling capabilities. The DPE is expected to have throughput of less than 10 MIPS even in advanced designs and about 2 MIPS for the Advanced Development Model. On the other hand, the SPE is designed for radar, acoustic and video signal processing where throughput of 10 to 200 MIPS may be required.

Figure 7.6 compares the major component building blocks of the SPE with the building blocks of the sequential DPE. In principle, the SPE achieves its high throughput based on the fact that signal processing entails reiterative processing of relatively continuous data streams. This allows pipeline computation where, once the processor pipeline is filled, the total throughput of the machine is basically limited by the time it takes to complete the longest operation in the sequence. Obviously, in order to keep a processor pipeline busy, it is necessary to stream instructions to all elements in the pipeline simultaneously. Because the number of operations which must be controlled in this manner is greater than can be managed using a relatively short OP CODE, microprogramming methods are used.

While extensive microprogramming was found non-essential and hardly cost-effective for the DPE, it is an absolute necessity in the SPE. Furthermore, the software management problem raised by microprogramming for the DPE does not seem as grave in the case of the SPE. Whereas, for the sequential DPE, microprogramming could result in uncontrollable modification

Property	Data Processing Element	Signal Processing Element
Required Throughput	<1 MIPS	>10 MIPS
Arithmetic	Real and Non-numerical data	Complex variables
Precision	high (32 bits or higher)	low (16 bits)
Array Sizes	small arrays	large arrays
Branching	considerable branching	little branching
Procedure and data sizes	Procedure large, data small (1/10 of precedure)	Procedures small, data large (100 times procedure)
Arithmetic Unit	AU: Floating Point 32 bit arithmetic 750 nsec multiply	SPAU: Fixed Point 16 bit complex halves 300 nsec for 4 multiplies and 6 additions
Control Unit	PMU: Instruction fetch Unit 32-bit instruction Long sequence on major cycles	MCU: Instruction fetch unit 64-bit microinstructions Tight loop for major major cycle
Queue	APQ from PMU to AU 16 words x 41 bits	Buffer MCU to SPAU 4 x 4K words x 32 bits
Local Instruction Memory	Task Memory: RAM 4K words x 36 bits	Microprogram memory 1024 word modules x 64 bits Expandable to 4K word.
Local Data Memory	16 words x 42 bits	Two 16 words x 16 bits

Figure 7.6 A Comparison of the DPE and the SPE

of system architecture to the extent that (a) HOL compilers could not be easily written (if at all), (b) programs could not be easily run in multi-programmed environments and (c) software maintenance might only be attempted by the programmers responsible for their "individualized" instruction sets. The operational conditions for the microprogrammed SPE circumvent most of these problems. First, the applications for the SPE constitute a much smaller set than the potential uses for the DPE by the very nature of the cost/performance ratio of microprogrammed hardware. Secondly, the funa-mental operations which constitute the kernels for signal processing can be developed once and then maintained in the sense of "firmware", allowing, in turn, reasonable measures of software design automation through macro-synthesis [7.15].

7.5 CURRENT AND FUTURE DEVELOPMENTS

The current developments for the Signal Processing Element are concentrating on a test-bed model to be built by March 1974(the same time as the scheduled delivery of the Advanced Development Model for the sequential DPE). The SPE test-bed model is to be built with off-the-shelf equipment and to be compatible with other AADC components. The Microprogrammed Control Unit is to have 150 nsec cycle time and be capable of emulating the Q20 and AN/UYK-7 computers. The microprogram memory is to be 150 nsec bi-polar 2K words x 32 bits and to be RAMM (Random Access Main Memory) and Task Memory Compatible. The SPAU and other SPE components for the test-bed model will be built as described in the previous section.

The objectives of the SPE test-bed are to produce a facility to be used as a:

1. System simulation laboratory
2. System configuration laboratory
3. Benchmark facility.

The development of programming languages to be used with the SPE is a very important area for future development. The programming languages can be divided into three categories:

1. Support software for program development include a micro-programming language translator
2. Executive software for MCU
3. High level languages for signal processing applications.

Some work has already been completed on developing support software for program development. The AMIL (A MIcroprogramming Language) and its translator have been developed and are now running. (See Subsection 7.3.6).

A MCU simulator has also been developed to check the translator output and to act as the first step in developing an SPE simulator. The basic support software is scheduled for release in April 1973. It will be available to anyone who wants it.

The program development that is currently in progress includes developing an SPAU simulator which will then be combined with the MCU simulator and the AMIL translator. The result will be an SPE simulator that can be programmed directly in a Fortran-like language. This should prove to be a very valuable tool for SPE system configuration and check-out, as well as for program development.

The development of executive software for SPE is just beginning. The development of high level languages for signal processing is still an area that needs a lot of research.

References for Signal Processing Element

7.1 Slide Presentation on AADC for FY72; R. S. Entner, NAVAIRSYSCOM; Undated, probably spring 1972; Unpublished; Unclassified; (NPS).

7.2 An Introduction to the ILLIAC IV Computer; D. E. McIntyre; Datamation; Vol. 16, No. 4; April, 1970; (NPS).

7.3 The ILLIAC IV Computer; G. H. Barnes, R. M. Brown, D. L. Slotnick and R. A. Stokes; IEEE Transactions on Computers; Vol. C-17, No. 8; October 1968; (NPS).

7.4 PEPE: A Parallel Processor; Digest of Papers for COMPCON 72; September 12-14, 1972; IEEE Catalog No. 72CH0659-3C; pp 57-72; Contains 4 papers on PEPE Computer Architecture, Support Software System, Application to Radar Data Processing and Application to the Ballistic Missile Defence Radar Data Processing; (NPS).

7.5 Baseline Associative Processor; John E. Shore and Frank A. Polkingham; NRL; March 1969; Unclassified; (6).

7.6 AADC Associative Processor, Interim Report; John E. Shore; NRL; May 24, 1970; Unclassified; (30).

7.7 Associative Processor for Air Traffic Control; Kenneth J. Thurber, Honeywell Systems and Research Center; SJCC 1971; pp 49-59; (NPS).

7.8 Another Associative Processor; J. E. Shore and T. L. Collins; NRL Report 73-48; December 1971; (69, NPS).

7.9 Software Simulation of an Associative Processor; J. E. Shore; NRL Report 7351; December 1971; (70 NPS).

7.10 Matrix-Parallel Processor Study - Final Report; Westinghouse Electric Corporation; 15 September 1970; NAVAIRSYSCOM Contract N00019-70-0264; Unclassified-NOFORN; (39, NPS).

7.11 Development of a General Purpose Signal Processor - Signal Processing Tasks of the AN/TPS59; NRL Memorandum Report 23-53; November 1971; Confidential; (68).

7.12 Second Thoughts on Parallel Processing; J. E. Shore; NRL Reports 73 64; December 1971; (71, NPS).

7.13 The Advanced Avionics Digital Computer Revisited; R. S. Entner; NAVAIR-SYSCOM; October 12, 1971; Unpublished paper; Unclassified; (NPS).

7.14 AADC Development Program Progress Report No. 8; R. S. Entner; NAVAIRSYSCOM; July 1, 1971; Unclassified; AD-727-607; (58, NPS).

7.15 AADC Development Program Progress Report No. 10; R. S. Entner; NAVAIRSYSCOM: May 31, 1972; Unclassified; (78, NPS).

7.16 Cellular Logic-In-Memory Arrays - Final Report - Part II; W. H. Kautz and M. C. Pease III; Stanford Research Institute; ONR Contract N00014-70-C-0404; November 1971; (81).

7.17 Microprogrammed Control Unit (MCU) Programming Reference Manual; John D. Roberts, Jr.; NRL Report 7476; August 15, 1972; (NPS).

7.18 Signal Processing Element, Users' Reference Manual; William R. Smith and John P. Ihnat; NRL Report 7488; Sept. 5, 1972; (NPS).

7.19 Signal Processing Element Functional Description: Part 1 - Micro-programmed Control Unit, Buffer Store and Storage Control Unit; John P. Ihnat, William R. Smith, John D. Roberts, Jr., Y. S. Wu and Bruce Wald; NRL Report 7490; Sept. 12, 1972; (80, NPS).

7.20 Signal Processing Element Functional Description: Part 2 (Preliminary)-Signal Processing Arithmetic Unit; William R. Smith and Harold H. Smith; NRL Memorandum Report 2522; October 1972;

Chapter 8

MEANS OF EVALUATING AADC DEVELOPMENTS

Table of Contents for Evaluating AADC Developments

Chapter 8

MEANS OF EVALUATING AADC DEVELOPMENTS

8.1 INTRODUCTION AND SUMMARY

Although a means of evaluating the development of AADC and accurately predicting the performance, cost and reliability is of the utmost importance, relatively little has been published on this specific subject. There are several means of evaluating the development, including:

1. Measuring the load on existing avionic computer and thereby projecting the future requirements,
2. Simulating the operation of individual AADC modules,
3. Simulating the module interaction or the overall AADC operation,
4. Simulating an application using the AADC system,
5. Modeling the operation of AADC modules,
6. Breadboarding at the DPE, memory and bussing level (equivalent to CPU, memory and channel level in more common terminology),
7. Devising a test plan for the breadboard of the model including what to measure, how to measure and how to interpret the results, and finally,
8. Producing a prototype of individual modules for testing the complete AADC system.

According to the author's count, there is one completed study on measuring the load on existing avionics computers (but there must be others) [8.1]. (The AADC is currently sponsoring advanced analytical studies with Grumman Aerospace and LTV Corporations examining the computer requirements for the F-14 and A-7 class aircrafts.)

The author also counts three studies simulating AADC modules (case 2 above) and two reports on the simulation of module interactions as a system (case 3 above), and two reports on simulating the AADC application to a particular problem area (case 4 above). There are also three reports on other facets of evaluating the AADC. Apparently, one of the current projects is to obtain Optimized Simplex Processor breadboard or Advanced Development Model. (The references for these reports are cited in Section 8.2 below,)

There are also plans in 1973 fiscal year for completing the DPE and SPE register-level simulations, assembling a SPE breadboard, procuring verification hardware for DPE and I/O, and procuring feasibility models for both the ferroacoustic and the semiconductor BORAM memories [8.2, page 17].

Therefore, the low number of reports in this area is probably not an indication of the lack of activity; but rather an indication that evaluation studies are being described along with the particular subsystem.

8.2 SPECIFIC EVALUATION STUDIES

8.2.1 Measuring the Avionic Computer Workload

As mentioned above a study on measuring the computer load on the E2B aircraft is reported in [8.1]. It was found that the E-2B workload constitutes a processor workload of less than 100,000 instructions per second, and that all jobs can be partitioned into tasks of less than 4K words. Compared to AADC performance the E-2B workload is very small, requiring only 5 percent of the processing capability of the 2 MIPS Data Processing Element. The Task Memory size has been selected as 4K words to hold any E-2B task. (TM may be expanded in later versions.)

Current projects call for the measuring of computer load on the F-14 and A-7 class aircraft [8.2, paragraph 27].

8.2.2 Simulation of Individual AADC Modules

Three studies have been reported specifically on simulating the operation of AADC modules. The first was the simulation of the Associative Processor by J. E. Shore at NRL [8.3]. (See Subsection 7.2.1 for a brief description of an associative processor as a signal processor.) The second was the simulation of the instructions of the Data Processing Element. The instructions were simulated in the exact way that they would be executed on the DPE. Thus the simulation acts as a definition of the DPE instructions, as well as, a tool for debugging programs written for the DPE. Both the Program Management Unit (PMU) and the Arithmetic Processor (AP) instructions were simulated. For further details see Subsection 5.8.1 or [8.4].

The third study on the simulation of AADC modules is the simulation of Microprogrammed Control Unit of the Signal Processing Element. The report on this work is scheduled for release in April 1973. Another project is now underway at NRL expanding the MCU simulator into a Signal Processing Element simulator. For further details see Section 7.5.

8.2.3 Simulation of AADC System

Two studies have been reported on simulating the interaction of AADC modules. The first in 1969 was a UNIVAC report of a AADC simulation module [8.5]. Apparently the Navy decided not to pursue their approach. The other study is an early (1970) simulation of AADC at NRL. The project included the simulation of BORAM, Task Memory, Random Access Main Memory, Data Processing Element and the internal busses as resources. The load was represented as demands for these resources in the event-oriented simulation in SIMSCRIPT. Reference [8.6] describes the model and the assumptions in an easy to read manner. There are no results in this reference; results are published in reports referenced in the next subsection.

8.2.4 Simulation of AADC Applications

One report, that is available on simulating AADC operation on a specific application, is the simulation of the E-2B work load on the AADC system. This simulation is a continuation of the NRL project discussed in the previous subsection. Actually this study is limited to the simulation of program modules movement, or paging, between BORAM and TM to determine the best size of the BORAM blocks (or pages) and the size of the Task Memory. [8.7 and 8.8].

A continuation to the above project, which is a simulation of the AADC with three different avionic workloads, is reported in [8.9]. The avionic workloads are the E-2B, F-111 and future AADC requirements as defined by a GE study. Simplex and multiprocessor configurations are modeled along with certain features of the proposed AADC executive operations. The operating configurations include non-paged, paged and multiprogramming configurations. The study concluded that the simplex processor configuration was sufficient

for any of the three avionic workloads. The report is very comprehensive and well written, and is well worth reading [8.9].

A recent simulation study is the one on the simulation of AADC page replacement algorithms and their affect on the AADC performance. It was concluded that the replacement algorithm has very little affect on performance. See Chapter 3 or [8.10].

8.2.5 Other Evaluation Studies

Reference [8.11] provides a Cost-by-Function model for evaluating avionic computer systems by NAVAIRDEVCEN dated March 1971. Reference [8.12] is a similar, but classified, document by NAVAIRDEVCEN dated April 1971. Reference [8.13] is a review of AADC documentation by Hughes Aircraft Company dated October 1971. (These reports may, in fact, be misplaced because they were placed here based on the titles only.)

8.3 AADC BREADBOARDS

This heading is included in this report in the anticipation that AADC breadboard modules will be a very important technique in evaluating the AADC development in the near future. The Advanced Development Model of the Data Processing Element, described in Section 5.8 and [8.14], is just such a breadboard model. It is scheduled for delivery in March 1974.

Hopefully, other means of evaluating AADC development will also be reported here in the near future. For example, the results of the following project would be interesting and useful: a simulation of the P3C or S3A aircraft workload on the hypothetical AADC High Order Language, to evaluate the features and power of the HOL and test the degree to which the hardware actually supports the "ideal" software.

References for Evaluating AADC Developments

8.1 Defining the E-2B Digital Avionics Characteristics for the Simulation of Alternate AADC Hardware Configurations (U); System Consultants Inc.; November 17, 1971; NAVAIRDEVCEN Contract N62269-70-C-0274; SECRET-NOFORN; (42).

8.2 AADC Development Program Progress Report No. 10; R. S. Entner; NAVAIRSYSCOM; May 31, 1972; Unclassified (78, NPS)*.

8.3 Software Simulation of an Associative Processor; J. E. Shore; NRL Report 7351; December 1971; (70).

8.4 Arithmetic and Control Logic Design Study for AADC - Final Report; Part III; Raytheon Corp; NADC Contract No. 62269-72-C-0023; December 1972.

8.5 Advanced Avionic Digital Computer Simulation Model, Preliminary Report (U); Univac Advanced Systems Group; November 12, 1969; Unclassified-Proprietary; (17).

8.6 Simulation Model for the AADC; William R. Smith; NRL Report 2172; September 1970; (NPS),

8.7 Simulation of AADC System Operation with an E-2B Program Workload; William R. Smith; NRL Report 7030-19; January 1971; (48, NPS).

8.8 Simulation of AADC System Operation with an E-2B Program Workload; William R. Smith; NRL Report 7259; April 1971; (NPS).

8.9 Simulation of AADC Simplex and Multiprocessor Operation; William R. Smith; NRL Report 7356; February 29, 1972; (75 NPS).

*AADC Bibliography number, and availability at the Naval Postgraduate School.

8.10 Simulation of AADC Page Replacement Algorithms; William R. Smith; NRL Memorandum Report 2464; July 1972; (NPS).

8.11 A Cost-By-Function Model for Avionic Computer Systems; NADC Report No. NADC-SD-7088; March 30, 1971; Volumes I and II; (52).

8.12 Integrated Analysis Document for AADC; (U) - Interim Report; NADC Report No. NADC-SD-7132; April 15, 1971; Confidential; (53).

8.13 AADC Documentation Reviews; Hughes Aircraft Company; October 15, 1971; NELC Contract No. N00039-70-C-3552; (66).

8.14 All Applications Digital Computer, 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceedings are not yet available.

Chapter 9

HIGH

ORDER

LANGUAGE

Table of Contents for HOL

Appendices

List of Figures for HOL

List of Tables

Chapter 9

HIGH ORDER LANGUAGE

9.1 INTRODUCTION AND SUMMARY

This chapter presents the developments in defining and producing a very powerful High Order Language that can effectively and efficiently use the AADC system - one that can significantly reduce the development, documentation and maintenance costs of the AADC Software.

For the purpose of this report, a "High Order Language (HOL)" is defined as a language with many powerful extensions beyond those in the present "high level" languages, such as Fortran, Algol and PL/I. The HOL must be capable of generating efficient executive, I/O, test, display, data, file manipulating programs. Also it must have powerful vector, matrix, list, character and bit manipulating features. (Although the equivalent of these features can be obtained in present languages, they are not easily programmed and do not execute efficiently.) For example, CMS-2 (the Navy's Compiler Monitor System) is an attempt at defining a HOL. CMS-2 has the ability to define operating system procedures in Algol-like subroutines, and it has the ability to reorganize data structures at run time. Also, CMS-2 is designed especially for real-time command and control applications, which involve large data files.

Two conferences have been held on the HOL for AADC; one in June 1970 and the other in February 1972. The second conference was a good introduction to AADC for software specialists but did not present any concrete proposals for the design of the HOL for AADC. (The proceedings of this conference are not yet available.) Three papers have been written on the updating of CMS-2 to the AADC HOL, and one paper was written on how MTACCS (Marine Tactical Air Command and Control System) requirements should affect the CMS-3 (extended CMS-2)

requirements. There is also a project currently underway to specify the goals of the AADC HOL much more precisely.

This is one of the first times that the software specialist has had a chance to influence the design of the hardware. How about some suggestions?

9.2 DESIGNING A HOL

9.2.1 Problems With Existing Computer Languages

The main problem with existing computer languages (i.e., Fortran, Algol, PL/I) is that they are designed for application programming only and for sequential processing computers only. They are not designed to produce, and are quite unsuitable for producing, the many other types of programs in a modern computer system.

In the avionic and command-and-control* fields, the problems are even worse. Some of the current software problems in these fields are long lead times, non-transferability, poor documentation, difficulty in debugging, long validation times, very high cost, and specialized highly-trained personnel are required. For example, the high cost of computer hardware, coupled with large space and weight requirements have dictated that the avionic computers to be as small as possible. This meant that programs had to be very compact and very efficient - thus favoring assembly and machine language programming. With assembly languages the programming problems are even worse than with present "high level" languages. Programs are even more difficult to write, to debug, and to document. In existing avionic systems, octal patch are allowed and frequently used for connections, there are no language standards, no algorithm banks, no modularity specifications and no cooperating hardware. Thus there is a real need for an effective High Order Language. (Most of this material is taken from the slide presentation [9.3]).

9.2.2 Advantages of HOL

HOL programming can reduce software problems because HOL programs are more easily understood, (largely) self documenting, more easily debugged

*These include NTDS and MTDS (Navy and Marine Tactical Data Systems, respectively).

and more easily maintained. Figure 9.1 gives a very simple example of the advantages of HOL over an assembly language program.

However, before a HOL can be effective, it must be able to produce efficiently executing programs for all software areas including executive, I/O test, display and data manipulation, as well as the standard application areas. Figure 9.2 shows this diagrammatically. Many other advantages will be obtained from a HOL that is effective in these areas. Useful and enforceable language standards will be feasible as soon as a HOL exists that can describe the total computer programs.

Furthermore, effective modularity will be possible with an HOL. In the past, high memory costs have resulted in highly integrated programs which have excessive subroutine sharing, excessive branching, use of programming "tricks", unpredictable (or difficult to follow) program paths, reentrant programming (instead of using another copy), and suboptimal algorithms. These programming techniques are often considered advantageous since they improve the computer performance, but in actual fact they often produce unreliable and more expensive software. The wrong way to package software is to jam procedures into a small memory like "sardines"; the right way is to package software in separate individual modules. Software modularity can be effective if organized in the following ways:

1. By Function - each function has its own module.

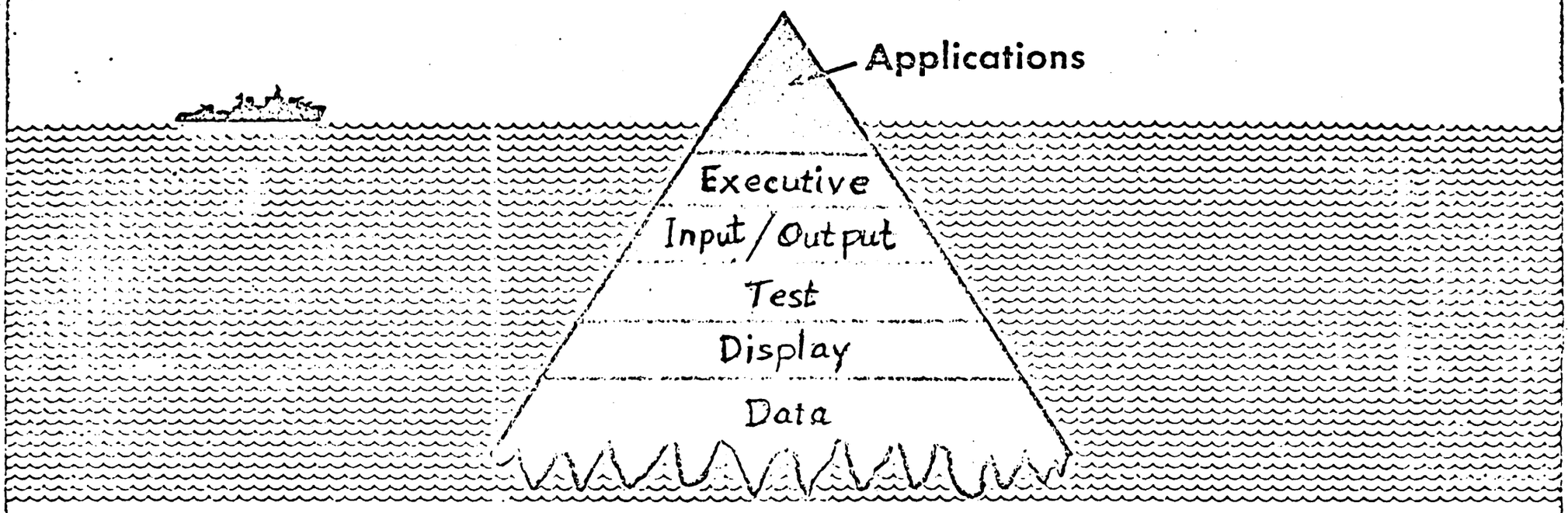
2. At Electrical Interface - allow number conversions at interfaces, i.e., a fixed-point arithmetic routine is used at one side of the electrical interface for aircraft velocity and a floating-point routine on the other side for aircraft altitude.

Sample of High Order Language (HOL)

- Arithmetic Expression
  - H = A + (B-C)(D/E+F)
- Assembly Level Program
  - Enter B
  - Subtract C
  - Store B-C
  - Enter D
  - Divide E
  - Add F
  - Multiply (B-C)
  - Add A
  - Store In H
- CMS-2 Statement
  - SET H EQ A+(B-C)*(D/E+F)

Figure 9.1. HOL vs Assembly Language

Aerospace Program Components



Present High Order Languages Cannot Be Used to Generate Efficient Executive, I/O, Test, Display Procedure or Common Data Descriptions.

Figure 9.2. Avionic Software Components

3. With Standard Mechanical Interfaces - standard module packages will optimize the "pin-to-word" counts, minimize branching and reduce fragmentation.

The advantages of good software modularity are:

1. "Where to go?" - reduces the problem of unpredictable program paths and excessive branching.

2. Software Environment - it reduces the problems of trying to fit a program into a fixed size space. One program can be divided into several modules. Also one program segment can be used by and therefore located in several modules.

3. Software Reliability - is improved by reducing the complexity of the program because there are relatively few functional modules.

HOL Algorithm banks will save excessive duplicate programming by storing test case solutions to recurring problems, such as, weapon delivery, frequency analysis, data compression and analysis, multiple source tracking data, correlation and optimization, file searches, display image generation and control, and many others. (See chapter 10.)

HOL will also overcome the problems of non-cooperating hardware which usually has fixed point arithmetic, conventional registers, conventional repertoires, software assembly-language executives and slow speed implementations. All the application programs will be written in the HOL and only the HOL compiler needs be written in some other language. (In many cases even the compiler can be written in the HOL.)

Probably the most important advantage of a High Order Language for AADC, and its raison d'etre, is that the Navy will regain some control over the ever mounting software development costs. The Navy will be able to specify HOL requirements in its contracts, and possibly by MIL spec. Thus the Navy will have much more control over the design and development of its computers, and will be much more capable of supporting and maintaining the complex computer system in the operational environments.

9.2.3 AADC Software Oriented Features

The following are some of the significant AADC software features of the Data Processing Element (Chapter 5) that help in the HOL implementation:

1. Fix and floating point arithmetic - eliminates the need to scale variables and constants.

2. General purpose push-down registers - allows instruction execution to be deferred until all data (or operands) are available.

3. HOL statements - arithmetic logical and conditional statements are executed directly from left to right, reducing program complexity and reducing the number of set/saves by 50 percent, which reduces the program size.

4. Macro-instruction repertoire - permits specification of complete trigonometric, logorithmic, complex, vector, matrix and list operations in one or two macro-commands. This will result in improved compatibility with HOL and minimize program storage, as well as, allowing improved computer operating efficiency.

5. Many special vector and matrix operators.

6. Very powerful data manipulating instructions.

7. Real-time executive - structures Program Modules in real-time and on-line, minimizes need for extensive software integration and permits dynamic software reconfiguration.

8. Instruction look-ahead - improves processor throughput by a factor of 2 by decoding and executing instructions concurrently.

9. High speed - the AADC PE provides the following sequential throughput capability:

   - 4 MIPS with 30:70 instruction mix, all floating point arithmetic and 10 nsec off-the-shelf technology.

   - 8 MIPS as above, except with 2-3 nsec AADC semiconductor technology.

- 10+ MIPS as above, when instruction handling capability is considered.

For more information on the DPE instruction repertoire see Chapter 5.

9.2.4 HOL Metacompiling

The metacompiling technique allows a single compiler to be used for many object computers as shown in Figure 9.3.

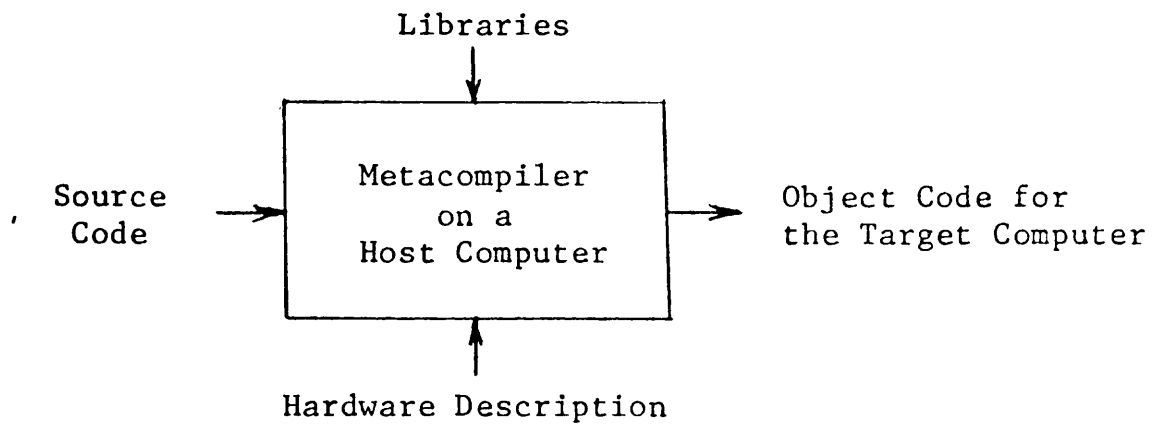


Figure 9.3 Use of a Metacompiler

Source code, such as an application program and a description of the target computer hardware are fed into the Metacompiler on the host computer. The source code calls any procedures or routines it needs from the library. From these input, the metacompiler generates object code for direct execution on the target computer.

Figure 9.4 shows a simple block diagram for using a metacompiler. First the Statement of Requirements (SOR) is fed into the task-load estimation block, which can refer to the algorithm bank to simplify its estimations. The output from task-load estimation block is fed into the hardware definition block and into the operational program block. The hardware definition block then selects hardware modules and options from the set of available modules and feeds its output to the metacompiler. On the other path, the

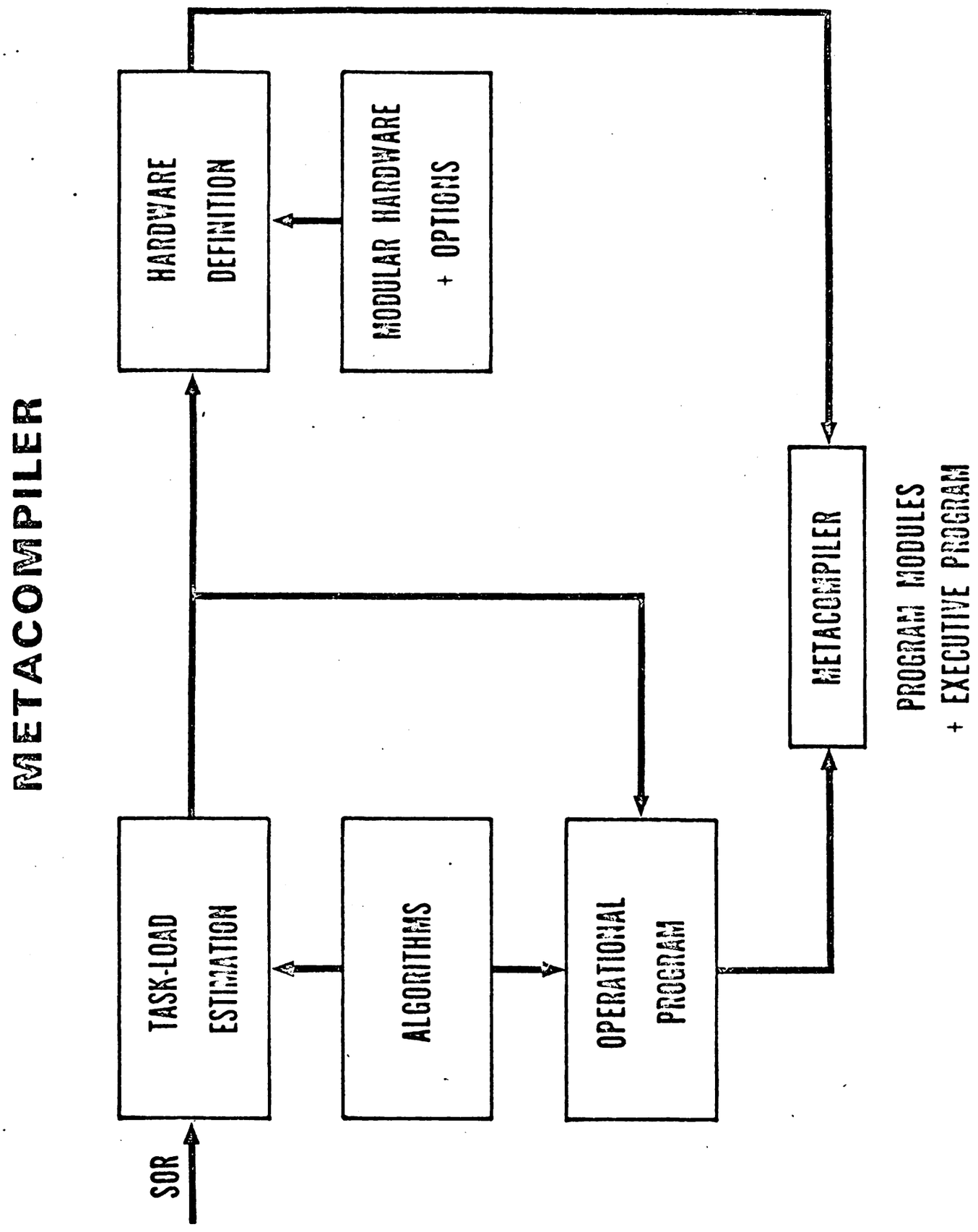


Figure 9.4. Block Diagram of a Metacompiler

operational program block uses the task-load estimation to select algorithms which are then processed and fed to the metacompiler. From these inputs the metacompiler generates the program modules and executive programs for the target computerin such a way that they will staisfy the Statement of Requirements. Significant saving can be obtained by using metacompilers for a HOL. For further information on the metacompiler technique see [9.4 and 9.5].

9.2.5 Software Cost Reduction with HOL

The following cost saving will be obtained with the AADC HOL:

1. Single High Order Language - with hardware and software compatibility.
2. Ability to document programs - in the single HOL.
3. Available Program Modules - in algorithm bank.
4. Simplified training - for Navy and Contractors.
5. Available supporting software.

9.2.6 Comparison of CMS-2 to Other Programming Languages

This subsection compares CMS-2 (Compiler Monitor System) with other "high level" languages. First Appendix 9.1* gives an overview of the CMS-2 language. CMS-2 is a "high level" statement-oriented computer language similar to JOVIAL, Fortran and PL/I. References [9.7 to 9.11] provide more information on the CMS-2 language. Reference [9.10] concentrates on the utility of CMS-2 statements - which ones are essential, which are redundant and which ones are difficult to implement. Reference [9.11] is the CMS-2 programming manual, of which Volume 2 is of the most interest since it describes the language.

*Not included at this time.

Table 9.1 is a comparison of CMS-2 with JOVIAL, Fortran, APL and PL/I programming languages [9.6, pages 54, 55 - Enclosure 3]. As can be seen, CMS-2 has several features that are missing in the other languages. The features of particular interest are the system, the data structure and the input/output features. One CMS-2 feature that is of dubious value is the ability to intermix machine code with CMS-2 statements. Although this is very desirable when the speed or powerfulness of machine code is required, it has perpetuated the use of machine code when it was not really necessary, thus eliminating any hopes of producing transferable programs in the high level language. One obvious shortcoming is that CMS-2 does not have the powerful vector and matrix operations that are contained in APL.

9.2.7 Goals of the AADC HOL

At this time the goals of the AADC High Order Language have not been specified, but a project is currently underway to make such a specification of goals. See Section 9.4 for further details.

Table 9.1. Comparison of CMS-2 to Other Programming Language

FEATURE	CMS-2	JOVIAL	FORTRAN	APL	PL-I
Input/Output					
Can describe input/output devices?	Yes	Yes	No	No	No[1]
Allows Extensive formatting of data?	Yes	No	Yes	No	Yes
Allows tape control functions?	Yes	Yes	No	No	Yes
Range of automatic output conversions?	Yes	Yes	Yes	No	Yes
Stream and record processing?	Yes	Yes	No	No	Yes
Miscellaneous					
Arithmetic expressions in subscript?	Yes	Yes	Yes	Yes	Yes
Addition of subroutines, procedures?	Yes	Yes	Yes	Yes	Yes
Linkage transmission of name or value data?	Yes	Yes	No	Yes	Yes
Mixed arithmetic expressions?	Yes	Yes	Yes	Yes	Yes
Manipulation of bits of data?	Yes	Yes	No	Limited	Yes
Manipulation of characters of data?	Yes	Yes	No	Yes	Yes
Initialization of data?	Yes	Yes	Yes	Yes	Yes
Packing of part-word data values?	Yes	Yes	No	No	Yes
Scaling specified or automatic scaling ?	Yes	No	No	Yes	Yes
Capability to do array manipulations with single reference?	Limited	No	No	Yes	No
Built in collection of subroutines for common mathematical functions?	Limited	Yes	Yes	Yes	Yes
Provide intermixing of machine code?	Yes	Limited	No	No	No[2]
Control over use of machine code?	No	No	No	No	No
Provision for jump tables?	Yes	Yes	Yes	Yes	Yes
Allows user-index register assignment?	Yes	No	No	No	No[4]
Full character set?	Yes	Yes	Yes	Yes	Yes
Data Types					
Integer, floating point, literals, Boolean?	Yes	Yes	Yes	Yes	Yes[3]
Typed Pointers?	No	No	No	No	No
Status variables?	Yes	Yes	No	No	No
Complex numbers?	No	No	Yes	Yes	Yes
Double precision floating point?	No	No	Yes	Yes	Yes
Complete part-word data elements?	Yes	Yes	Yes	No	Yes
Multiword data elements?	Yes	Yes	No	Yes	Yes
Character strings?	Yes	Yes	No	Not Efficiently	Yes
Internal Process Operators					
Basic logical operators?	Yes	Yes	Yes	Yes	Yes
Relational operators?	Yes	Yes	Yes	Yes	Yes
Standard mathematical interpretation?	Yes	Yes	Yes	Yes	Yes
Automatic table searching?	Yes	No	No	Yes	No
Boolean algebra?	Yes	Yes	Yes	Yes	Yes

Table 9.1. (Cont'd).

FEATURE	CMS-2	JOVIAL	FORTRAN	APL	PL-I
Looping Operations					
Allows looping within preset range?	Yes	Yes	Yes	Yes*	Yes
Allows nested loops?	Yes	Yes	Yes	Yes	Yes
Allows incrementing by preset values?	Yes	Yes	Yes	Yes	Yes
Allows alternate transfer points?	Yes	Yes	Yes	Yes	Yes
Decision Making					
IF Statements?	Yes	Yes	Yes	Yes	Yes
Compound IF statements?	Yes	Yes	No	Yes	Yes
Alternative statements?	No	Yes	No	Yes	Yes
Data Structure					
Control source of implied data description?	Yes	Yes	No	Fair	No[5]
Arrays with simple elements?	Yes	Yes	Yes	Yes	Yes
Arrays with compound elements?	Yes	No	No	No	Yes
Variable-length tables?	Yes	Limited	No	Yes	Yes
Variable-size arrays at run time?	No	No	Yes	Yes	Yes[3]
Horizontal or vertical tables?	Yes	Yes	No	Limited	Yes
Provides for local and global structures?	Yes	Yes	No	Yes	Yes
Allocation					
Dynamic storage allocation on procedure entrance?	No	No	No	Yes	Yes
Data-element equivalencing?	Yes	Yes	Yes	Yes	Yes
Express relative origin of data values?	Yes	Yes	Yes	Yes	Yes
Can define structures over structures dynamically?	Yes	No	No	No	Yes[7]
Define absolute allocation?	Yes	Yes	No	No	No
Allows declaratives defined where inserted?	Yes	Yes	Yes	Yes	Yes
System Features					
Source language debug capability?	Limited	No	Limited	Fair	Limited
Selective listings?	Yes	No	No	No	Yes
Object library provision?	Yes	Yes	Yes	Yes	Yes
Flexible library handling in language?	Limited	No	No	Limited	Yes

NOTES:

1. Provided by operating system
2. Allowed by the PL/1 language, but not yet implemented.
3. Easily constructible in the language.
4. Not pertinent to a high-level language.
5. Feature undefined.
6. "Include" facility has some of this feature.
7. Available in some implementations.

---

* Due to its parallelism, loops are often not used in APL algorithms.

** Can list lines of a subroutine but not parts, or all, of several routines.

9.3 EXTENDING CMS-2 TO AADC's HOL

This section contains references to projects for determining the practicability of extending CMS-2 to becomes the AADC HOL (sometimes called CMS-3 or CMS-TOO).

Reference [9.7] is the proceeding of the first High Level Aerospace Computer Programming Language Conference held in June 1970 and discusses some of the general problems of the AADC HOL definition. (The second such conference is presented in the next section.) Two relatively old (1970) references on using CMS-2 for existing avionic applications and on implementing CMS-2 on the AADC are shown in [9.12 and 9.13] respectively.

Reference [9.14] describes the Marine Tactical Air Command and Control System (MTACCS) requirements on the CMS-3 (extended CMS-2) language specifications. The recommendation are that CMS-2 must be stronger in two areas:

1. Data base definition and handling
   - Multiple COMPOOL Core Definition (COMPOOL are compiled procedures that can be combined with other procedures without being recompiled.)
   - COMPOOL defined mass storage file definitions
   - Conversion of core/mass storage formats (i.e., with simply a Move operator)
   - Data conversion operators.
2. Character String Capability.
   - Encode/decode extensions
   - Insert/delete/concatenate operators
   - Decimal arithmetic

A more recent report studying the problems of CMS-2 transferability from AN/UYK-7 to AADC suggests that system designers and programmers can strongly influence the transferability of software. The report makes several specific suggestions that should be of general interest to system designers and programmers [9.15].

The idea, of allowing the applications to influence the programming language which in turn influences the computer software and hardware, is indeed a good one and should be given full support. It is time for computer specialists to start looking at the application areas first, then designing the languages to solve these applications and finally designing the software and hardware to implement the "user-oriented" languages. The traditional reverse ordering was based on economics and levels of knowledge which are no longer valid.

9.4 CURRENT STATUS OF HOL

The following excerpt is taken from AADC Progress Report No. 9 dated November 1971 [9.6]:

> 21. AADC Progress Report No. 7 [9.16] contained a preliminary statement of Work for a Request for Proposal to develop improvements to the Navy's existing CMS-2 program language. The purpose of these improvements was to allow efficient expression and, hence, economical compilation in the areas of:
>
> - applications
> - executive/operating system
> - input/output
> - test
> - display
> - array processing.
>
> 22. The improved language also provided means to express data in ways that allow universal interpretation. Such data description techniques would permit the future integration of large data bases by allowing ready communications among systems programmed in different languages.
>
> 23. Enclosures (1) and (2) present the latest Navy thinking on the subject. Present plans call for the release of a final form of the Statement of Work as an RFP within the next few months, preceded by a conference to be held at the Naval Electronic Laboratory Center, San Diego early in January (actually February) '72. Questions concerning the conference should be directed to Mr. Warren Loper, Code 5200, NELC, San Diego, Calif.

Thus, AADC progress report No. 9, pages 13-58 include "A Statement of Work of a Plan to Define HOL Primitives for AADC Computer - Preliminary" (pages 13-31), "Goals of the Language" (pages 32-46), and "Document Support Request for Approval for RFP for HOL Study" (pages 47-58). The objective of the Work Plan and the Goals of Language are attached as Appendices 9.3 and 9.4 [9.6].

The following excerpt is taken from the AADC Progress Report No. 10 dated May 31, 1972 [9.1.7], and reflects the May 1972 thinking on the HOL:

> 11. Improvements to CMS-2: An AADC Software Conference on Command Control Software Technology for 1975-1985, cosponsored by NAVAIRSYSCOM and the Naval Electronic Laboratory Center, was held at the LeBaron Hotel in San Diego on 15 - 17 February 1972. The stated purpose of the conference was to "address the questions of requirements that will be imposed (on software systems) and the methodologies that will be available (to satisfy these requirements in the 1975-1985 time frame)"*. An unstated purpose of the conference was to expose an important segment of the software community to the hardware and architectural concepts embraced by AADC. In addition, the conference provided an opportunity to openly discuss the goals of the AADC software effort and, in particular, the programming language development utilizing the existing CMS-2 language as a basis.
>
> 12. A preliminary statement of work for the language RFP was enclosed with the ninth AADC Progress Report. The Proceedings of the Second AADC Software Conference is now in preparation. Proceedings of the first such conference held at the Naval Research Laboratory on 29 - 30 June 1970 is available from NTIS, Springfield, Va. 22151.

Based on the philosophy that a "Universal computer language" will still fail because a specialized language is always better for specialized applications, the AADC program is now in the process of developing a single kernal language with potential for extensions. The advantages of this approach are:

- specialized application-oriented languages can be obtained as extensions to the kernel
- a single language structure
- improved adaptibility to "unpredictable" requirements

---

* From the Official Program for the AADC Software Conference On Command Control Software Technology for 1975-1985; NELC; 15-17 February 1972.

Thus a kernal language called CMS-2K is now being developed for AADC. It has definitional facilities, operators for arrays, block structure, fixed lexical structure, built-in data element primitives, a very flexible expression format, etc.

The first language to be developed from the CMS-2K (the kernel) is called CMS-2R. It is intended as a replacement for CMS-2, but may not be upward compatible with it. The language will contain string, matrix, vector and complex operators, as well as, many operating system support functions.

The current contracts are to develop the CMS-2K and CMS-2R languages but do not include implementing these languages [9.18]. A student thesis project is now underway at NPS to implement a CMS-2 compiler on the IBM 360/67 computer.

9.5 HOL PROJECTS

There is a need for a study to define the desirable HOL constructs and to determine the feasibility of implementing them in the AADC (or for modifying AADC to accomodate them). Some steps are:

1. Define the HOL constructs that would simplify the writing, debugging, documenting and updating of real-time, scientific and data processing application programs.

2. Repeat Step 1 above for each of executive, I/O, test, display and data reorganization types of programs.

3. Determine the feasibility of implementing the HOL constructs identified above on AADC, i.e., estimate the cost of implementing each feature.

4. Select a minimal set of constructs that satisfy all the requirements in 1 and 2 above and can be effectively implemented in AADC.

5. Expand Step 4 to include other desirable constructs and features and determine the incremental cost of implementing these.

6. Determine how effectively CMS-2 meets the requirements identified in Step 1 and 2 above. Some of this work has already been done - See Section 9.3 and [9.8 to 9.12].

7. Determine the cost of making the AADC HOL upward compatible with CMS-2.

8. Implement the CMS-2K compiler on a computer.

9. Determine the suitability of CMS-2K as a kernal for implementing CMS-2, APL, Fortran, COBOL, JOVIAL, etc.

10. Using CMS-2K, define and implement languages that are as close as possible to CMS-2, APL, Fortran, COBOL, JOVIAL, etc.

11. In each case in item 10 above, develop a translator from the parent language to the new language.

9.6 A HOL FOR SIGNAL PROCESSING

Although the previous sections have addressed the problem of developing a HOL for sequential processing (for the Data Processing Element), there is also a need for a High Order Languages designed specifically for signal processing and for the executive system. These HOL languages would be used to program the Signal Processing Element (Chapter 7) and the Master Executive Control (Chapter 6).

The programming languages that are needed for signal processing can be divided into three areas;

1. Support software for program development including a microprogramming language.

2. Executive software for the Microprogrammed Control Unit (MCU).

3. High level languages for signal processing applications.

Some work has already been completed on developing support software for program development. The AMIL (A Microprogramming Language) and its translator have been developed and are now operational. AMIL is a Fortran-like language for specifying microprograms for the Microprogrammed Control Unit of the SPE. Since AMIL eliminates the need for specifying bit patterns, it can be considered a high order microprogramming language. The basic support software that has been prepared is scheduled for release in April 1973 to anyone who wants it.

The development of executive software for the Signal Processing Element is now beginning without the aid of a HOL with executive defining capabilities. The Master Executive Control (MEC) for AADC has also been

developed without such a defined HOL. Also no work has yet been done on developing a HOL for signal processing applications. Thus the development of High Order Languages for executive systems and signal processing applications is still an active area for further research and development.

References to HOL

9.1 Supplementary Information Regarding the Determination and Specification of the Preliminary Instruction Repertoire for the AADC (U); System Consultants Inc.; February 27, 1970; NAVAIRDEVCEN Contract N62269-69-C-0574; Confidential-NOFORN-Proprietary; (23).

9.2 Report on the Determination and Specification of the Preliminary Instruction Repertoire for the AADC; System Consultants Inc.; February 27, 1970; NAVAIR-DEVCEN Contract N62269-69-C-0574; Unclassified; AD-867-055; (24, NPS)*.

9.3 Slide Presentation on AADC for FY72; R. S. Entner, NAVAIRSYSCOM; Undated, probably spring 1972; Unpublished; Unclassified (NPS).

9.4 Compiler Construction for Digital Computers; David Gries; John Wiley & Sons, Inc.; New York; 1971.

9.5 A Compiler Generator; W. M. McKeeman, J. J. Horning, D. B. Wortman; Prentice Hall; 1970; (NPS).

9.6 AADC Development Project Progress Report No. 9; R. S. Entner; NAVAIRSYSCOM: November 1, 1971; (67, NPS).

9.7 High Level Aerospace Computer Programming Language Conference Proceedings of 29 - 30 June 1970; R. S. Entner; NAVAIRSYSCOM; Unclassified; AD-733-454; (34, NPS).

9.8 A Technical Overview of Compiler Monitor System 2; John P. O'Brien; Computer Science Corporation Document CSC-STD70-009; Proceedings of the High Level Aerospace Computer Programming Language Conference; June 1970; pages 111-150; Available in [9.7]; (34, NPS).

*AADC Bibliography number, and availability at the Naval Postgraduate School.

9.9 CMS-2 Compiler Design; Systems Consultants Inc.; Op Cite; pages 202-240; (34, NPS).

9.10 Analysis of the CMS-2 Programming Language; Raytheon Corp.; Report No. BR-6704; December 1, 1971; Unclassified; (72, NPS).

9.11 Compiler Monitor System-2; Volumes I to IV; M-5012, Fleet Computer Programming Center, San Diego, California; June 1969.

9.12 Final Report on the CMS-2 Computer System, Part I, Evaluation of the CMS-2 Compiler Language for Existing Avionic System Application (U); Systems Consultants Inc.; October 13, 1970; NAVAIRDEVCEN Contract N62269-70-C-274; Confidential-NOFORN; AD-513-557; (40).

9.13 Final Report on the CMS-2 Compiler System Part II, Implementing the CMS-2 Compiler on the Advanced Avionics Digital Computer; System Consultants Inc.; October 13, 1970; NAVAIRDEVCEN Contract N-62260-70-C-0274; Unclassified-NOFORN; (41).

9.14 Influence of MTACCS Requirements on CMS-3 Language Specification; Hughes Aircraft Co.; NAVELECSYSCOM Contract No. N00039-70-C-3552; January 21, 1972; (74, NPS).

9.15 CMS-2 Software Transferability Study, AN/UYK-7 to AADC - Guidelines for system designers and programmers developing functional programs; N. S. Mathis; NELC Technical Document 207; November 1972.

9.16 AADC Development - Program Progress Report No. 7; R. S. Entner, NAVAIR-SYSCOM; February 4, 1971; AD-727-605; (57, NPS).

9.17 AADC Development - Program Progress Report No. 10; R. S. Entner, NAVAIR-SYSCOM; May 31, 1972; (78, NPS).

9.18 All Applications Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceeding are not yet available.

Appendix 9.1*

An Overview

of the

CMS-2 LANGUAGE

(Compiler Monitor System)

*This appendix is not yet available. In the meantime References [9.7 to 9.11] will have to suffice.

Appendix 9.2

to

AADC Course Notes

Objectives of the Work Plan to Define HOL Primitives for AADC*

1. OBJECTIVE:

The objective of this task is to identify, define and prepare a plan for the implementation of the revisions to the Compiler Monitor System II (CMS-2) language needed to support the effective use of the AADC (Advanced Avionics Digital Computer) [1,2] in a broad spectrum of military applications including the ITACS (Integrated Tactical Air Control System) [3] and the MTACCS (Marine Corps Tactical Command and Control System) [4].

The emphasis given the various goals of the language in Exhibit A is impacted by the requirements (and opportunities) of a real-time environment and, predominately, by military requirements. Thus, not only present but also expected avionics, command and control, intelligence and other military requirements of the language must be identified and correlated. Inconsistencies among the goals of the language must be recognized and documented, permitting a selection of an "optimum" set (of compromises) consistent with military requirements. Then these requirements and their related language goals must be compared with the Navy's standard programming language, CMS-2 [5,6] to identify and fully document:

(1) Inconsistencies between the requirements and their above related goals on the one hand and CMS-2 on the other (mutual exclusion problems), and,

(2) Revisions to CMS-2 necessary to meet these requirements and goals.

---

*Taken from "A Statement of Work of a Plan to Define High Order Language Primitives for the AADC Computer - Preliminary", Enclosure (1) to AADC Progress Report No. 9 [9.6, page 15].

Appendix 9.3

Requirement for Specifying the Goals of AADC's HOL*

GOALS OF THE LANGUAGE

A language specification effort must be oriented to a set of specific goals and a method of measuring quality in terms of these goals. Two methods for accomplishing this are available and both should be used. The first method is a stated list of criteria which the language is expected to meet. Many of the criteria are obvious but should be listed to insure their consideration during the design. The second method is sample programming. Such samples subject a language concept to the ultimate measure of utility and will often override intuitive conclusions. The samples need not all be large; often short fragments serve the desired purpose. A few reasonably lengthy attempts are necessary, however, to determine that no problems of clarity will exist in practice. Then too, samples can be used in tutorial documentation as examples of style.

The goals for a tactical systems language must express the nature of the programs to be written in the language. Traditionally, a tactical data system has been distinguished by a heavy reliance on fixed-point computations involving quantities of relatively low precision, a need to pack data as tightly as possibly into the small available storage, specialized but relatively simple input/output operations, and an executive system integrated with the program almost to the point of being indistinguishable from it. It is expected that the spectrum of applications of AADC programmable hardware will contain applications of this character. Avionics applications will continue to impose on the computer limitations of weight size, heat generation, etc., implying programming constraints unnecessary in a large computer center. However, a broadening of the types of computation to be performed can be observed in such areas as digital fire control and the Marine Corps Tactical Command and Control System (MTACCS) [1] where greater emphasis has been placed on computation accuracy and speed; alphanumeric and graphic information input, processing, storage and display; message routing; and data base management. Further, experience over the last few years has shown that there is a small yet significant number of ancillary computational chores to be done in a Computer Programming Center that are not limited to the scope of a tactical data system. Shipboard Naval Intelligence processing systems provide one example introducing requirements for multilevel security. There is also the attractive possibility of sharing shipboard equipment for other purposes during noncombat conditions (provided that its primary mission is not compromised). All this suggests that a "tactical systems language" should be capable of handling more than pure tactical data systems.

It would be a serious error to reach for some sort of universal language. Such attempts in the past have not met with any particular success and we can have little confidence of doing noticeably better now. After all, the primary purpose of the language is to express algorithms having the characteristics previously discussed. Extensions and generalizations are welcomed provided they do not dilute ability to meet the central requirements. Within the language, dynamic extensibility will solve many problems. Outside the language, an interactive facility for defining the syntax and semantics of new problem-oriented languages and generating their processors will solve others.

---

*Taken from Exhibit A to "A Statement of Work of a Plan to Define High Order Language Primitives for the AADC Computer - Preliminary", Enclosure (1) to AADC Progress Report No. 9 [9.6, pages 32,33].

A key factor in programmer efficiency is the number of basic tools available for his use. Does the language (or do the languages) contain the features that allow an easy expression of the problem? If the program under development interfaces with other systems or equipment not currently available, the programmer should have a means of easily simulating the missing pieces. If the programmer is working on problems that have large documentation requirements, he should have an automated means of generating and updating all the documentation (not merely flowcharts).

Another consideration is the total systems environment. This should provide the programmer or group of programmers ready access to the equipment and provide a means of communication between not only the programmer and the machine, but also between programmer and programmer. The programmer-machine interface is again dependent on the above factors whether on-line consoles or remote batch processing systems are provided. One of the NTDS problems is the programmer to programmer interface, where eastern seaboard programmers require western seaboard information, and vice versa. With today's technology, it is feasible to provide a common data-base for NTDS modules which is accessible from remote locations.

It is desirable for the language to allow the programmer to insert additional information "for the possible benefit of the translator" [2:p.38] and to provide the means to specify optimization techniques to a compiler or specify the degree of various types of optimization to be performed by the compiler. Also, certain facilities of the language may be parametrically inhibited to prohibit use of these facilities in the source language of certain modules. For example, one may lock out dynamic storage allocation and free space management when programming for a small hardware configuration.

Relevant tools of software engineering [3] technology should not be ignored in development of the language. For example, in support of the design of specifications for problem-oriented languages as well as the procedure-oriented system programming language, automated syntax completeness, consistency and ambiguity analysis is now possible.

Chapter 10

A P P L I C A T I O N S

F O R

A A D C

Table of Contents for AADC Applications

List of Figures

Chapter 10

APPLICATIONS FOR AADC

10.1 INTRODUCTION AND SUMMARY

Although this is the most important chapter in the report, it is, unfortunately, one of the shortest. Never before has the Navy known so far in advance what the future Navy computers will be, and now the Navy has an opportunity to develop application programs while the computer is being developed, instead of after it is produced and delivered. Equally important, the Navy now has the opportunity of allowing the applications to influence the software design, which in turn can influence the hardware design. If the Navy can develop an applications-oriented computer and have the application programs ready when the hardware is delivered, the Navy will have made another major step in solving its computer oriented problems.

This section presents references to an E-2B aircraft simulation study, the requirements for MINCOMS (Multiple Interior Communication Systems for aircraft), and the On-board checkout and system interface requirements for the F-14C. Also presented is the proposed Automated Design Facility (ADF) which is designed to provide automatic configuration and checkout of AADC for a new application.

10.2 POSSIBLE AADC APPLICATIONS

10.2.1 Avionic Applications

Certainly the most important application for the AADC, and the reason for initiating the AADC project as the Advanced Avionics Digital Computer, is the future avionics computer applications.

Some of the proposed 1975-1985 avionics computer tasks are:

- Navigation
- Weapon Delivery
- Sensor Monitor and Control
- Radar Signal Processing
- Acoustic Signal Processing
- Target Signature Recognition
- Target Tracking
- Sensor Correlation
- Data Compression
- Countermeasure Monitor and Control
- Communication Format and Control
- On-board Checkout
- Automatic Flight Control
- Display Signal Format and Control
- Environmental Control

Figure 10.1 gives some other examples of avionic tasks for AADC showing a typical air-to-ground avionics system (taken from [10.1]). Figure 10.2 shows how the AADC may be interfaced to the aircraft via MINCOMS (multiple Interior Communication Systems) and to several other systems including ITACS (Integrated

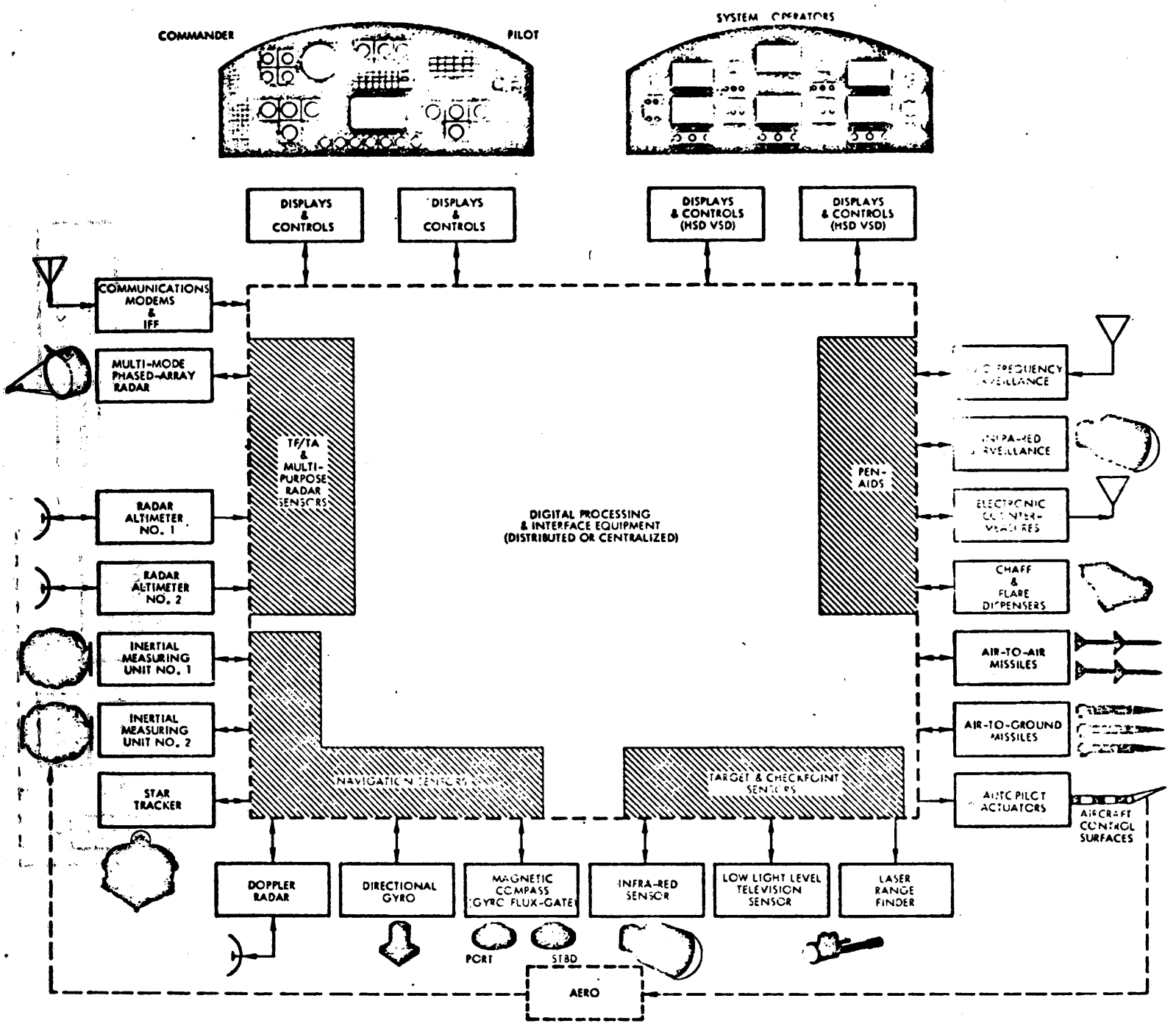


Figure 1 – Air-to-Ground Avionic System – Typical Configuration

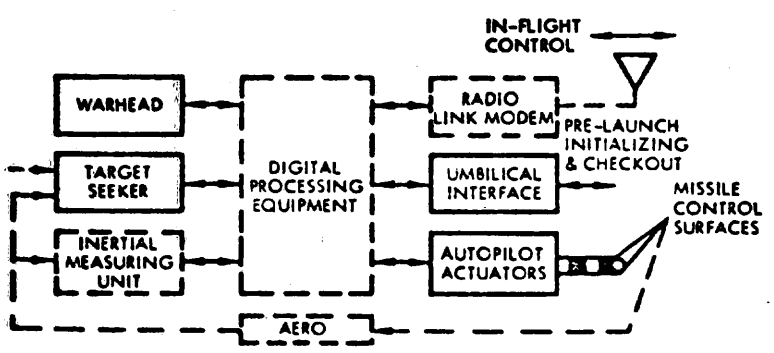


Figure 2 – Missile Guidance and Control System

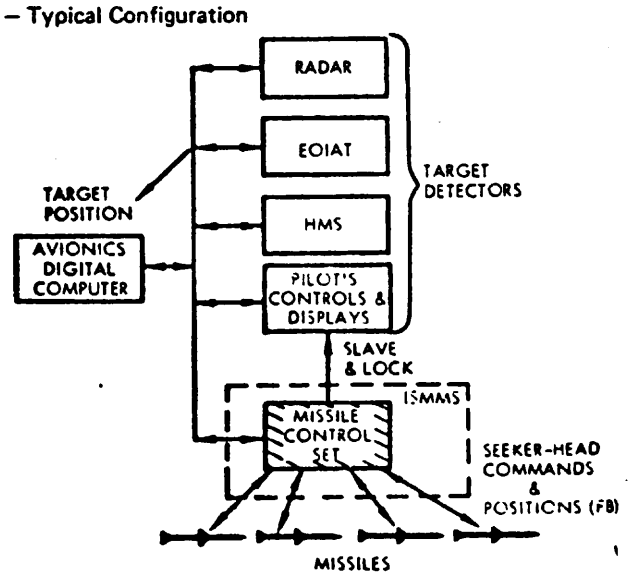


Figure 3 – Missile Control Set – System Configuration

Figure 10.1. Avionic Tasks for AADC

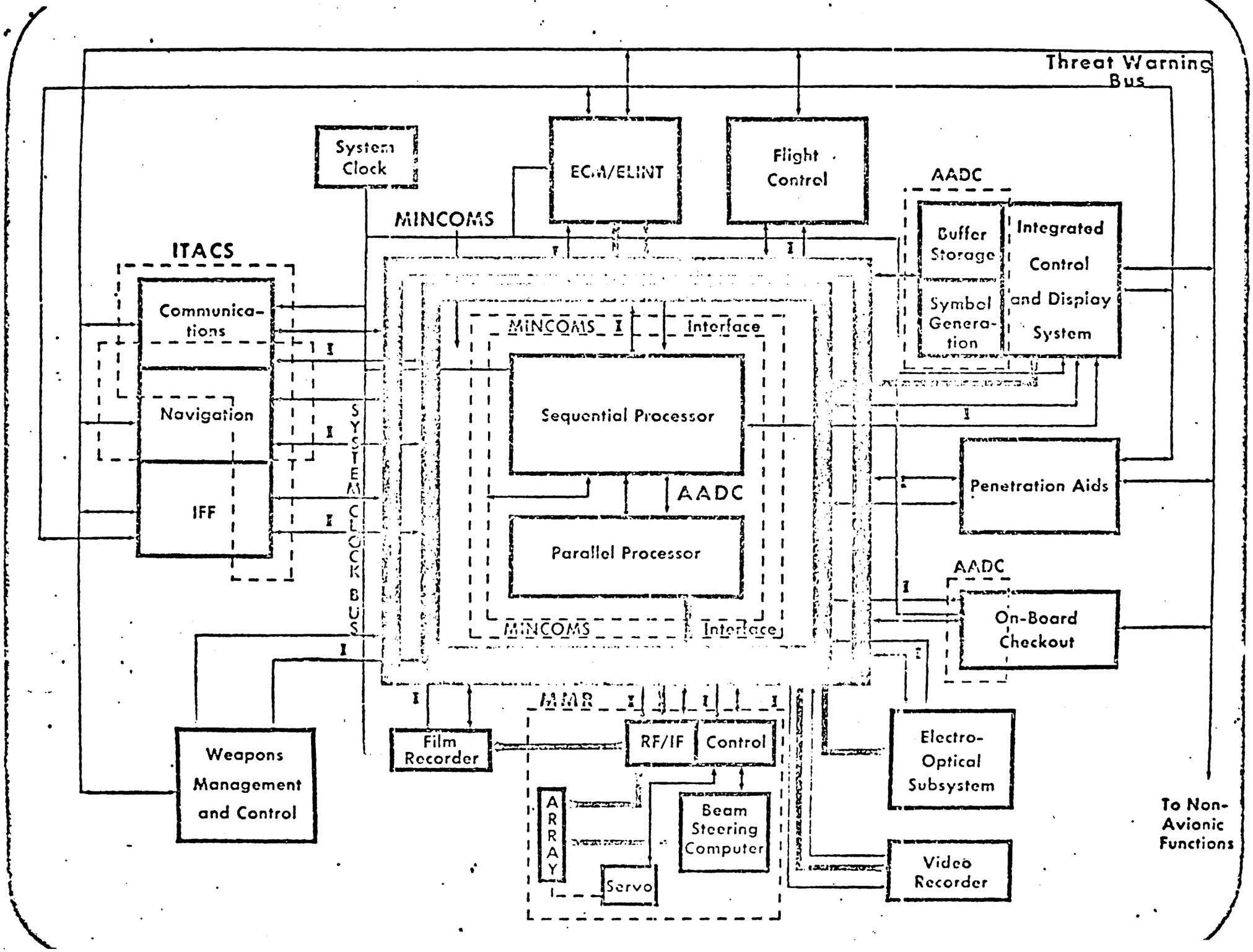


Figure 10.2. Typical AADC Avionic Application

Tactical Air Control System), ECM (Electronic Counter Measures) and the MMR (Multiple Mode Phased-Array Radar). The dotted rectangles in the figure represent the functions handled by AADC. For background information on MINCOMS see [10.2 and 10.3].

Two reports are available concerning the simulation of the E-2B aircraft. The first defines the E-2B digital characteristics for the purpose of the simulation and is classified secret [10.4]. The second report describes the simulation which is concerned mainly with the optimum block (page) size for BORAM and the Optimum Task Memory size. The study does not address the many other problems of using the AADC on the E-2B aircraft [10.5].

A much more general simulation, which is a continuation of the above, is reported in [10.6]. This report describes the simulation of the AADC simplex and multiprocessor operation on three avionic workloads - the E-2B, the F-111 and future avionic requirements as defined by a GE report. The major conclusion from the simulation is that the AADC Simplex configuration can handle all of these avionic workloads. For further information see subsection 8.2.4 or [10.6].

Four volumes of a report by Grumman Aerospace Corporation on the On-board check-out and system interface requirements for the F14-C aircraft are available in [10.7 - 10.10]. According to AADC Progress Report 10 dated May 31, 1972, the AADC program is sponsoring two studies with Grumman Aerospace and LTV Corporations to determine the computer requirements for future F-14 and A-7 class aircraft. From these studies it is hoped to predict other future Naval ADP requirements.

Reference [10.11] is a Grumman Aerospace Corporation report on the AADC interface requirements for a representative F-14C avionic weapon system. The primary goal of the study was to provide detailed definition of the interface of future aircraft systems to AADC for operational and checkout functions. The report is divided into 3 distinct tasks:

1. Task 1 defined thirteen subsystems for the F-14A aircraft and determined that the total interface requirements between the subsystems and AADC would be less than 200,000 bits/sec.
2. Task 2 defined the input/output requirements of the AADC configured to meet the requirements of the F-14C baseline system.
3. Task 3 defined the functional requirements of a Data Handling System which would transfer information between the AADC I/O and the subsystems.

Some of the important conclusions and recommendations in the report are as follows:

1. Thirteen explicit subsystems on the F-14C aircraft were identified as airframe, control and display, environmental control, flight control, hydraulics, fuel, lighting, control of communication, mission and traffic, navigation, propulsion, electrical power, and finally, weapon control.
2. The total input to AADC was 79 kilobits per second and total output is 88 kilobits per second - with over half this being weapon control. Thus a 200 kilobits per second interface capability would be adequate for the F-14C.

3. The AADC configuration recommended for the F-14C is the multimemory, multiprocessor configuration with a single (4K word) Random Access Main Memory providing the interface with the input/output unit (see Figure 10.3). In this configuration all data required for operation of the subsystem is stored in the RAMM while in transit to or from the Data Processing Element of AADC. In addition to the RAMM, the recommended, AADC I/O unit includes a Bus Control Unit and a Memory Module - which stores instructions for the BCU.
4. The command/response method was recommended as the best method for the Bus Control Unit to control the Data Handling System.
5. Asynchronous I/O scheduling and double buffering is recommended.
6. The hardware MEC is recommended for the F-14C because of the implied ultra-reliability and speed advantage. Furthermore, the required reliability of the I/O suggests that the MEC and the I/O should be part of the same unit namely the AADC I/O Unit(i.e., included with the BCU, RAMM and Memory Module).
7. It was concluded that the present definition of interrupts and their relative priorities is inadequate for the F-14C mission. Accordingly, the number of interrupts should be increased from 31 to 128.

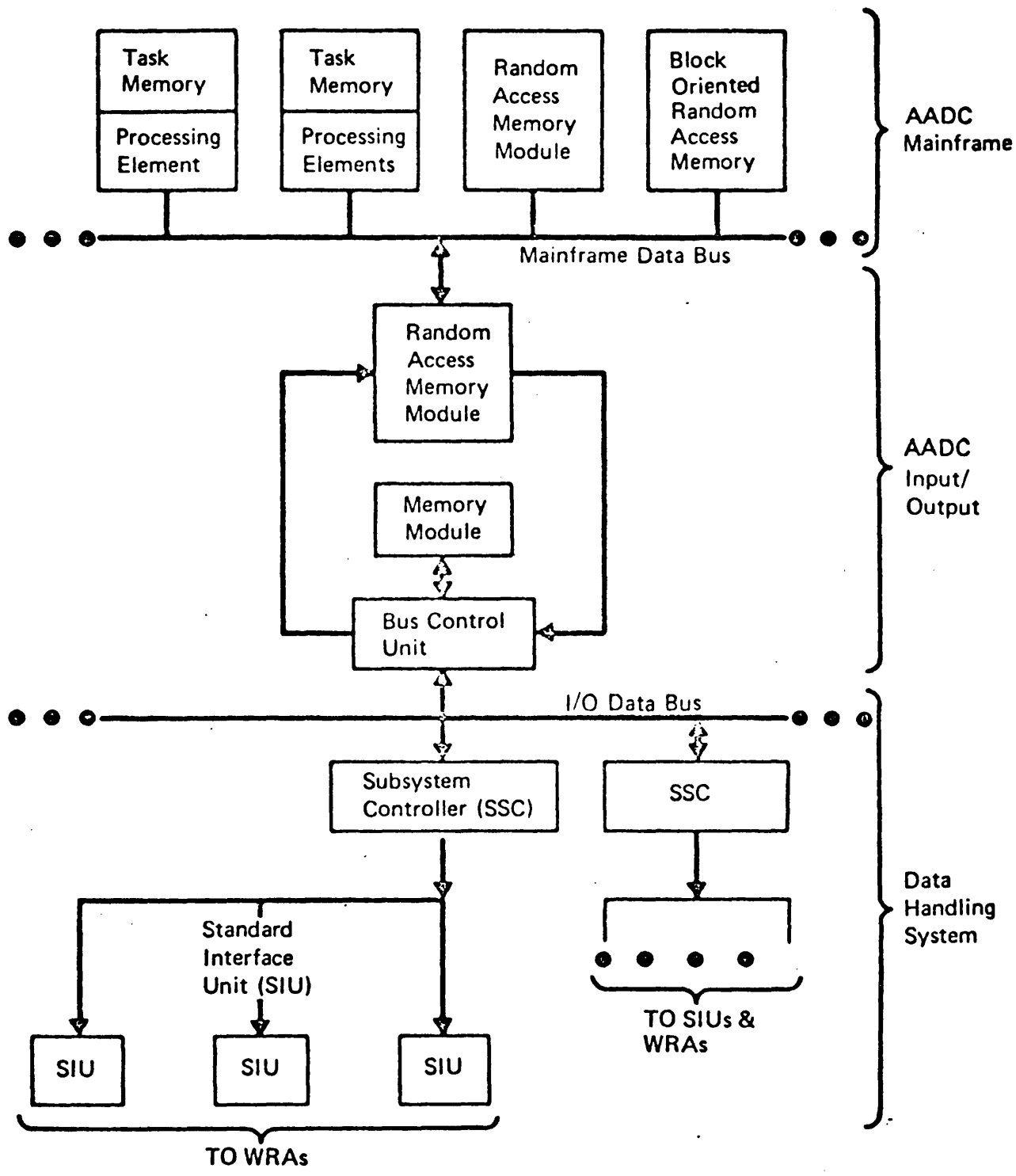


Figure 10.3 AADC Block Diagram for F-14C Aircraft

8. The Data Handling System, which interfaces to the Bus Control Unit of the AADC I/O, should be composed of Sub-System Controllers and the Standard Interface Units. The Standard Interface Units provide compatible interfaces between the SubSystem Controllers and the Subsystem Weapon Replacement Assemblies.
9. Several recommendations on the type of data transmission lines, coupling and bussing units are also included in the report.

This report is the first comprehensive report on an application for AADC and is recommended reading for all interested in AADC avionic applications [10.11].

Two other special purpose avionic applications are also being considered for AADC. These are the safety-of-flight computer and the aircraft electric power controller. For more information on the second application see Section 4.6.

10.2.2 Avionic Related Applications

This subsection is intended to describe avionic related applications, such as the modeling of aircraft systems and simulating aircraft systems in real-time. In a large simulation, an AADC may be used to interface to the real aircraft data gathering equipment, another may be used to simulate (or fake) other non-available aircraft equipment, a third may act as the aircraft safety-of-flight computer and a fourth may be the main aircraft computer. Finally, a fifth AADC may be required to coordinate the simulation and schedule events. At the predicted AADC cost, this would be quite a reasonable type of a simulation project.

Another possible use for AADC is in multiple platform systems. Bruce Wald at NRL is expected to publish a report on this in the near future [10.12].

10.2.3 NTDS and MTDS Applications

The standard computer for the Navy Tactical Data System (NTDS) and the Marine Tactical Data System (MTDS) is the AN/UYK-7 computer. Two studies have been completed to determine the transferability of AN/UYK-7 applications to the AADC. The first is the study of the compatibility of the hardware [10.13] while the second is a study recommending means of producing software transportable from AN/UYK-7 to the AADC [10.14].

10.2.4 Other Applications

Many applications have been suggested for AADC in the last year since the AADC redefinition to All Applications. These range from normal batch ADP processing to general time sharing processing and to special applications such as line concentrators, super modems, data channels and electric power controllers. They include land-based and shipboard multiprogramming and multiprocessing applications. Because of the very powerful PE, an AADC single processor system can often be used to replace a third generation multiprocessor system. No specific studies on these applications have been reported at this time.

10.3 AUTOMATED DESIGN FACILITY

Probably the most important concept in applying AADC to many different problem areas is the development of an Automated Design Facility (ADF). The ADF is intended to reduce the problems of configurating the AADC architecture, developing application programs, debugging the programs and proving the operational competence in the new application. A block diagram of the ADF is shown in Figure 10.3. Many of the blocks require considerable development. For example, the algorithm bank requires the development of the best case solution for several types of functions. Some examples of the problem oriented algorithms include ballistic trajectory prediction, maneuverable target tracking, multi-source data correlation and optimization, data compression and enhancement, display image generation and control, etc.

It is hoped that the ADF will be able to reduce the Statement Of Requirement (SOR) into useful hardware and software in a fraction of the time required by conventional procedures. In addition to compiling applications programs, the synthesizer will generate the necessary executive parameters to enable the MEC of a particular version of the AADC to schedule the execution of the problem oriented tasks. Scheduling will occur on-line and in real time [10.15, pages 21 and 29].

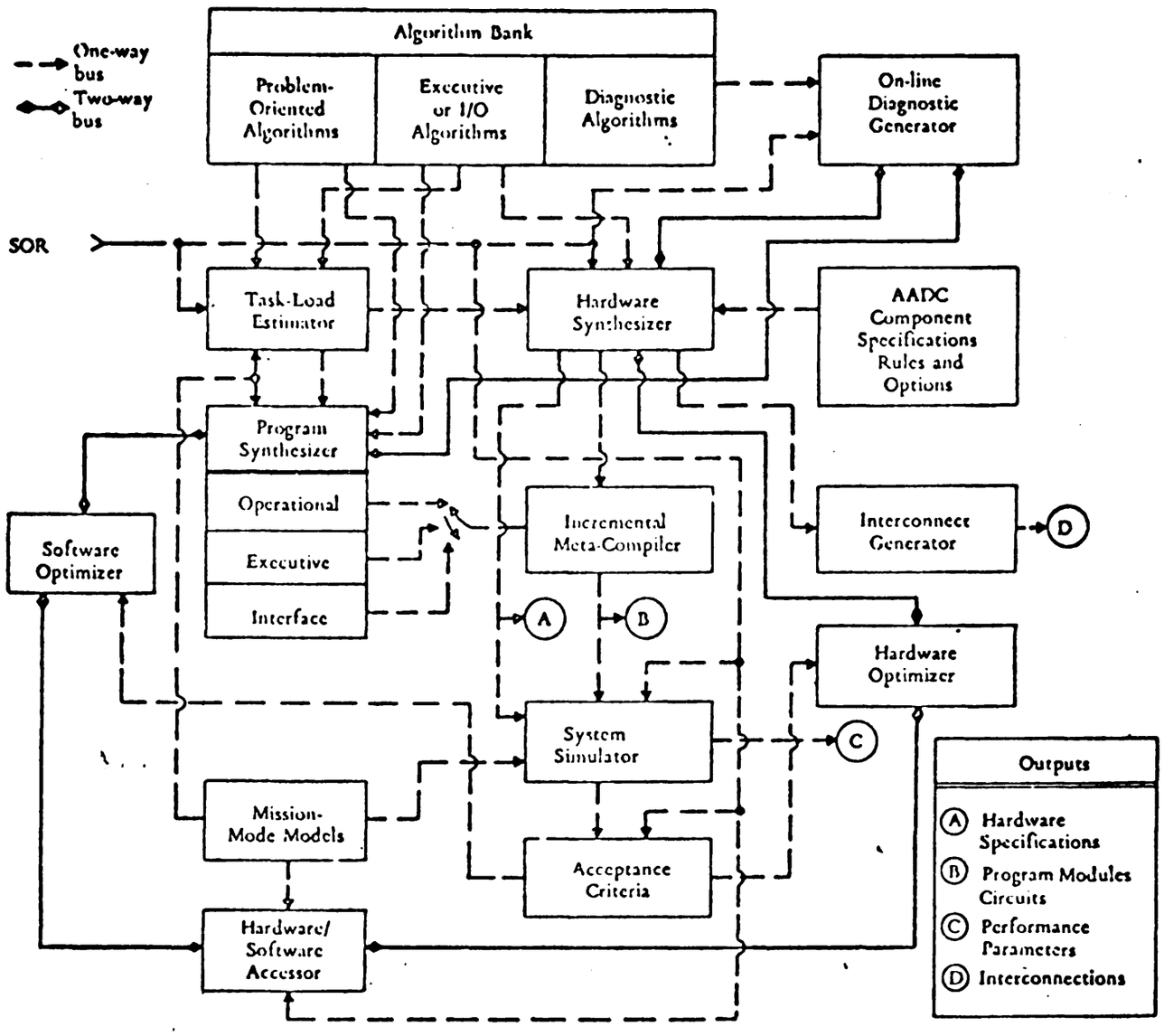


Figure 10.4. **Preliminary AADC Automated Design Facility Functional Block Diagram**

10.4 CURRENT STATUS

Mr. Hollingsworth of NADC gave an excellent presentation with some very informative slides on the probable applications of AADC. He first commented that the "AA" in AADC should stand for "Applications Assurance". In other words, the AADC proponents need to demonstrate AADC performance and strategy on specific applications; it is not sufficient to show that AADC is technically possible. Mr. Hollingsworth listed 20 aircraft and 5 ships that are in some stage of development and could be candidates for AADC. This part of the proceeding of the AADC 1973 Symposium should be very interesting reading when it becomes available [10.16].

10.5 CONCLUSIONS

This chapter has been a very brief outline of the current activities in defining applications for AADC. In fact, very little has been done in this area yet. Some studies are underway but there are many others that need to be done. Here is your opportunity to contribute to the AADC development program.

Never before has the Navy had an opportunity to develop applications while the hardware and software systems are being designed. Equally important here is an opportunity for users to define applications for AADC and thereby influence the design of the HOL, the software and the hardware for AADC. How about your input?

This report has attempted to present a study guide for AADC. It is organized in modular fashion to allow the reader to concentrate on his area of interest wihtout missing any essential background. It has covered a wide range of subjects and has undoubtedly skipped over some essential material and dwelt too long on others. (For example, some improvements for Chapter 6 are already suggested.) Also it is rather difficult to stay current when the AADC hardware and software are still undergoing further developments; and yet it is essential that the Navy begin planning and preparing for the AADC impact. One of the most important means of preparing for AADC is to inform and educate the Navy and Industry personnel on the AADC developments and capabilities.

Finally, this is actually a draft report and any suggestions concerning connection, omissions or recommended deletions will be kindly received and appreciated. Updates and corrections to the report will undoubtedly be required as the design and development of the All Application Digital Computer continues at an ever increasing pace.

References to AADC Applications

10.1 A Universal Function Unit for Avionics and Missile Systems; Frank J. Langley, Raytheon Co.; NAECON '71 Record; May 17-19, 1971; pp 178-185; Available from IEEE Transaction on Aerospace and Electronic Systems, Reference publication 71-C-24 AES; (54, NPS).*

10.2 Naval Air Systems Requirements, MINCOMS (Multiple Interior Communications Systems), General Requirements for; NAVAIRSYSCOM Document No. AR-63; March 2, 1970; Unclassified; (26).

10.3 Naval Air Systems Requirements, MINCOMS (Multiple Interior Communications Systems), Terms and Definitions for; NAVAIRSYSCOM Document No. AR-64; March 2, 1970; Unclassified; (27).

10.4 Defining the E-2B Digital Avionics Characteristics for the Simulation of Alternative AADC Hardware Consideration (U); System Consultants, Inc.; November 17, 1970; NAVAIRDEVCEN Contract N62269-70-C-0274; Secret-NOFORN; (42).

10.5 Simulation of AADC System Operation with E-2B Program Workload; William R. Smith; NRL Report 7030-19; January 27, 1971; Unclassified; (48, NPS). (Also see NRL Report 7259; April 22, 1971 with same title and author.)

10.6 Simulation of AADC Simplex and Multiprocessor Operation; William R. Smith; NRL Report 7356; February 29, 1972; Unclassified; (75, NPS).

10.7 On-Board Checkout and System Interface Requirements for F-14C Aircraft (U) Volume I - Final Report; Grumman Report D-512; September 7, 1971; NAVAIRSYSCOM Contract N-00019-70-C-0087; Confidential; (60).

10.8 On-Board Checkout and System Interface Requirements for F-14C Aircraft (U) Volume II Task A - Mission Statement; Grumman Report No. D-512; July 7, 1971; NAVAIRSYSCOM Contract N-00019-70-C-0087; Confidential with Secret Appendix; (61).

*54 means reference number 54 in the AADC Bibliography, Enclosure 1 to AADC Progress Report No. 10, and NPS means available at the Naval Postgraduate School.

10.9 On-Board Checkout and System Interfact Requirements for F-14C Aircraft (U) Volume III Task B - Baseline System Performance Definition; Grumman Report No. D-512; July 8, 1971; Confidential with Secret Appendix; (62).

10.10 On-Board Checkout and System Interface Requirements for F-14C Aircraft (U) Volume IV Task E - Mission Analysis Presented Outputs; Grumman Report D-512; September 7, 1971; Unclassified; (63).

10.11 Future Naval Aircraft Subsystem/AADC Interface Definition for Operational and OBC Requirements (U) - Final Report; Volume 1; Grumman Report D-545; April 17, 1972; NAVAIRDEVCEN Contract N62269-70-C-0065; Unclassified (89,NPS).

10.12 Advanced Multiplatform Computer System; Bruce Wald; NRL report to be published in the Summer of 1972; (79).

10.13 AN/UYK-7 - AADC Transferability Study -- Hardware Portion; Final Report; J. J. Symanski; NELC Code 3200; October 1, 1971; (65).

10.14 CMS-2 Software Transferability Study, AN/UYK-7 to AADC: N. S. Mathis; NELC Technical Document 207; November 13, 1972; (NPS).

10.15 Proceedings of High Level Aerospace Computer Programming Language Conference; NRL, June 29-30, 1970; R. S. Entner - Conference Coordinator, NAVAIRSYSCOM; Unclassified; AD-733-454; (34, NPS).

10.16 All Application Digital Computer 1973 Symposium; Orlando, Florida; January 23-25, 1973; Proceeding not yet available.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12
Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
Naval Electronics Laboratory Center Code 5200 271 Catalina Boulevard San Diego, California 92152	20
Naval Electronic Laboratory Center Code 3000 271 Catalina Boulevard San Diego, California 92152 ATTN: Mr. Bruno Whitney	5
Naval Air System Command Code AIR-5333F4 Jefferson Plaza #2 Arlington, Virginia 20360	15
Office of Naval Research Code 430C Arlington, Virginia 22217 ATTN: Mr. Joel Trimble	2
Naval Research Laboratory MIS Building 43, Room 146 Washington, D.C. 20390 ATTN: Dr. Bruce Wald	2
Chairman, Computer Science Group Code 72Bv Naval Postgraduate School Monterey, California 93940	1
Assistant Professor G. H. Syms, Code 53Zz Computer Science Group Department of Mathematics Naval Postgraduate School Monterey, California 93940	5

	No. Copies
General Dynamics Electro Dynamic Division P. O. Box 2507 Pomona, California 91766 ATTN: Library, MZ 6-20	1
CS4900 Course Department of Mathematics Naval Postgraduate School Monterey, California 93940	30
Dean of Research Naval Postgraduate School Monterey, California 93940	1
Naval Air Development Center Warminster, Pennsylvania 18974 ATTN: Code 8131, Mrs. Olive Redell	10

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

All Application Digital Computer: Course Notes

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report, 1972-1973

5. AUTHOR(S) *(First name, middle initial, last name)*

Gordon H. Syms

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1973	417	191

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. WR-2-6297	NPS-53ZZ73031A
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Air Systems Command

13. ABSTRACT

This report is a set of course notes, or text, on the proposed Navy All Application Digital Computer. The AADC, as it is called, is a programmer-oriented, general purpose, modular digital computer that was originally designed to meet all the 1975-1985 Naval airborne data processing requirements, but it has now had its role generalized to include "All Applications." Since the AADC combines many of the most advanced computer hardware concepts now under development in the United States, the study of AADC should be of general interest.

The all application role includes real-time and time-sharing computations, and special applications such as line concentrators, super modems, data channels and aircraft electric power controllers.

This report includes a chapter on each of the following: a general introduction and summary of all chapters, AADC architectures, all application role, hardware technology, Data Processor Element, Master Executive Control, Signal Processing Element, evaluating AADC developments, High Order Language, and AADC applications.

The report will be used for a 33-hour course for graduate students at the Naval Postgraduate School, but could be used for other audiences or for shorter courses.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
All Application Digital Computer						
Advanced Avionic Digital Computer						
AADC						
Fourth Generation						
New Technology						
LSI						
Computer-on-a-Chip						
Signal Processing						
Operating Systems						
Multiprocessor						
Multiprogramming						
Virtual Memory						