AD-A056 105 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 1/3
A STUDY OF ALTERNATIVES FOR VSTOL COMPUTER SYSTEMS.(U)
APR 78 U R KODRES , J D BUTTINGER
UNCLASSIFIED NPS52-78-001 NL

1 OF 3
AD A056 105

NPS52-78-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A STUDY OF ALTERNATIVES FOR

VSTOL COMPUTER SYSTEMS

Uno R. Kodres
James D. Buttinger
Richard W. Hamming
Carl R. Jones

April 1978

Prepared for:
Naval Weapons Center
China Lake, California 93555

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral T. F. Dedman
Superintendent

Jack R. Borsting
Provost

The work reported herein was supported in part by the Naval Weapons Center under the project number 77 WR30155.

This report was prepared by:

UNO R. KODRES
Associate Professor of Computer Science

JAMES D. BUTTINGER
Instructor of Administrative Science

RICHARD W. HAMMING
Adjunct Professor of Computer Science

CARL R. JONES
Associate Professor of Administrative Science

Reviewed by:

G. H. BRADLEY, Chairman
Department of Computer Science

Released by:

W. M. TOLLES, Acting
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-78-001
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A STUDY OF ALTERNATIVES FOR VSTOL COMPUTER SYSTEMS
5. TYPE OF REPORT & PERIOD COVERED: Final Report for Period Feb 1977 - Dec 1977
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Uno R. Kodres, James D. Buttinger, Richard W. Hamming, Carl R. Jones
8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93940
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: N60530 77WR30155

11. CONTROLLING OFFICE NAME AND ADDRESS: Commander, Naval Weapons Center, China Lake, CA 93555
12. REPORT DATE: April 1978
13. NUMBER OF PAGES: 251 pages (253 P.)

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited

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

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
airborne computer systems
cost analysis distributed systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):
This study assesses the impact of Large Scale Integration on future airborne digital systems, with a focus on the VSTOL systems. The study addresses the design, implementation, testing, servicing and the associated life cycle costs of airborne digital computer systems, both the hardware and the programs necessary for successful operation of the system. The scope of the study is limited to the computer system, not the sensors, keyboards, displays and other peripheral equipment.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

78 07 10 162

251 450

The study provides: information for decision making on the future course of action, a design philosophy, a process analysis methodology, and a life cycle cost analysis method.

ABSTRACT

This study assesses the impact of Large Scale Integration on future airborne digital systems, with a focus on the VSTOL systems. The study addresses the design, implementation, testing, servicing and the associated life cycle costs of airborne digital computer systems, both the hardware and the programs necessary for successful operation of the system. The scope of the study is limited to the computer system, not the sensors, keyboards, displays and other peripheral equipment.

The study provides: information for decision making on the future course of action, a design philosophy, a process analysis methodology, and a life cycle cost analysis method.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. EXECUTIVE SUMMARY

A. PURPOSE

The purpose of this study is to assess the impact of Large Scale Integration (LSI) of electronic circuits with respect to future airborne digital systems. Although the findings are applicable to any airborne system, the focus of this study is the VSTOL aircraft projected for development and production in the 1985 time frame.

B. SCOPE

The study addresses the design, implementation, testing, servicing and the associated life cycle costs of airborne digital computer systems, both the hardware and the programs necessary for successful operation of the system. The scope of the study is limited strictly to the digital computer systems and does not include the sensors, which provide the data, the displays, keyboards, and switches which provide the human interface, or the effectors which help carry out the actions of the system.

C. OBJECTIVE

The study provides information for decision making on the future course of action in the highly volatile electronic circuit industry. The study also provides a design philosophy, an analysis methodology useful for program design, and a life cycle cost analysis method applicable to similar studies.

D. METHOD OF APPROACH

The study first explores the implications of the technological changes which are brought about by Large Scale Integration. Although the technological changes relate to hardware, these changes imply corresponding changes in system architectures and programming. One of the most important cost components is the software design. The study describes a set of software design principles which emphasizes uniformity, homogeneity, and a testable design. The design principles are applicable particularly to tactical systems which are known to be complex and difficult to test.

We separate the software design from hardware implementation. The software design can be carried out without committment to a specific computer hardware. The operational programs can be developed and tested on developmental systems which are specifically suited for program development.

Software design, implementation and testing is a time

consuming process and in major systems takes years to develop. Decisions on which computer hardware to use can be made at a point in time near the end of the development cycle. This insures an up-to-date hardware implementation and an improved transferability of software products.

We see two major trends in airborne system's architectures. These alternatives for hardware implementation are: the homogeneous and the heterogeneous systems. The homogeneous system consists of a collection of computers each of which is functionally identical. The heterogeneous systems contain at least two functionally different types of computers: the "mission computers" and the "embedded" computers.

In order to develop a life cycle cost analysis for the two major design alternatives, we first develop the projected functional requirements for the VSTOL (attack version) tactical system. Because the functional requirements of VSTOL will be similar to the presently operational attack aircraft, A6-E, we study the A6-E in great detail. From the detailed data we can estimate worst case execution times, the number of variables shared by processes, the number of instructions, constants and variables in the programs. By knowing the execution times for particular instructions on a given computer, we can estimate the execution times of program segments executed on that computer.

From this data we can compare the homogeneous and heterogeneous implementation alternatives for the A6-E, or for a projected system such as the VSTOL . The life cycle cost estimate is developed for the alternatives.

Based on the analysis of the A6-E system, it is established that presently marketed LSI computers can be

used to implement the systems. We develop cost comparisons which use presently available cost data. We project the cost comparisons into the 1985 timeframe by structuring two scenarious and three cases within each: the "most likely", the "optimistic", and the "pessimistic".

E. MAJOR FINDINGS

1. The Software Aquisition Cost

When we discuss the cost of software, we distinguish between software development costs and software aquisition costs. The development is a "human intensive" activity and its cost is high in comparison to production costs of LSI circuits. Although program development costs are variable, the variability is generally bounded by $5 - $80 per instruction. The program aquisition cost is dependent on the number of potential users of the program. Software development tools such as editors, assemblers, compilers, debuggers and operating systems can be bought for $30 - $1000 per program. The aquisition cost per instruction for widely used programs ranges from $.001 - $.02, about three to four orders of magnitude different from the program production costs. Therefore custom built software, which is exclusively designed for the Navy (CMS-2 compilers, AN/UYK-7 operating systems, AN/UYK-20 operating systems) is high in aquisition cost in comparison to widely used compilers and operating systems. To minimize software costs it is important to avoid custom built software as much as possible.

2. The Hardware Aquisition Cost

We again distinguish between hardware development costs and hardware aquisition costs. Hardware desigr and development is a "human intensive" process, hence the design and development cost is high. The hardware aquisitien cost varies widely. If a distributed computer system is built from modular LSI single board computer systems which are widely used, the aquisition cost per ccmputer is in the range from $500 - $2000. If the computer is a custom design, the price per computer, even if LSI chips are used in the desigr, jumps tc the range $20,000 - $50,000. To minimize aquisition costs it is important to avoid custom designs and custom built hardware as much as possible. In the highly competitive LSI hardware market, the widely used hardware aquisition costs are likely to drop and make the future cost differential between custom designs and widely accepted designs even greater.

3. The Software Maintenance Costs

Changes in software have an extremely high ccst per instruction. Literature quotes a range from $500 - $8000 per instruction. The cost is dependent on hcw modular the programs are, how well they are documented, the complexity cf programs, the language used, etc.

Any errors in the program which pass the acceptance tests are particularly expensive because the maintainers have tc become as familiar with the program as the originatcrs. Exhaustive acceptance tests, on the cther hand, are impossible to conduct (There are approximately 10^{**} possible program paths in the A6-E prcgram). Mcdular design and thorough module testing is the way to achieve success.

The cost differential between homogeneous and heterogeneous systems for maintenance and update of software depends on the languages used to construct the software. If only a single higher level language is used, the cost differential is small. Assumed here is that the higher level language compiler aquisition cost has been included in the software aquisition cost. If assembly languages are permitted, the educational cost of software maintainers will vary in proportion to the number of distinct computer types used in the system.

4. The Hardware Maintenance Cost

The hardware maintenance cost differential between homogeneous and heterogeneous systems is large. The educational costs of maintenance personnel are proportional to the number of distinct computers in the system. The test procedures and test programs necessary vary in direct proportion with the number of distinct component computers. The spare parts inventory, the paperwork in the supply system, and the documentation at the repair facility - all these costs are multiplied by the number of distinct computers in the system.

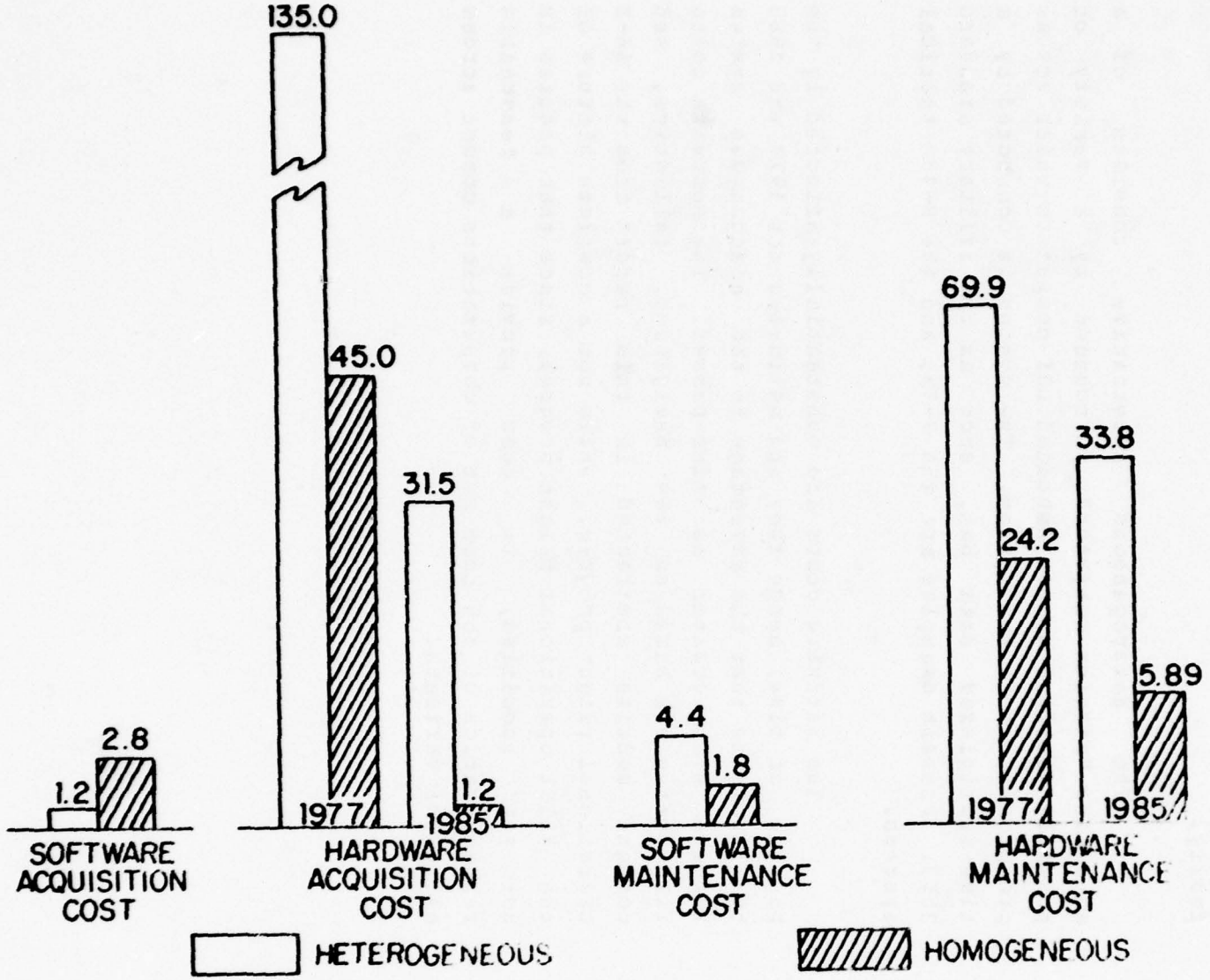
5. Cost Summary Projection

Figure I.1 presents the estimated cost comparison between two implementation alternatives: the homogeneous and the heterogeneous. The homogeneous alternative consists of a system of identical LSI computers in a homogeneous network. These computers are commercially successful and satisfy military standard requirements imposed by the severe environment in which they must function. Three examples of

presently existing systems are: Digital Equipment Corporation's LSI11, INTEL's 8080, Texas Instruments' 9900 family.

The heterogeneous alternative consists of a so-called "mission computer" surrounded by a variety of possibly distinct sensor embedded LSI computers which act as pre-processors to the system. The system is connected by a time-multiplexed data bus, such as the military standard 1553. Present examples are the F-15, and the F-18 tactical systems.

The hardware costs are substantially affected by the passage of time, hence they are estimated for 1977 and 1985 to illustrate that the advantage to the homogeneous system becomes even greater as time passes. The software costs illustrated are based on the navigation, ballistics, and command modules abstracted in this report from the A6-E operational flight program. While not a complete picture of the VSTOL operational flight program, since that program is not yet specified, it does provide a reasonable representation of the core set of computations common across all VSTOL variants.



COST SUMMARY PROJECTION 1977 & 1985
(MILLIONS OF 1977 $, BASED ON 1,000 PLANE BUY)

Figure I-1

Figure I.1 shows that the estimated software aquisition costs do not differ substantially in the two alternatives. There are uncertainties dependent on the aquisition strategy. If the Navy special language CMS-2 is a requirement for all programs, including embedded processor programs, then a CMS-2 compiler would have to be written for each computer. We assume that this will not be done and that a variance will be granted for the periferal computers. If a widely used higher level language (FORTRAN, BASIC) is permitted, a relatively small aquisition cost is associated with the compilers for the LSI computers.

The differences between the homogeneous and heterogeneous alternatives are greater for the hardware aquisition cost estimates. A special purpose design for the Navy cannot be cost-shared and hence the hardware aquisition cost for the "mission" computer is high ($50,000). The embedded computers are low cost items if no uniformity requirements are imposed and each subcontractor uses "his own" favorite embedded computer. In the homogeneous alternative embedded computers must be identical, hence a higher aquisition cost is required if the embedded systems must be redesigned. The estimated hardware aquisition costs for the heterogeneous system is nevertheless higher.

The software maintenance cost estimate is higher for the heterogeneous system. The cost difference is greater if assembly language programs are used in the embedded computers.

The hardware maintenance cost estimate has the largest difference between the homogeneous and heterogeneous alternatives. If the hardware reliability is as high as is expected, then the total estimated maintenance cost will be lower than our present experience indicates. The difference

in total costs between the design alternatives will therefore be less pronounced than shown in Figure I.1.

Not included in this summary but specified in Chapter IX is the additional cost of aircraft overdesign for the extra weight inherent in the mission computer of the heterogeneous alternative.

The total cost estimate comparison between the system's alternatives shows that the homogeneous system has substantial advantages over the heterogeneous alternative. However, to carry out the homogeneous design concept requires a high degree of discipline and cooperation between contractors, subcontractors and the Navy project office. Because projects are funded on the basis of aquisition costs rather than lifecycle costs, the homogeneous system must show its advantages during the aquisition phase, while most of the cost differential will appear during the maintenance phase.

II. INTRODUCTION

A. BACKGROUND

Although analog devices which might be called analog computers have been used in airborne applications for a long time, digital computers have been used only recently, in the late 1960's and early 1970's. The Navy attack aircraft A6-E and A7-E have a comprehensive tactical system based on a general purpose digital computer. The antisubmarine warfare aircraft, the P3-C and S3-A, the radar surveillance aircraft, E2-C, and the electronic warfare aircraft E6-B all depend on a general purpose digital computer system as a vital part of the weapons system. Because of decreasing costs, weights, and power requirements and increasing reliability and capability of digital electronics, the trend toward more use of digital technology is clear.

There is also an observable trend in the system's architecture of the Navy's presently acquired systems. The F-18 typifies the concept of the tactical system consisting of a central "mission computer," a dual CPU AN/UYK-14, which is the so called airborne Navy "standard" computer, surrounded by a distributed set of "embedded" computers, each of which is dedicated to some fixed functional task such as navigation, flight control, or fire control. The "mission" computers together with the "embedded" computers, which may be different from each other, make up a "heterogeneous" digital system.

In light of the rapidly changing Large Scale Integration (LSI) technology, there are numerous choices to implement future airborne systems. Which choice is made will have important consequences in cost, weight, reliability, and capability. This study addresses two major alternatives of system's implementation: the homogeneous alternative consisting of a system of functionally identical processing elements connected into a regular network; the heterogeous alternative which contains a central "mission computer" with a mix of "embedded" processing elements connected into a network by a serial time multiplex data bus, such as the MLSTD 1553(B).

The technical feasibility of the homogeneous alternative is subject to question, because there is a belief that the currently available LSI processing elements are too slow and too small to carry out the tasks demanded by a real time tactical system. A substantial part of this report is devoted to establishing the technical feasibility of the homogeneous alternative.

Several reports have addressed the implementation of tactical systems using LSI processing elements. The Honeywell report [13] represents a view which anticipates that the airborne computing will soon become distributed among identical LSI processors which are connected by a data bus of high data rate. The report recommends that we proceed with laboratory models instead of paper studies. Although the study anticipates microprocessors of some capability, the authors in 1973 did not anticipate the powerful single chip computers available in 1977.

Texas Instruments, [4] produced a report in 1975, which accurately projected the availability of 16 bit microcomputers with multiply and divide functions executed at the speeds which are realized in 1977. Their analysis

accurately projects costs, develops design methodology for distributing computing among a collection of 20 independently operating processors connected by a local bus into affinity groups. The affinity groups in turn are connected by a global bus to form the tactical system. The report includes analysis and design tools which are worked out in great detail and provide a designer with useful tools to implement a tactical system with presently available computers. Their report closely parallels the analysis found in this report.

A report by McDonnell-Douglas [9] represents the view that a central mission computer, surrounded by the special purpose computers is the preferred design. Distributing the processes among many small computers creates reliability problems, according to their report.

Sperry Univac study [8] concerns itself more with design methodology rather than any particular implementation. The methodology suggests a series of seven steps which tends to separate the software design from the hardware implementation. The operational flight program is designed in terms of decompositional units which are at the final stages of design mapped into hardware or firmware.

A more general report which addresses the tactical computer needs not only for airborne computers but tactical systems used in the Army and Navy, was published in 1976 [3].

The Army/Navy Computer Family Architecture (CFA) Selection Committee's final report recommends the use of the PDP-11, IBM S/370 or Interdata 8/32, based on architectural suitability, support software availability and life cycle costs. This final report recommended to both the Army and the Navy a suitable family of computers to implement

tactical systems. The committee did not consider LSI computers, possibly because some of these computers had not been announced at the time the committee started its work. However, the committee's recommendations are largely based on the availability of support software for these systems which are architecturally acceptable. The existing Navy standard computers, AN/UYK-7 and AN/UYK-20 failed to qualify architecturally under the criteria used by the committee, and would be poor choices because the support software base, even at this point in time, is inadequate. The committee did not anticipate the impact that LSI computers have had on the cost of support software. The committee did not differentiate between the development cost of support software and sales price of support software when the customer base is large. The committee tacitly assumed that software development would be carried out on the same computers that are used for the application. There is general agreement that program development is best done on special developmental systems, as is the case with many LSI computers.

B. A METHOD FOR ESTIMATING VSTOL'S NEEDS

Before we can realistically compare alternative system's implementations, we must estimate program size, execution speed requirements, and data flow requirements. We introduce methodology based on graph theory, which permits a detailed analysis of execution speeds and data flows. We apply these techniques to analyse the A6-E operational flight program.

If the execution speeds for the instructions of a particular computer are known, then we can estimate accurately the execution times that a program segment would

require if executed on that computer. Similarly, if we know the data bus data transfer speeds, we can accurately estimate the time required to transfer data between computers. Based on this analysis, we can accurately estimate the number of processors needed to carry out the A6-E operational flight program using a homogeneous distributed system or a heterogeneous distributed system.

Because system's needs for VSTOL are similar to fixed wing aircraft which carry out the same functions, we can use the A6-E tactical program as a starting point for extrapolating the operational program requirements for VSTOL, attack version. Similarly, the S3-A can be used as a starting point for estimating the system's requirements for VSTOL used as an antisubmarine aircraft.

By establishing feasibility of homogenecus distributed systems with presently available commercial LSI processors, it is clear that the improved capability of the LSI computers by 1980's will reduce the number of processors required and also reduce the presently experienced hardware costs.

C. ORGANIZATION OF THE REPORT

Chapter III describes the so-called LSI revolution, its implication both in hardware and software development. The industry trends in process control applications and embedded computing are discussed and related to the future of Navy airborne tactical computing.

Chapter IV states the problems posed by rapidly changing technology. Design principles applicable to airborne tactical system's software and hardware design are stated.

The two major design alternatives: homogeneous and heterogeneous are discussed in detail.

The methodology for the analysis of distributed systems is given in Chapter V. Execution time analysis techniques and data flow analysis are both based on the concept of graphs. From this analysis execution times can be estimated, data flow requirements between processing elements can be determined and partitioning strategy can be formulated.

Chapter VI applies the analysis methodology to the A6-E system. A detailed association of computational steps with program segments is obtained from the A6-E operational flight program documents. Data flow requirements between suggested program partition elements is calculated and program size estimates are given in both higher level languages and a machine language. Estimates of the VSTOL operational flight program are obtained.

The systems implementation alternatives are considered in Chapter VII. A proposed homogeneous distributed system's design using presently available LSI processors is compared with a heterogeneous design. The implementations with improved technology in the future will allow more capable systems with smaller computers, less weight and at less cost.

Comparisons in the reliability of the designs are considered in Chapter VIII. Economic analysis with two scenarios for future possibilities are considered in Chapter IX.

III. IMPLICATIONS OF LARGE SCALE INTEGRATION

A. TECHNOLOGICAL CHANGE AND LSI IMPLICATIONS

The technology of Large Scale Integration is one of the most significant technological events in the twentieth century. A machine can now be endowed with "intelligence". Although computers have been in existence for more than thirty years, only very special machines could afford "intelligence". For example, the Mars lander was one such machine. In the near future many machines will have capabilities of the Mars lander.

The agriculture industry's tractor which converts chemical energy into mechanical work has made it possible for two percent of the population in the United States to feed not only the entire population of United States but millions of others. At the turn of the century sixty percent of the population was required to feed the rest. Similarly, with "smart" machines, the productivity of each of us can increase to such an extent that only a minority of workers are required in the direct production of the world's goods, the rest could be employed in services or information processing. Radical changes will take place in both the social structure and our values as a consequence of a silicon chip which can be made into a willing slave, a skilled pilot, or a deadly weapon.

The technology of Large Scale Integration affects military airborne systems in three major ways:

1) The cost of the hardware is potentially radically reduced.

2) The system's weight and power requirements are radically reduced.

3) The system's design distributes the computing among several computers. Distributed system's architecture allows future add-ons to be made in an orderly way.

In order to gain insight into why the radical cost reductions in hardware are possible, we start with cost analysis for LSI technology.

In producing any product, the cost can be divided into a nonrecurring cost, NRC, and a recurring cost per unit item, RCU. If we produce N items of a certain type, then the sales price per unit, SU, should be such that the income on the left of the inequality exceeds

$$N*SU > N*RCU + NRC$$

production cost on the right. In general, the quantities in this formula are time dependent, so that each should be expressed more accurately as N(t), SU(t), RCU(t), NRC(t). In order for the producer to continue successful operation in the long run, at some point in time

$$\int_0^T N(t)*SU(t)\,dt > \int_0^T N(t)*RCU(t)\,dt + \int_0^T NRC(t)\,dt$$

It depends on the company's pricing policy whether or not the inequality holds at several points in time or strictly in the long run. For companies which do not change their sales price, the above formula simplifies to

$$SU > RCU + NRC/N(t)$$

To frame the cost issues in LSI technology by this inequality, first note that the microelectronics issue of the Scientific American [21] breaks down the manufacturer's recurring cost of producing the LSI chip as follows.

Cost Component	Cost/Chip	Cum Cost/Chip
Untested chip on a wafer	$0.10	$0.10
Tested chip assuming 20% yield	$1.00	$1.10
Packaging and package testing a chip	$0.50	$1.60
Assembling chips on a circuit board 100 circuits/board	$1.00	$2.60
Cabinet and power supply for a 20 board system	$0.35	$2.95

The nonrecurring costs are measured into the millions of dollars. These costs include: the cost of market surveys to decide what to make, the logical design, the layout design, the documentation for the design, design of tests for each chip, the writing of users manuals, advertizing and

disseminating information about the product, life testing, user education etc. An estimate for the nonrecurring costs associated with the very successful INTEL 8080 chip is $5,000,000.

The 8080 chip sells from several distributors at $15 per chip in single quantities. Putting the numbers into the cost formula

$$\$15 > \$1.6 + \$5 * 10^6/N(t)$$

shows that sales of the number of 8080 chips to date has totalled at least about half million. Later in this report a sales estimate of about $6 * 10^6$ microprocessor devices for the industry is made. With INTEL holding about 30% of the market, that would be roughly $1.8 * 10^6$ devices or a total cost per chip of $5.

It is very important to understand that Large Scale Integration by itself will not reduce the cost of computer hardware. It is only when a chip or a system becomes popular that the sales price drops to recurring production costs. As an example, the AN/UYK-14 is built from LSI chips. The total estimated production cost for the chips is:

Memory 65K x 16	= 65 chips, 16K bits each
CPU	= 14 chips
I/O channels	= 32 chips
Miscellaneous	= 20 chips
Total	=131 chips
	x $3.00
Estimated recurring	

production cost = $393

While it is true that military specifications require different packaging and additional testing of the chips, which causes a cost escalation by a factor of 2 - 3 per chip, the estimated recurring production costs would still range from $800 - $1200 per system.

The present aquisition cost of the AN/UYK-14 is approximately $50,000. There is no reason to suspect excessive profits simply because the nonrecurring design, test, documentation and customer education costs are high and the potential sales volume is relatively low. Hence even with a contract strategy which separates development from production phases, as with the AN/UYK-14, the contractor is still probably amortizing development costs in the production phase of the contract.

In order to make effective use of the LSI technology, it is important to select a system which is widely used. The nonrecurring production costs can then be shared among a large population of users.

A massive use of a system has also important implications in software costs. The same cost formula can be applied to software production.

$$SU > RCU + NRC/N(t)$$

The nonrecurring cost is estimated to be in the range $5 - $80 per instruction. The recurring cost usually involves

duplicating a magnetic tape, a paper tape, a floppy disk, or a deck of cards. In all cases the recurring cost is almost negligible. Therefore, the aquisition cost of a program which has a large number of users becomes small. For example, a floppy disk operating system, including utility programs such as assemblers, editors, debuggers, compilers, for an INTEL 8080 system sells for $75 per copy. The length of the program is about 30,000 instructions. Therefore the aquisition cost per instruction is $.0025. Typical sales prices for FORTRAN, BASIC, COBOL compilers range between $500 - $1000 per compiler. This contrasts with the Navy's estimated aquisition cost [3] of $4,900,000 for the CMS-2 language compiler. Even if there are 100 Navy program development centers using this compiler, the cost to the user is $49,000.

E. INDUSTRY TRENDS IN EMBEDDED COMPUTING

Microprocessors are architecturally designed to permit maximal use of the chip area. The continuing trend is to place more and more circuits on a chip. The trend is in three directions:

1) The microcomputer on the chip (INTEL 8048, TI-9940).

2) Greater arithmetic capability on the CPU chip (TI-9900) i.e. 16 bit multiply and divide function on one chip.

3) Byte slice chips which can be used to build computers of arbitrary word length (INTEL 3000, AMD 2900).

A few years ago the microcomputer system contained a board which had the CPU with additional chips to communicate

with memory and input /output. The memory and input/output ports were on separate boards. The single board computers combine the CPU, memory, and input/output circuits on one board. The latest trend is to put the CPU, memory and input/output circuits on a single chip.

Undoubtedly the most useful chip will be the one which is arithmetically powerful, contains a sufficient amount of memory which can be extended and has input/output capability.

C. THE FUTURE OF NAVY AVIONICS

If the single chip computer of the 1980's will have 8K-16K bytes of memory, it will be powerful enough to perform each of the modular functions which are currently implemented in airborne tactical systems.

The leading microcomputer manufacturers have developed parallel time multiplexed bus technology (INTEL-MULTIBUS, DEC UNIBUS, TI TILINE). The hardware bus technology is supported by distributed single board real time executive software so that the user need not involve himself with anything but applications programming.

By the first half of the 1980's powerful distributed systems consisting of a network of single-chip computers on one board will be replacing the present single board computers.

Auxiliary memory in the form of magnetic bubbles and charge coupled devices will provide essentially unlimited auxiliary memory for these applications where auxiliary memory provides memory space for occasionally used programs.

The presently limited human capability of writing programs will be the only delay in the process of creating useful systems.

The Navy has an opportunity to use this new technology to its fullest. It cannot afford to do so by continuing to create its own special brand of computers (AN/UYK-14) and continue to use its own special brand of languages (CMS-2). The dedicated airborne tactical systems can be designed and implemented by the use of distributed LSI processors. Chapter IV and Chapter VII show in detail how this can be accomplished for the A6-E system, or systems similar in function, VSTOL . Our design makes use of the presently available LSI processors in order to establish feasibility. The more powerful LSI processors likely to exist by 1985 will make future system's design easier, the total systems weight susbstantially smaller and the computer system's reliability higher.

IV. PROBLEM STATEMENT

A. DESIGN PRINCIPLES

1. Introduction

This section discusses some principles for the design of large, special purpose computer systems such as occur on aeroplanes, ships, and other large, complex systems involving information gathering, transformation, and transmission.

First, the words "large" and "system" imply a systems design approach, which in turn means that we ask, "What must we do?" rather than the usual, "what can we do?" Of course in the end we must be able to carry out the design, but understanding the systems constraints should precede putting something together to see if it "works".

Second, we have learned from past experience that testing is a major problem in computing systems. Why is testing so hard? At the bottom is the simple fact that the growth of the number of combinations of the various parts of a typical computer system (both hardware and software) is astronomical; it is totally impractical to try every combination to see if it works. For example, a million computers working for a million years at a million times current speed (a million operations per second) can do less than one millionth of the possible 64 bit by 64 bit

multiplications. Thus we cannot hope to test the entire complex system as a whole, we can only verify that for an infinitesimal fraction of special situations the system works, and then hope that the rest is correct. Therefore, we must actively seek designs that make it possible to test the parts separately, and then test the interconnections at the interfaces one by one at most.

As a practical example of this combinatorial growth principle, consider the fact that it is easier to design an integrated circuit than it is to design the tests for it; that it is easier to manufacture an integrated circuit than it is to test it afterwards to see if it operates properly; that it is easier to build software and other programs than it is to test them.

As an example of two different approaches to the design of a software package consider the problem of writing a program to integrate a particular second order non-linear ordinary differential equation. If you begin by writing a general purpose integration routine, then you can test it using various standard functions like sines and cosines, growing and decaying exponentials, Bessel functions, etc. With a wide variety of well known functions you can probably cover most of the particular aspects of the equation you have to solve. Finally by specializing the general purpose routine to the particular problem , you will have a great deal more confidence, at a lot less labor, than with the direct approach to the special case. Of course the general program may well be slower and use more storage than the optimal, special purpose routine, but by going the general purpose path you have gained a lot in both reliability and savings in the effort in testing.

The same applies to hardware. It has been observed in the literature that it is easier to test a computer with

a "random access storage" device than it is one with a "read only storage" device. In the latter case you must provide extensive testing equipment outside the computer to do the testing; in the general purpose case you can use the device itself to aid in much of the testing.

From these examples we extract three principles:

1. We must deliberately seek designs that make the testing, both of hardware and software, as easy as possible; not only initially but over the life of the system with its many changes and upgrades.

2. General purpose hardware and software offers flexibility at the testing stages, and furthermore it tends to be composed of relatively independent parts. Therefore at every stage the general purpose system should be considered instead of special purpose tricks that appear to save money and effort at the moment.

3. A homogeneous system, both hardware and software, tends to be easier to test, both in the designing of the tests and their execution. Furthermore, there is a great degree of self-testing of the homogeneous structure as it is used in the various ways in the whole system. Any errors that are found and removed are thereby removed from all their appearances at the same time - they need not be found again and again if the correcting is done to the system, not to the detail where it is first found.

But it is obvious that the problem to be solved by the whole computer system is highly varied. Therefore the underlying problem is to confine this variability as much as possible, and to construct as regular a system as possible so that the regular system may be tested relatively independently of the particular details of the system we

face.

If it is still doubted that testing is the problem, then consider the endless number of field changes that will occur during the life of the system. The cost of these changes, and the risks of errors, will greatly exceed the cost of the initial construction if the classical methods of computer system design are used. In our judgement the solution to the problem of designing a computer system lies along the lines indicated - regular, systematic, general purpose systems so that the testing problem can be adequately handled. Once the general system is tested, then the special cases of the particular problem can be used with fair confidence. The cost is that somewhat more capacity in speed and storage is needed (or equivalently a few more computer chips are needed). Estimates suggest that this extra cost is in the few percents, possibly in the tens of percents, but is nowhere near double the minimal capacity.

2. Generalized Testing

It comes as a surprise to many people that there can be general purpose testing methods that are relatively independent of what is being tested. A simple example of this is the testing of the computed answers of a sequence of equally spaced function evaluations. Typically this spacing will be close enough for the function to be "smooth". Thus the usual method of constructing a difference table of the function can be used to reveal isolated errors. Similarly, in flight, streams of smooth data can be checked for smoothness, and isolated errors located as they occur.

Another example of generalized testing is as follows. Given a double precision routine for integrating ordinary differential equations (in practice this should

include the ability to handle given singularities) one can test library programs of special functions by supplying their differential equations and starting values, and then comparing the results at a tight net of points. One gets tens of thousands of checks without any human ever being bothered with providing the check data. The same tool works on most special functions, and once tested on a couple of functions it is probably well debugged. The general purpose tester, by its generality, gets debugged early, so that apparent errors that later appear are most likely due to the program tested not the test program (which in the past has been one of the curses of testing). The errors in the tester are much more likely to be discovered from its multiple use than are those of special purpose testers.

As a final example of general purpose testing, one can test a data transmission system by encoding a random input message into an error detecting and/or an error correcting system, and at the receiving end isolated errors can be detected without knowing what that input was. Of course if the same random number generator were at the receiving end, a more complete check could be obtained.

This, then, is one of the things we seek; general purpose methods of testing that can be automated in the sense that humans do not have to construct the correct answers to be used in the testing. Human capacity is too limited to do much this way. Such a general purpose tester can supplement the comparatively few tests that humans can devise and apply to the whole system. We need an ensemble of tests that do not require human thinking to apply to each one , so that from the outputs alone the machine itself can locate many (not all) errors. Thus the computing capacity of the system can be used to test itself without exhausting the human invention of special tests and the efforts necessary to apply them.

3. Testing Hardware

In practice the manufacturer tests his product as much as he can, but it is in the field use over a wide range of users that the final bugs of a complex piece of hardware are found. The same is true of the computer chip manufacturer; in the long run they must depend on the testing power of daily use to locate the residual errors.

This suggests that when possible, standard, widely used computer chips be used rather than special purpose ones. In the long run more will be accomplished in most cases. This is not to say that special testing of chips should not be done locally, but that these tests should be directed towards the special circumstances of their use. Again, it is completely hopeless to test every combination of so complex a device as a microcomputer.

If the same kinds of general purpose chips are used throughout the system, then the testing is much reduced; alternatively, given the same amount of testing resources, a few types of chips can be more completely tested than can a wide variety of chips.

Along these lines, apparently very little is now known about the kinds of failures that modern integrated chips have. And until they are better known, it is impossible to come up with good design to compensate for the failures. Once the "noise" of the failures is known, then there are many different ways of compensating for them. If compensation for errors is made both for those that do occur and for those that are merely thought to occur, then much of the effort will be misdirected. Thus it is believed that the users should begin serious life testing of commercial

chips so that when the time occurs for the final design to be made from the integrated circuits, the basis for good design will be known. If the testing is started shortly before the final design must be made, then the life testing will have too short a time to be meaningful.

4. Reliable Computing from Unreliable Parts

This is not a new field. Error detecting and error correcting codes have been known and used for many years. The codes in the literature have been designed mainly for "white noise". Special situations and particular failure modes require other inventions, but the field is sufficiently well known so that invention is not hard to do. For example, suppose you decide to use read only storage devices for all programs, but that occasionally such storage devices fail completely. How does one construct a reasonable way for recovering without duplicating all of storage? One way is to put an error correcting code on each storage chip, and this will allow isolated errors to be corrected. These error correcting codes can have their bits in location 000... and the top end of the chip without much trouble, so that the checking bits are not scattered all over the storage device. If a larger failure occurs, say a whole bank goes out, or the error rate rises sharply indicating a disaster in the near offering, then we can do the following. We carry a spare storage device which has the logical sum of all the other storage units, except the failing one, into the selected spare, we can reconstruct the failing one (without reading it at all). Any isolated errors in it can be corrected by its own error correcting code. It is not being claimed, in the absence of any reliable data, that this should be done - it is given only as an example of how one can compensate for unreliable parts without the heavy cost of duplicating every part, or even triple or

"quadding" each part. The more parts you put into the total system the more failure you will have and the less you will be able to test the individual parts (since it is supposed that you have a fixed, finite amount of effort to do this). More intensive checking on fewer parts is better than less intensive checking on more parts.

5. Testing Software

Just as using the same kinds of parts in the hardware greatly eases the problem of testing, so too will the use of the same kinds of software. Care should be taken that the same functions are not programmed in trivially different ways in different parts of the network of computers. The software should be approached as a whole - systems design is necessary in software.

Since the software must do different things, it follows that there must be differences. How, then can we get homogeneity in it? One method is to start with the mathematical equations in a standard notation (say FORTRAN) along with the corresponding Boolean logical realionships and the timing conditions. Then using appropriate general purpose compilers (say FORTRAN plus a separate timing checker) we can get the code we need generated by the machine itself.

If an error occurs, there will be a great tendency for the programmers (judging by past experience) to "patch in machine code". This should be resisted as much as possible. Instead, go back and fix the cause, the original statements, the bug in the compiler, or whatever it turns out to be, but do not fix the isolated error as an isolated error. The object should be to get all the final code to be machine generated in a uniform fashion by as simply

constructed compilers a possible. Thus by testing the general purpose compilers on many problems where the answers are easily known and checked, the compilers can be thoroughly tested. Then (as in the earlier cited differential equation example) the special cases that arise in practice will be more surely compiled correctly. Furthermore, all the inevitable changes that arise in the course of the life of the aeroplane will use the same well tested compilers, rather than going through the hands of new programmers who will have forgotten, if they ever knew, why things were as they were.

While we believe that much more can be done to create homogeneous software, the appropriate theory is not yet available, so the above is the best we can recommend at this time.

An example of getting fairly homogeneous software is the idea of having a table of status values and a program that uses the table to decide what to do next. Thus all the priorities are easily located and isolated for close inspection and control. The table values can be easily changed in mid flight, but the formula of evaluation should be changed only after long, careful study on the ground.

6. Reliable Software from Unreliable Programmers

The same problems of reliability occur in software as occur in hardware, though perhaps a bit more severely. The answer we have given above, use the system to generate the software rather than let unreliable humans touch the final version, seems to be the best protection against the all too common isolated, foolish errors.

More needs to be done along these lines, but studies

of the kinds of software errors that occur are too few and too scattered to be very useful at this time. By the time software is to be built, much more will have been discovered on how errors arise. But the principle that the machine should write the final code will still apply.

Fault tolerant computing, both software and hardare, has received extensive attention in the literature and should not be ignored. But so far as we can see from moderately careful study, there are no fundamental principles in all that has been written. Instead, there are many good remarks, observations, and suggestions that should not be ignored, but neither should it be depended upon too much.

7. Testing the Statement of the Problem

Even if the hardware runs correctly and the software is written to specifications, there is still the chance that the given equations, Boolean statements, and timing conditions are wrong. Thus there must be testing of them too, as well as the whole system. First, many of the equations that are to be used cannot be completely new; much of the material must have been used in similar situations, and these should be used whenever possible to compare with what is being proposed.

Second, it is possible to design systems simulators, much as has been already done for some parts of the problem, that will test the system behavior as a whole before it is constructed in hardware and software, and can also be used to generate check tests on the complete target system. Several systems simulators of varying degrees of detail should be seriously considered.

Third, it is possible to build test equipment to simulate reality so that the assembled system has pseudo real signals as inputs. For example, a simulated target can be rolled across a hangar floor to see that the numbers at various places in the computer system are very close to the theoretical ones. Much as it seems to be trivial, the testing of the original proposed system is necessary. As experience has shown, errors can creep in at this early stage, let alone at later stages when small (apparently) changes in the terminal sensors and effectors are made. Each such change requires a careful examination to see if the changes are consistent with other assumptions.

8. Summary

History has shown that the past habit of "letting testing occur in its natural place" is very expensive . The combinatorial complexity of computers, both hardware and software, makes complete testing of current systems impossible.

A few people have finally realized that design begins with testing (acceptance tests if you wish). Do not design what you will not be able to test carefully.

Generally, small, standard, flexible units are more easily tested then are specially designed ones. Through use, isolated errors that escape the initial tests are caught. Once caught the error is to be removed, not by patching, but by careful analysis of how it escaped the testing, and then the cause is removed.

General purpose testing (both initially and in flight) is an area that offers great returns from limited human effort, and should be pursued further.

E. TWO ALTERNATIVES: HOMOGENEOUS OR HETEROGENEOUS

1. Distributed Dedicated Computing

The term distributed computing is a broad term which has a widely different meaning to different people. An attempt to define the term "sharply" leads to disagreement and sometimes even emotional outbursts simply because my definition is right and yours cannot possibly have any merit.

In the context of this report, distributed computing is used in the broad sense which includes systems which are at one end of the spectrum completely uncoupled and at the other end of the spectrum very tightly coupled, such as multiprocessor systems which share some memory. In short, distributed computing refers to a computing process which separates a task into two or more individual tasks carried out by two or more processors.

In the case of the Navy airborne tactical systems, the A6-E the A7-E-D, the P-3C would not be distributed systems, whereas E-2C, S-3A and F-14, F-18 would be distributed systems.

Some of the terms frequently used to refer to such systems are federated or dispersed systems. The term federated connotates a certain structural hierarchy so that one computer acts as the executive and others are subservient computers. The term dispersed connotates physical dispersal or distance between the processing

elements. In our use of the term distributed, no connotations of this type are implied. If we wish to discuss physically dispersed processing concepts, we would refer to such distributed systems as physically dispersed distributed systems. In our use of the term, distributed processing does not imply that the processing elements are in any sense coequal and homogeneous in structure. We would call such systems physically homogeneous distributed systems. Physically homogeneous distributed systems can be organized into logically hierarchical distributed systems where one processing unit may assume the role of the executive and the others as subordinates, closely modeling the military hierarchy. Systems which contain nonhomogeneous processors are called heterogeneous.

The following binary tree summarizes our nomenclature and taxonomy.

Digital Computing Systems

I. Nondistributed

II. Distributed

A. Heterogeneous
1) Logically hierarchical
2) Logically of equal rank

B. Homogeneous
1) Logically hierarchical
2) Logically of equal rank

Each of the categories can be either physically close or dispersed.

A dedicated computing system is one in which the

same set of tasks is executed over and over again. The contrasting situation occurs at university computing centers where a task stream is constantly changing and where the same program is seldom executed more than once.

2. Current Trends In Airborne Tactical Systems

The F-15 and F-18 show a trend away from the single central processor systems. Distributed systems in one form or another is the apparent trend. Also, these systems are heterogeneous in that several types of computers are used.

There is a natural force toward heterogeneous systems because the airborne systems manufacturers, who act as the major contractor usually hire subcontractors for subystems: radar, electronic warfare, communications etc. Each of these subcontractors probably has a different LSI computer which they prefer to use. To force them to redesign a system using a different computer would naturally increase the contract cost. Optimization of the cortract cost tends to encourage processor diversity.

From the point of view of life cycle costs, diversity of computers creates substantial additional costs, namely

1) Stocking line removable units for each type of computer at the repair station.

2) Documentation for each distinct unit must exist at repair stations.

3) A service person must either go to several different schools in order to learn to service different

systems or different service personnel for each distinct computer type is needed.

4) Software upkeep costs for the diverse systems will also become higher again because of documentation and personnel education costs.

Our cost analysis will show the penalties the Navy has to pay eventually if the objective of the project is to minimize the acquisition costs of the systems only.

3. Benefits Of Homogeneous Systems

The most important benefit of homogenous systems is that human beings who are designing, servicing and using such systems need to only learn one system. The human effort to design, service and use computers is definitely the most costly item in any system. Many chips of computer hardware can be bought for the daily wage of a hardware or software service engineer. Concern for minimizing computer memory by clever programming techniques is only economical when a large number of identical systems are designed. To try to isolate a hardware problem to anything other than the computer itself is soon becoming obsolete. How many of us bring a hand held calculator to a serviceman to fix it? Because a new calculator costs $20, no technician can afford to spend more than half an hour to service it.

Proliferation of LSI computers at this point in history is unavoidable. Any new revolutionary development will attract many groups who are trying to share in the profits and who cannot afford to spend time worrying about standards. Being first, establishing a reputation and staying first is more important and more effective in

establishing standards than spending endless hours trying to reach a compromise. It is however already clear that proliferation of microcomputers is ending. Many manufacturers find that they are too late or offer too little, and many have already closed their doors, or sold out, or merged with others. The defacto standards are being established.

Although the proliferation of LSI computers creates a tendency to have heterogeneous systems the homogeneous systems have substantial benefits. To add onto a homogeneous system requires typically adding another board into an empty slot. To isolate problems can be done more easily because swapping identically functioning replaceable units is a very simple and widely used technique for troubleshooting.

It is not hard to convince oneself that homogeneous systems have many advantages over heterogeneous ones. The question is, how can one push effectively against the natural trend of unique devices which has developed in the avionics industry.

A solution is to encourage two trends which are developing naturally among LSI computer manufacturers. One trend is the mutual agreement by several companies to manufacture the same product. The 8080 is manufactured by several companies including Texas Instruments and INTEL. The other trend is toward single board plug to plug combatibility. The SBC9900 and the SBC8080/10/20 are plug to plug compatible even though they are manufactured by different companies. If the Navy chose its standards to include the "defacto" industry standards, then homogeneous systems could become an economic reality in the Navy avionics computing.

V. METHODOLOGY FOR ANALYSIS OF DISTRIBUTED SYSTEMS

A. FLOWGRAPHS

1. Introduction

Complexity of programs has received considerable attention from the theoretical point of view. The complexity measure used is normally the number of basic operations required to compute the result.

Another significant measure of program complexity, namely the complexity of program control, has just recently received some attention [17].

In this report both measures of complexity, the execution time required to compute the result, and program control complexity are viewed as two aspects of the same problem. By the use of graph theory, the discrete systems analysis problems arising in electrical engineering or hydraulics engineering are shown to be abstractly the same as those arising in computer programming. The flowgraph which is used to describe computer programs is somewhat different from the traditional control graph. The view of flowgraphs presented here also corresponds to network flow problems for which there is a unit cost associated with flows through an arc. In Section A.2 the abstract similarity of Discrete Systems Analysis and computer programs is pointed out. In Section A.3 Kirchhoff's laws are applied to derive basic relationships which describe the

behavior of the discrete systems. These relationships are dependent on the structure of the system only and are applicable to all discrete systems which can be characterized by a dual set of variables called the flow and potential variables respectively. Section A.4 shows how the techniques used to solve discrete systems problems are equally applicable to determine execution time values in program segments. Section A.5 derives an expression for the total execution time in terms of independent flow parameters. A summary is presented in Section A.6.

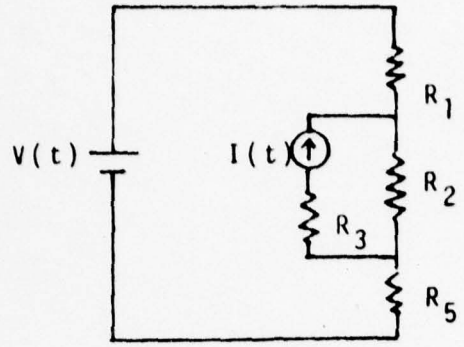
2. Abstract Similarity of Discrete Systems

In this section we introduce a view of programming which shows the abstract similarity of the programming problem to the problems in engineering and operations analysis. For a complete treatment of discrete systems arising in engineering, [12] and [16] are excellent sources. Reference [7] treats the problems in network flows.

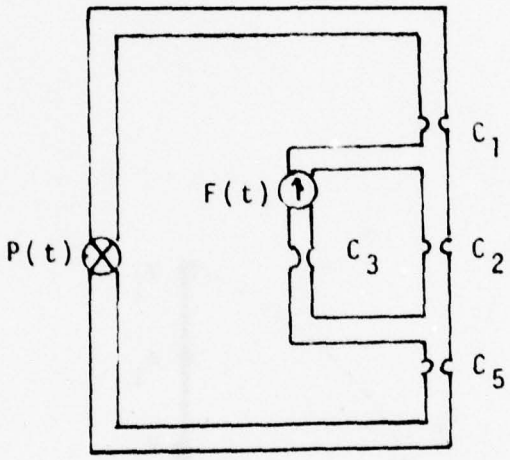
Figure V.A.1 illustrates three discrete systems arising from electrical engineering, hydraulics and computer programming.

We view the three discrete sytems as a collection of two terminal elements such as batteries, resistors, current generators, or pumps, constrictions in pipes and flow generators, or sequences of computer program statements, control statements and start and termination statements. The way in which the two terminal elements are joined together gives rise to a connected system of two terminal elements which may be described by the graph in Figure V.A.2.

(a)



(b)



(c)

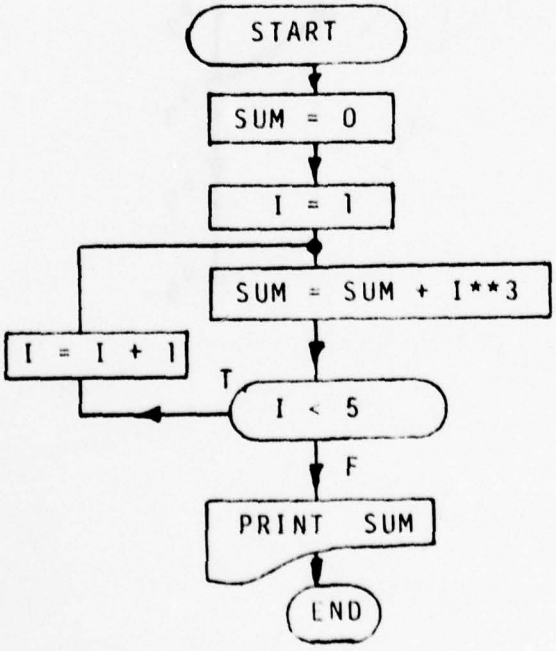


Figure V.A.1 THREE DISCRETE SYSTEMS

Figure V.A.2 ABSTRACTION OF THREE DISCRETE SYSTEMS

In Figure V.A.1(a) the vertex V_1 represents the terminal of resistor R_1 which is joined to the positive terminal of the battery. Arc a_1 represents the two terminal resistor R_1. The remaining correspondences are

$$a_2 : R_2,\ a_3 : R_3,\ a_4 : I(t),\ a_5 : R_5,\ a_6 : V(t)$$

where I(t) represents the current generating element and V(t) represents the voltage source.

In an analogous way the symbol X represents a pump whose one terminal is at a higher pressure than the other and is connected to the constriction C_1. Therefore, the arcs again represent corresponding two terminal hydraulic elements.

$$a_1 : C_1,\ a_2 : C_2,\ a_3 : C_3,$$

$$a_4 : F(t),\ a_5 : C_5,\ a_6 : P(t)$$

The traditional way of representing program flowcharts by a directed graph has been to associate functional statements with vertices and control paths with arcs. If, on the other hand, sequences of functional statements are associated with arcs, and control points in the program are associated with vertices, then we may define the concept of a flowgraph which coincides with the one in Figure V.A.2. The arcs a_i correspond to the sequences of statements:

a_1 : SUM=0; I=1;

a_2 : SUM=SUM+I**3; I<5;

a_3 : I<5 is TRUE; I=I+1;

a_5 : I<5 is FAISE; PRINT SUM;

a_4 : NO OPERATION;

a_6 : END-START SEQUENCE;

From the knowledge of the characteristics of the two terminal elements and from the way in which these elements are connected, we can determine the behavior of the system.

There are two complementary variables x(t) and y(t) which may be regarded as functions of time and which play a central role in discrete systems theory. In electrical engineering x(t) represents voltage differences and y(t) represents currents. The two terminal element, resistor R_1, is characterized by the relation:

$$x_1(t) = R_1 y_1(t)$$

If the resistance value R_1 is known and if the current through the element is $y_1(t)$ then the potential difference across the element will be $x_1(t)$. Each of the six two terminal circuit elements are characterized by a relationship of this type

$$x_2(t) = R_2 y_2(t),\ x_3(t) = R_3 y_3(t),$$

$$x_5(t) = R_5 y_5(t)$$

Voltage and current sources are specified by relations

$$x_6(t) = V(t),\ y_4(t) = I(t)$$

We should note here that inductive and capacitative elements were omitted from the examples for simplicity. a capacitative element would have the characteristic relationship

$$C\ d/dt\ x(t) = y(t)$$

The inductive element would be characterized by

$$L\ d/dt\ y(t) = x(t)$$

A knowledge of the characteristics of the two terminal elements and knowledge of how they are interconnected allows us to formulate a linear system of equations. If the system contains two terminal elements which have a differential relationship, then the system is a linear system of differential equations. In programming applications the two terminal elements are equivalent to resistors and hence only ordinary linear systems arise. In the hydraulics system described in Figure V.A.1(t) the variable x corresponds to pressure and the variable y corresponds to flow. Depending on the characteristics of the constriction, we may formulate the relations

$$a_1 : x_1 = C_1 y_1,\ a_2 : x_2 = C_2 y_2,$$

$$a_3 : x_3 = C_3 y_3,\ a_5 : x_5 = C_5 y_5,$$

$$a_6 : x_6 = P(t),\ a_4 : y_4 = F(t)$$

In computer programs, each sequence of instructions requires a certain amount of time to execute. Therefore we may associate the execution time T_i with each execution sequence. The variable y_i may be thought of as the number of times the execution sequence is executed, or it may represent the relative frequency or probability with which a particular execution sequence is executed. The total time that a program is in the given execution sequence therefore is expressed as

$$x_i = T_i y_i$$

Here T_i is analogous to resistance, y_i is analogous to current flow, and time consumed by the program segment is analogous to the potential difference. In our example in Figure V.A.1(c)

$$a_1 : x_1 = T_1 y_1,\ a_2 : x_2 = T_2 y_2,$$

$$a_3 : x_3 = T_3 y_3,\ a_5 : x_5 = T_5 y_5,$$

$$a_4 : x_4 = T_4 y_4,\ a_6 : x_6 = T_6 y_6$$

We note here that any directed cycle in the program corresponds to a current generator in electrical engineering. We must know something about such cycles in order to analyze programs. In this instance we can determine, looking at the program, that y_4 is executed four times. If we are interested in determining the total execution time for executing the program once, then

$$y_6 = 1 \text{ and } y_4 = 4, \quad ET = \sum_{i=1}^{6} T_i y_i$$

We note that if we think of T_i as a per unit cost of flow of a certain commodity, then the above formula represents a network flow problem with costs, where v_1 in Figure V.A.2 is a source vertex and v_5 is the sink vertex and where the flows y_i may be integer constrained in case we think of y_i as representing the number of times a given arc is executed. If we think of y_i as execution frequencies, or execution probabilities, then the integer constraint is removed. Arcs may have capacity constraints such as a data dependent loop which is maximally executed n-times.

3. Kirchoff's Laws

In all discrete systems of the type treated here, the Kirchhoff's law which states that the sum of the flows into a vertex is equal to the sum of the flows out of a vertex is applicable. This law implies that the flows y_i are related and in particular that (n-1) of the variables may be expressed in terms of the remaining ones, where n is

the number of vertices in the graph. References [12] and [16] develop a systematic way of expressing the so-called tree variables in terms of the co-tree variables. The following contains a synopsis of that development.

Definition 1. A spanning tree T of a connected graph G, of n vertices is a connected subgraph which contains all n vertices and has n-1 arcs.

One spanning tree of the graph in Figure V.A.2 is given by the heavy arcs. Each remaining arc in the graph belongs to the co-tree, that is, the complementary part of the spanning tree.

Definition 2. Let us consider a graph in which the directions of the arcs are ignored. Such a graph is called an undirected graph. If a co-tree edge is added to the spanning tree of such a graph, a unique sequence of edges, known as a circuit, is formed. This circuit consists of the co-tree edge whose terminal vertices v_i, v_j are connected in the spanning tree by a subset of edges in the tree. The circuit consists of the sequence of edges a_6, a_1, a_2, a_5. Each co-tree arc added to the spanning tree arcs in turn creates a unique circuit consisting of the co-tree arc and the arcs in the spanning tree.

The totality of circuits so formed constitutes a basis in the vector space of all circs.

Definition 3. Let $c = (x_1, x_2, \ldots, x_m)$ be a vector whose components x_i consist of zeroes or ones and where the index m represents the total number of edges in the

undirected graph. A vector c represents a circuit if $x_i = 1$ whenever a_i is in the circuit and $x_i = 0$ whenever a_i is not in the circuit. Such vectors form a vector space under componentwise modulo 2 addition. The vector space is called the space of circs.

Associated with the flowgraph in Figure V.A.2, we have the basis circuits

$$\begin{array}{c} \\ c_1 \\ c_2 \end{array} \begin{array}{c} \begin{array}{cccccc} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

The vector space of circs in this case consists of the modulo 2 sums

$$c_2 \oplus c_2 = (1 \oplus 1,\ 1 \oplus 1,\ 0 \oplus 0,\ 0 \oplus 0,\ 1 \oplus 1,\ 1 \oplus 1)$$
$$= (0,\ 0,\ 0,\ 0,\ 0)$$

$$c_1 \oplus c_2 = (1 \quad 0 \quad 1 \quad 1 \quad 1)$$

The total number of circs is 4 in this case, or more generally

$$2^M,$$

where

$$M = E - (V - 1) = E - V + 1$$

M is called the cyclomatic number, E is the number

of edges in the graph and V is the number of vertices.

The concept of circs remains unaltered if we consider the direction of the arcs in the circuit. The direction of the co-tree arc induces a direction in the circuit which is formed. By adding the co-tree arc a_6 in Figure V.A.2, the arcs a_1, a_2, a_5 agree with the induced direction and hence the corresponding vector is

$$c_2 = \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{matrix} \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

If a_6 were directed from v_1 to v_5 then the induced direction of the circuit would be opposite of arcs a_5, a_2 and a_1, hence the vector would be represented as

$$c_2 = \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{matrix} \begin{pmatrix} -1 & -1 & 0 & 0 & -1 & 1 \end{pmatrix}$$

Although the signs are unimportant in modulo 2 arithmetic because $-1 = 1$, the signs become important when the direction of flows is considered.

The relationship between the flow variables could have been obtained by applying Kirchhoff's laws of flow into a vertex is equal to flow out of a vertex at n-1 vertices. We have chosen to express the relationship by using the concept of circs. The result provided in detail in [12] and [16] is as follows.

Theorem 1. The tree flow variables can be expressed

in terms of the co-tree flow variables by using the coefficients matrix whose columns correspond to the signed incidence vectors representing the basis circuits.

$$\begin{bmatrix} y_1^t \\ \\ y_2^t \\ \cdot \\ \cdot \\ \cdot \\ y_{n-1}^t \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & & \\ x_{31} & & & \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ x_{n-1,1} & \cdot & \cdot \cdot & x_{n-1,M} \end{bmatrix} \begin{bmatrix} y_1^c \\ \\ y_2^c \\ \cdot \\ \cdot \\ \cdot \\ y_M \end{bmatrix}$$

where

$$x_{ik} = \begin{cases} 0 & \text{if tree arc } i \text{ is not in circuit } k \\ 1 & \text{if tree arc } i \text{ is positively incident in circuit } k \\ -1 & \text{if tree arc } i \text{ is negatively incident in circuit } k. \end{cases}$$

In the flowgraph in Figure V.A.2 the relationship of the spanning tree variable is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_4 \\ y_6 \end{bmatrix}$$

We note that in the first equation above

$$y_1 = y_6$$

which states that the flow cut from vertex v_1 is equal to

the flow intc vertex v_1.

Cne can use the seccnd Kirchhcff's law for potentials to relate the co-tree variables to the tree variables. In any circuit in the network the sum of the potential differences is zero.

Theorem 2. The co-tree potential variables can be expressed in terms of the tree potential variables by using the transpcse of the matrix X in Theorem 1.

$$\begin{bmatrix} x_1^c \\ x_2^c \\ x_M^c \end{bmatrix} = \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n-1,1} \\ x_{12} & x_{22} & \cdots & x_{n-1,2} \\ x_{1M} & x_{2M} & \cdots & x_{n-1,M} \end{bmatrix} \begin{bmatrix} x_1^t \\ x_2^t \\ x_{n-1}^t \end{bmatrix}$$

4. Problem Formulation and Solution

We shall now formulate and solve the three abstractly similar problems.

In the problem in Figure V.A.1(a) we have the following relationships for each of the two terminal circuit elements.

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_5 \end{pmatrix} = \begin{pmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 \\ 0 & 0 & R_3 & 0 \\ 0 & 0 & 0 & R_5 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_5 \end{pmatrix} \qquad (1)$$

By Kirchoff's laws, flow variables are related by

$$\begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I_4 \\ I_6 \end{pmatrix} \qquad (2)$$

and the potential variables are related by

$$\begin{pmatrix} v_4 \\ v_6 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_5 \end{pmatrix} \qquad (3)$$

Premultiplying (1)

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_5 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 \\ 0 & 0 & R_3 & 0 \\ 0 & 0 & 0 & R_5 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ I_5 \end{pmatrix}$$

Using (3) on the left and (2) on the right

$$\begin{pmatrix} v_4 \\ v_6 \end{pmatrix} = \begin{pmatrix} R_2 + R_3 & R_2 \\ R_2 & R_1 + R_2 + R_5 \end{pmatrix} \begin{pmatrix} I_4 \\ I_6 \end{pmatrix}$$

Depending on what information is prescribed, the problem can be easily solved. If we know $I_4 = I(t)$ and $v_6 = V(t)$, then v_4 and I_6 can be determined in terms of $I(t)$ and $V(t)$ as follows:

$$v_4 = (R_2 + R_3)\ I(t) + R_2 I_6$$

$$v(t) = R_2 I(t) + (R_1 + R_2 + R_5) I_6$$

$$I_6 = \frac{v(t) - R_2 I(t)}{R_1 + R_2 + R_5}$$

$$v_4 = (R_2 + R_3)\ I(t) + R_2 \left(\frac{v(t) - R_2 I(t)}{R_1 + R_2 + R_5} \right)$$

In the hydraulics problem in Figure V.A.1(b) precisely the same results would be obtained if the flow rate F(t) and the pressure difference P(t) were prescribed.

In programming problems we typically know the flow values. For example, if we execute the program in Figure V.A.1(c) once, then $y_6 = 1$. Because the loop is executed exactly four times $y_4 = 4$ and hence:

$$\begin{bmatrix} x_4 \\ x_6 \end{bmatrix} = \begin{bmatrix} T_2 + T_3 & T_2 \\ T_2 & T_1 + T_2 + T_5 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

Here x_4 represents the total execution time in the interior loop, whereas x_6 represents the execution time in the exterior loop.

5. The Total Execution Time

The total execution time can now be expressed in terms of the linearly independent flow variables.

$$\text{TOTAL TIME} = \sum_{i=1}^{6} x_i = \sum_{i=1}^{6} T_i y_i$$

$$= (T_1\ T_2\ T_3\ T_5) \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_5 \end{pmatrix} + (T_4\ T_6) \begin{pmatrix} y_4 \\ y_6 \end{pmatrix}$$

$$= (T_1\ T_2\ T_3\ T_5) \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y_4 \\ y_6 \end{pmatrix} + (T_4\ T_6) \begin{pmatrix} y_4 \\ y_6 \end{pmatrix}$$

$$= (T_2 + T_3,\ T_1 + T_2 + T_5) \begin{pmatrix} y_4 \\ y_6 \end{pmatrix} + (T_4\ \ T_6) \begin{pmatrix} y_4 \\ y_6 \end{pmatrix}$$

$$= (T_2 + T_3 + T_4,\ T_1 + T_2 + T_5 + T_6) \begin{pmatrix} y_4 \\ y_6 \end{pmatrix}$$

$$= (T_2 + T_3) \cdot v_4 + (T_1 + T_2 + T_5) \cdot v_6$$

$$= (T_2 + T_3) \cdot 4 + (T_1 + T_2 + T_5) \cdot 1$$

more generally

$$ET = \sum_{i=1}^{n-1+M} T_i Y_i = [T_1 \cdots T_{n-1}] \begin{bmatrix} y_1^t \\ \vdots \\ y_{n-1}^t \end{bmatrix} + [T_n \cdots T_{n-1+M}] \begin{bmatrix} y_1^c \\ \vdots \\ y_M^c \end{bmatrix}$$

$$= [T_1 \cdots T_{n-1}] \begin{bmatrix} x_{11} & x_{12} \cdots x_{1M} \\ x_{21} & x_{22} \\ \vdots & \\ x_{n-1,1} & \cdots x_{n-1,M} \end{bmatrix} \begin{bmatrix} y_1^c \\ \vdots \\ y_M^c \end{bmatrix}$$

$$+ [T_n \cdots T_{n-1+M}] \begin{bmatrix} y_1^c \\ \vdots \\ y_M^c \end{bmatrix}$$

$$= \left\{ \left[\sum_{i=1}^{n-1} T_i x_{i1}, \sum_{i=1}^{n-1} T_i x_{i2}, \cdots, \sum_{i=1}^{n-1} T_i x_{iM} \right] + [T_n, T_{n+1}, \cdots, T_{n-1+M}] \right\} \begin{bmatrix} y_1^c \\ \vdots \\ y_M^c \end{bmatrix}$$

$$= \left[\sum_{i=1}^{n-1} T_i x_{i1} + T_n \right] y_1^c + \left[\sum_{i=1}^{n-1} T_i x_{i2} + T_{n+1} \right] y_2^c$$

$$+ \left[\sum_{i=1}^{n-1} T_i x_{iM} + T_{n-1+M} \right] y_M^c .$$

6. Summary

This report shows that discrete systems analysis is analogous and applicable to the analysis of computer programs. The traditional view of flowgraphs [17] associates vertices with blocks of code and arcs with the flow of control. That view does not lead to the correspondence exhibited in this paper.

We would like to also point out the similarity of the flowgraph analysis to the problems encountered in network flows in which a unit flow through the arc is associated with the cost [7].

Because both discrete systems analysis and network flow problems are highly developed areas in engineering and science, the tools and techniques developed in these areas become applicable to computer program analysis.

Program analysis software development which automates the analysis of programs using the discrete systems analysis techniques described here is a task which we hope to encourage by this segment of the report.

B. DATA FLOWGRAPHS

1. Introduction

The basic inadequacy in program documentation in both flowchart and programming languages form is that both forms concentrate on how a problem is being solved rather

than what the problem is.

If we design a multiplier either in hardware, firmware or software, it is essential to know what the problem is. In proving correctness of programs we must first state what the problem is before we can verify that our method of solving it is correct.

There are three major ways in which we state what the problem is:

(1) by the use of formulas;

(2) by drawing boxes and joining them with lines;

(3) by the use of decision tables.

Formulas have the advantage that we have learned to manipulate them in order to derive equivalent expressions which serve to simplify the problems. Also, formulas can directly and easily be communicated to computers if compilers are available.

Formulas have disadvantages when they extend over several lines or several pages, or when a large collection of formulas are used to describe a problem. Computer designers have used both formulas and drawings to document the designs. IBM is a major user of drawings. Design documents which describe computer hardware are documented with box/line drawings.

Box/line drawings have both advantages and disadvanages. The major disadvantages of these documents are that we must relearn to manipulate the drawings in order to simplify them and that we have no compilers for graphics. Generally human transcribers are used to translate graphic

pictures into notation understandable to the computer. The computer redraws the pictures usually losing positional integrity for both boxes and lines and thus losing some (possibly important) information.

The major advantages of box/line drawings are that data flow can be explicitly shown across formulas, information is not constrained to lines (one-dimensional), and levels of abstraction are conveniently achieved by functionally labelling boxes and lines.

This segment formalizes the concept of the box/line drawings. Such drawings form a bipartite directed graph, bi-digraph, which we shall also call a data flowgraphs.

A program flowchart also corresponds to a graph construct called a flowgraph. To each execution sequence in the flowgraph corresponds a data flowgraph which describes at a chosen level of abstraction what happens to the data.

A similar data flow analysis is carried out by Allen and Cocke [1], although both their control flow and data flow are differently conceived and applied.

A control complexity analysis which makes uses of a control flow graph and introduces a complexity measure, the cyclomatic number of the control flowgraph, by McCabe [17], has some similarities to our concept of the flowgraph. The flowgraph in the in this section is abstractly the same as graphs used in circuit theory, discrete systems analysis, and cost oriented network flow problems.

Shneiderman et al [21] showed experimentally that flowcharting has little value in increasing programmer productivity. The value we see in flowcharts or flowgraphs is related to automated computer analysis of algorithms,

both the execution time analysis and the data flow analysis.

In the previous segment the flowgraph corresponding to the flowchart was defined and shown to be abstractly identical to similar graphs encountered in discrete system's analysis. In Section 2 the data flowgraph corresponding to an execution sequence in the flowgraph is defined. Section 3 illustrates the usefulness of the concept by applying it to the analysis of an airborne real time tactical system (A6-E). Section 4 is a summary of what this segment contains.

2. Data Flowgraphs

Although the flowchart analysis gives us an effective way of determining execution times and the complexity of programs, it does not give much information on the independence of processes or how processes can be executed simultaneously without interfering with each other. The concept of data flowgraphs helps to graphically display what happens to data, how data is transformed, and how one can partition the process into subprocesses with a minimal need of data transfers.

We illustrate the ideas first before we formalize the concepts. Referring back to Figure V.A,1(c) and Figure V.A.2 consider the graph consisting of arcs a_i and vertices v_i. Each arc of this graph corresponds to an instruction sequence which carries out a computation on some input data. For example, arc a_1 corresponds to a typical assembly language instruction sequence.

```
LDA 0

STA SUM

LDA 1

STA I
```

We associate with each data item a vertex c_i and with each operation another vertex o_j, Figure V.B.1 in a manner used by digital circuit designers ever since computers were first designed and manufactured. We connect the data item to the operation which uses the data item as an input by an arc directed into the operation vertex. The output data item or items are connected to the operation by arcs directed away from the operation vertex. The graphs resulting from carrying out this association with flowgraphs are graphs which contain two types of vertices, where arcs connect data vertices to operation vertices and where operation vertices in turn are only connected to data vertices. Such graphs are known as bipartite directed graphs, or bi-digraphs [2].

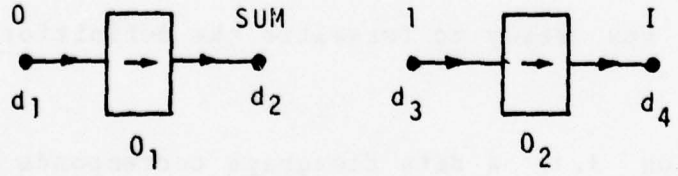


Fig.V.B.1 Two Components of the Graph Corresponding to the Flowgraph Arc a_1.

For each arc in the flowgraph we construct a corresponding bi-digraph. We note that control operations such as branch or halt instructions do not alter any data and hence do not require a correspondent bi-digraph.

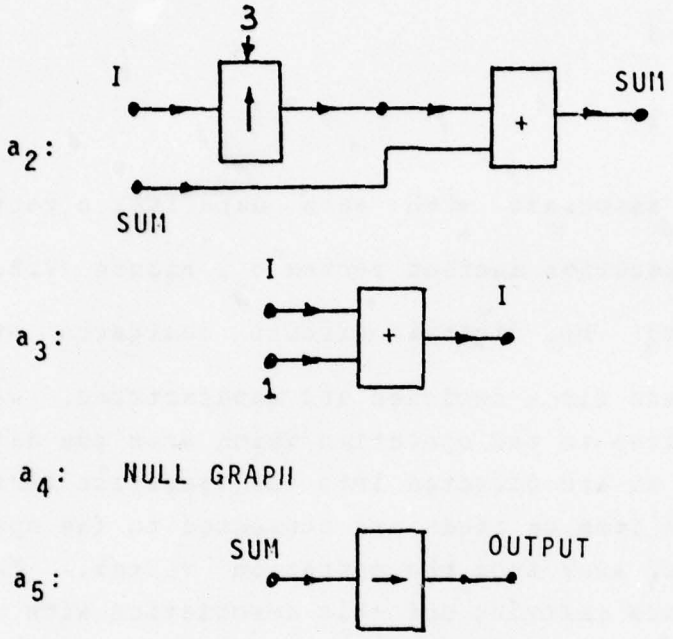


Fig. V.B.2 Components of the Data Graph.

We next determine an execution sequence or all execution sequences for a given flowgraph.

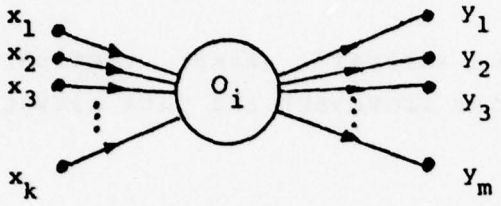
We are now ready to formalize the definition of a data graph.

Definition 3.1. A data flowgraph corresponds to an execution sequence in a flowgraph as follows. Let

$$S = (a_1, a_2, \ldots, a_n)$$

be an execution sequence in a flowgraph. To each arc a_i,

there corresponds a mapping of input variables $x_1, x_2, \ldots, x_k$ into output variables $y_1, y_2, \ldots, y_m$. This functional relationship is denoted by some operation c_i and indicated as a subgraph.



If one or more of the input variables of o_i is identical to an output variable of some other operation o_j, then we identify such variables by the same vertex in the data flowgraph. Similarly, if there is a common input or output variable to one or more operations o_j, we identify such variables by the same vertex in the data flowgraph. If this procedure is successively carried out for each arc in the execution sequence S, the resulting graph is a data flowgraph of S.

We have given so far very simple examples to illustrate the idea of a data flowgraph. It is easy to see that in a complex program there are large numbers of execution sequences and hence data flowgraphs. Therefore this method of analysis would become so complex that it becomes useless.

Fortunately the idea of a data flowgraph lends itself very naturally to levels of abstraction. We can illustrate this with the example which comes from the A6-E

tactical system.

3. An Illustrative Real Time System

a. Data Flowgraph Construction

An attack aircraft (A6-E) tactical system is used to illustrate the flowchart and data flowgraph analysis techniques.

Figure V.E.3 illustrates he top level flowchart which describes the system's operation. After a hardware checkout and initialization programs have been executed, the program goes into the infinite loop described by the flowchart. In this system the executive program is very simple: a sequence of tasks is executed without a set of priorities. The task is bypassed whenever there is no need to execute it. The analog inputs from the sensors are sampled periodically based on a real time system's clock.

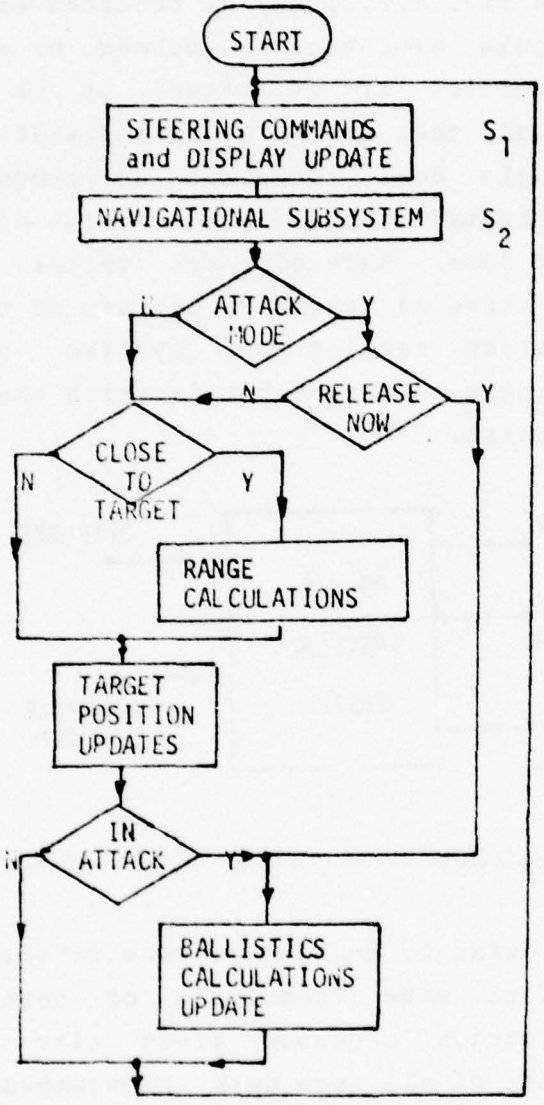


FIG.V.B.3 A6-E TACTICAL PROGRAM FLOWCHART

One execution of the infinite loop, that is, the transition from the control point above Steering Commands..." to the control point below Ballistics Calculations..." in Fig. V.B.3, may be regarded either as a set of all possible execution sequences, or as a single transition of control. If we regard it as a single transition of control, then we can represent what happens to the data with a single data flowgraph as shown in Fig. V.B.4. This representation corresponds to the overall view of what the program does. Each data set vertex represents data items which serve as inputs or outputs of the system, whereas the operation carried out by the system is represented by a single vertex, a box in which the operation is described in English.

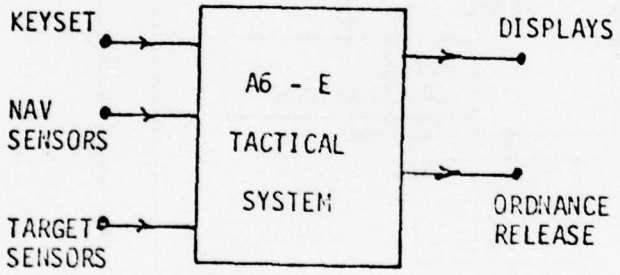


FIG. V.B.4 DATA FLOWGRAPH OF THE A6-E TACTICAL SYSTEM

If we wish to consider a more detailed view of the operation under the same transition of control, then each distinct execution sequence gives rise to a data flowgraph and the set of all such data flowgraphs describe in more detail what the single flowgraph expresses in Fig. V.B.4.

The Navigational Subsystem consists of eight sequential steps in Fig. V.B.5. the first step is entitled Air Data Quantities-1 in Fig. V.B.6. This flowchart gives the finest level of detail and enables a programmer tc translate the flowchart to a higher level language or an assembly language program.

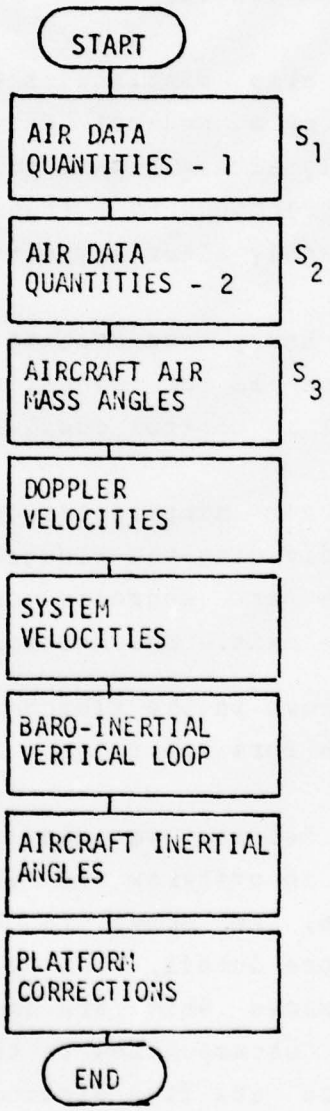


FIG. V.B.5 FLOWCHART OF THE NAVIGATIONAL SUBSYSTEM

Corresponding tc the flowchart in Fig. V.E.6 we construct a flowgraph in Fig. V.B.7. In the flowgraph we have distinguished between the vertices from which more than cne arc issues. The latter vertices are symbolized by a small diamord indicating that several execution sequences emanate from that vertex.

We also distinguish between twc types cf arcs: cne (symbclized by a square) corresponds tc a statement which permanently alters a data value; the cther (symbclized by an arrow) corresponds to a transition in control which dces not permanently alter any data values.

Ihe heavy arcs constitute a spanning tree cf the flowgraph which is of significance in execution time analysis as well as control complexity analysis.

Further simplification of the analysis can be cbtained by subdividing the flowgraph into sc-called ccntrol segments, which are segments of a program with a single entry and single exit. The control segment from v_0 to v_5 in Fig. V.B.7 is shown in the flowchart form in Fig. V.B.8, and in the flcwgraph form inV.B.9(a).

As before, we can generate a data flowgraph which describes an overview of what takes place in the control segment, as shown in Fig. V.E.10. If we are interested in more detail, then we look at all possible execution sequences which are desribed as a rooted tree in Fig. V.B.9(b). Corresponding to the six pcssible execution sequences, there are five distinct statement sequences and their corresponding data flowgraphs in Fig. V.B.11.

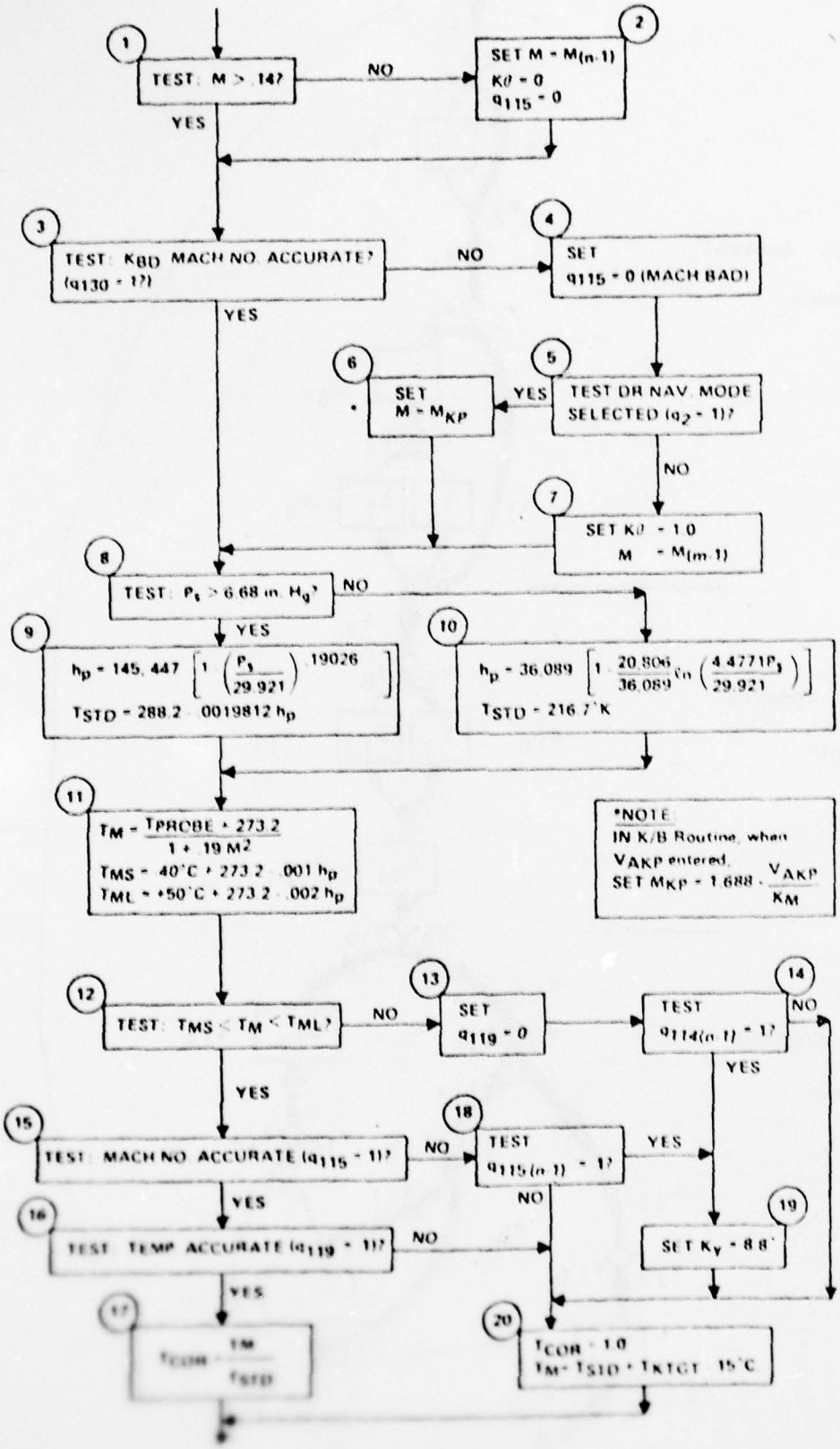


FIG. 5.3 FLOWCHART OF AIR DATA QUANTITIES-1

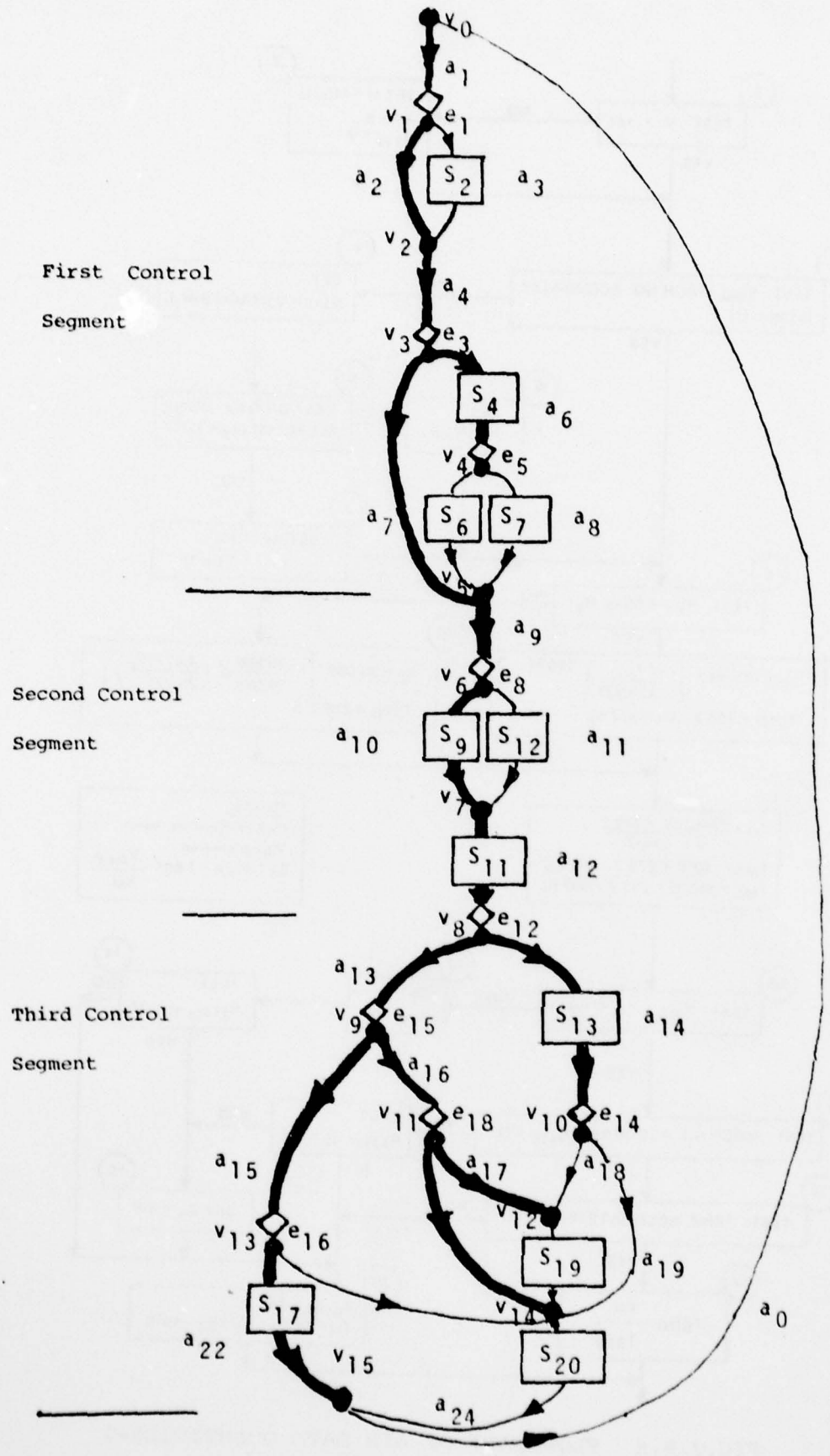


FIG. V.B.7 FLOWGRAPH OF AIR DATA QUANTITIES-1

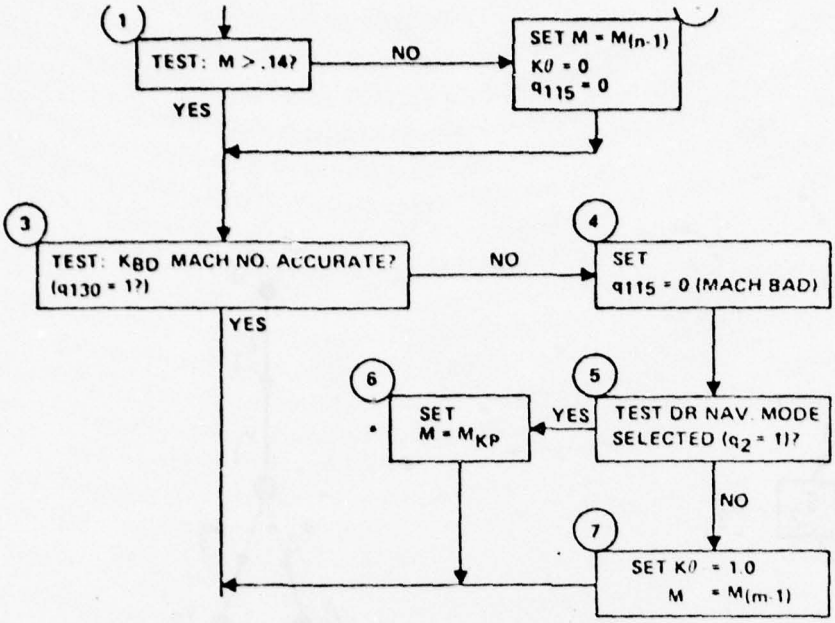


FIG.V.B.8 FIRST CONTROL SEGMENT OF AIR DATA QUANTITIES-1

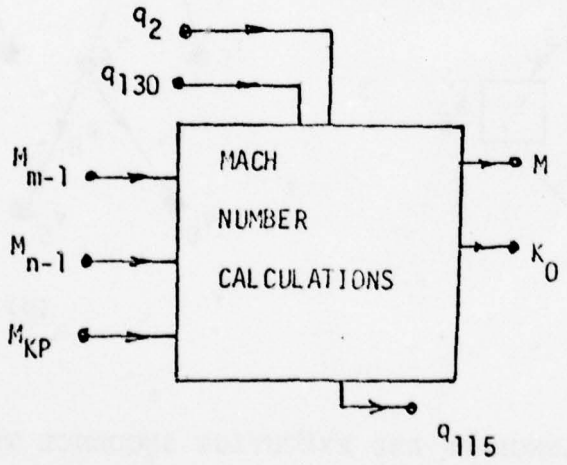


FIG.V.B.10 DATA FLOWGRAPH OF FIRST CONTROL SEGMENT

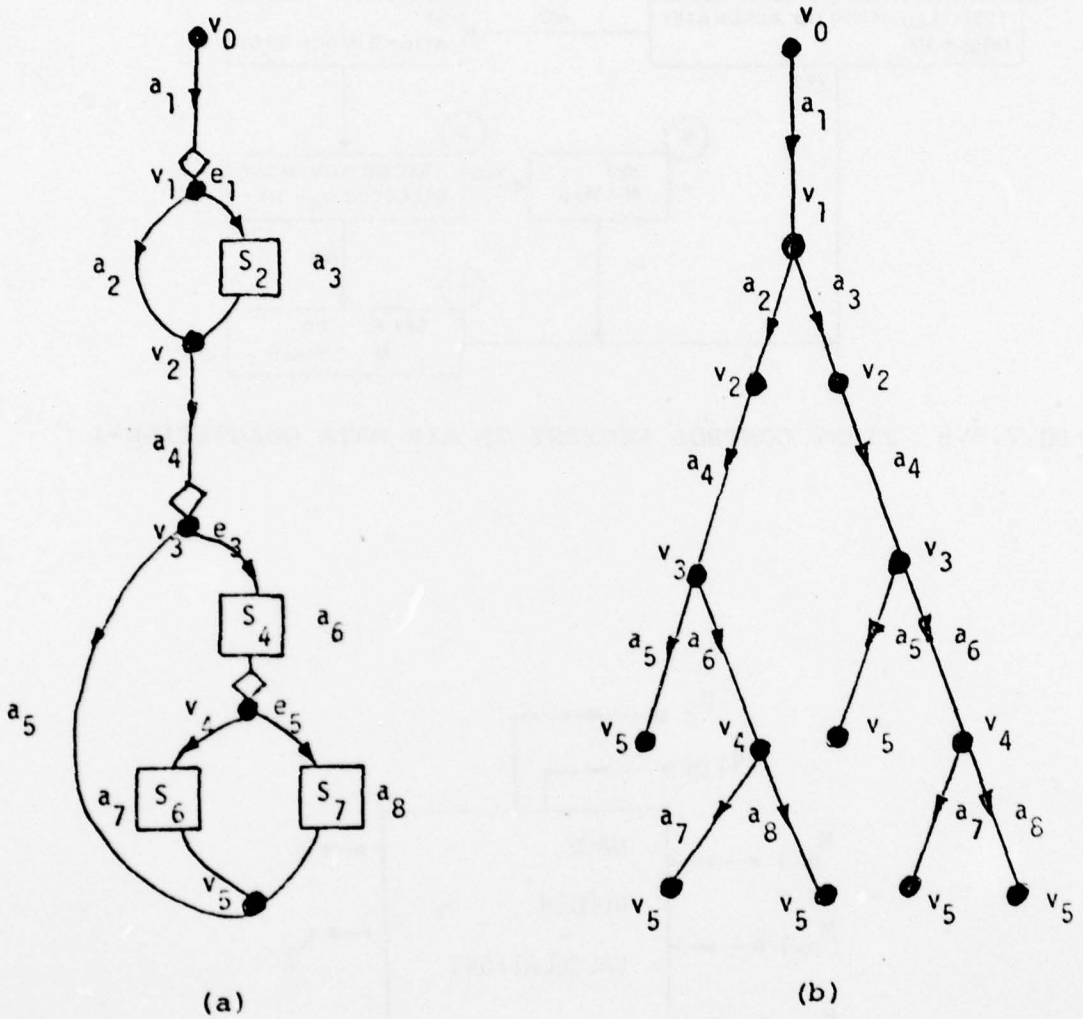


FIG.V.B.9 FLOWGRAPH AND EXECUTION SEQUENCE TREE

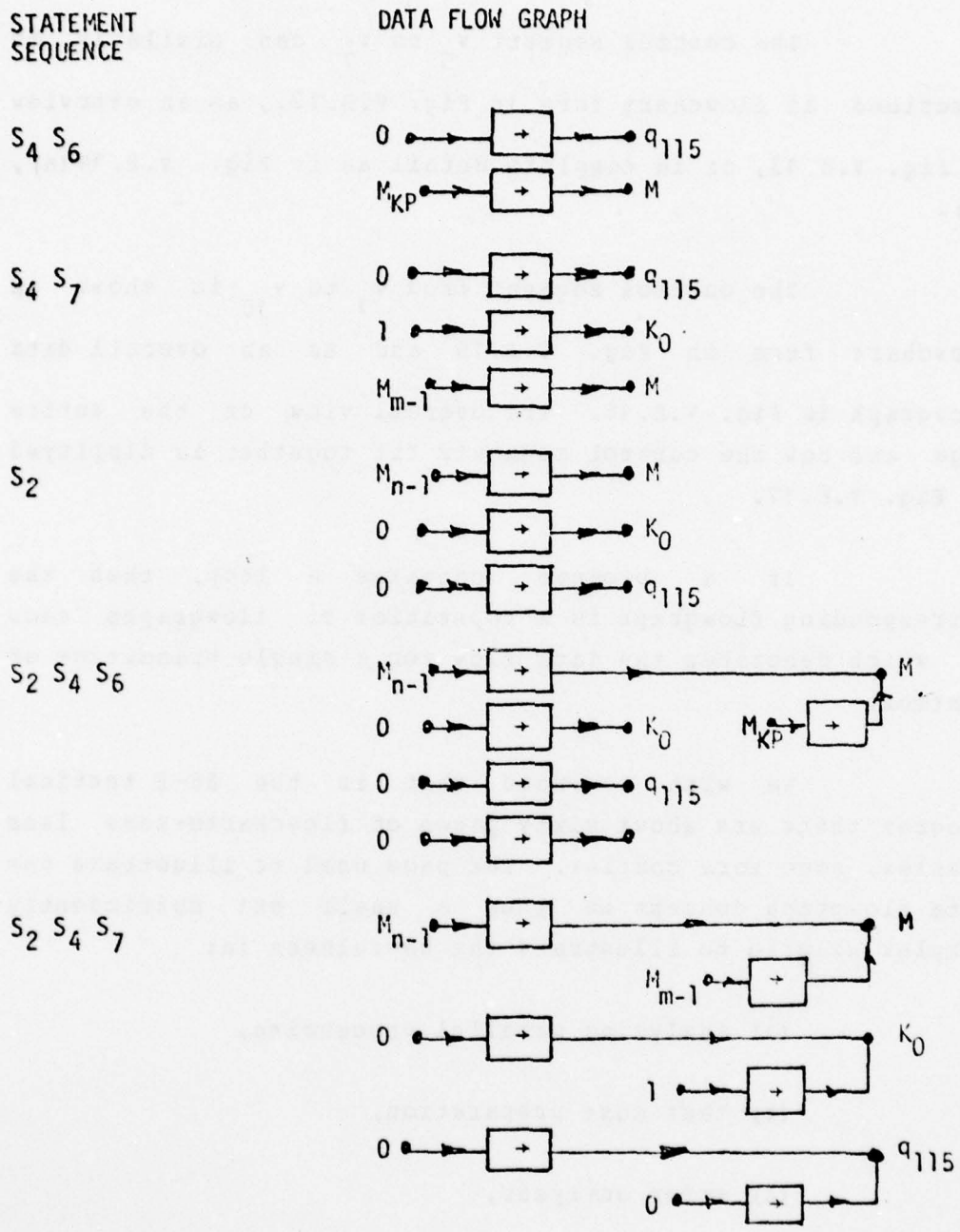


FIG. V.B.11DATA FLOWGRAPHS CORRESPONDING TO FIVE STATEMENT SEQUENCES

The control segment v_5 to v_7 can similarly be described in flowchart form in Fig. V.B.12., as an overview in Fig. V.B.13, or in complete detail as in Fig. V.B.14(a), (b).

The control segment from v_7 to v_{15} is shown in flowchart form in Fig. V.B.15 and as an overall data flowgraph in Fig. V.B.16. The overall view of the entire page and how the control segments fit together is displayed in Fig. V.B.17.

If a program contains a loop, then the corresponding flowgraph is a repetition of flowgraphs each of which describes the data flow for a single transition of control.

We wish to note that in the A6-E tactical program there are about sixty pages of flowcharts-some less complex, some more complex. The page used to illustrate the data flowgraph concept is thus a small but sufficiently complex example to illustrate the usefulness in:

(a) analyzing parallel processing,

(b) test case preparation,

(c) error analysis,

(d) program verification.

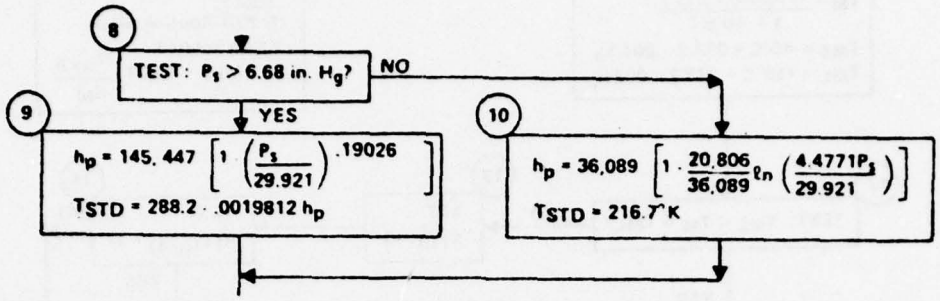


Fig.V.B.12 FLOWCHART OF SECOND CONTROL SEGMENT

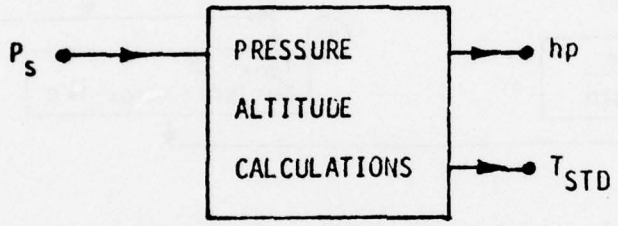


Fig.V.B.13 SECOND CONTROL SEGMENT'S DATA FLOWGRAPH

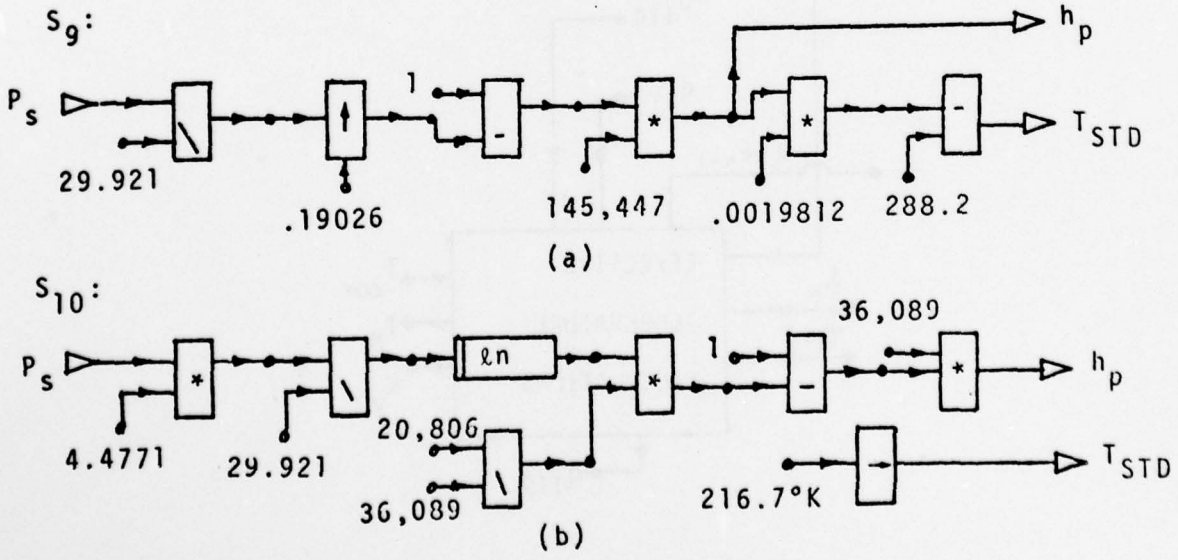


Fig.V.B.14 DETAILED SECOND CONTROL SEGMENT'S DATA FLOWGRAPH

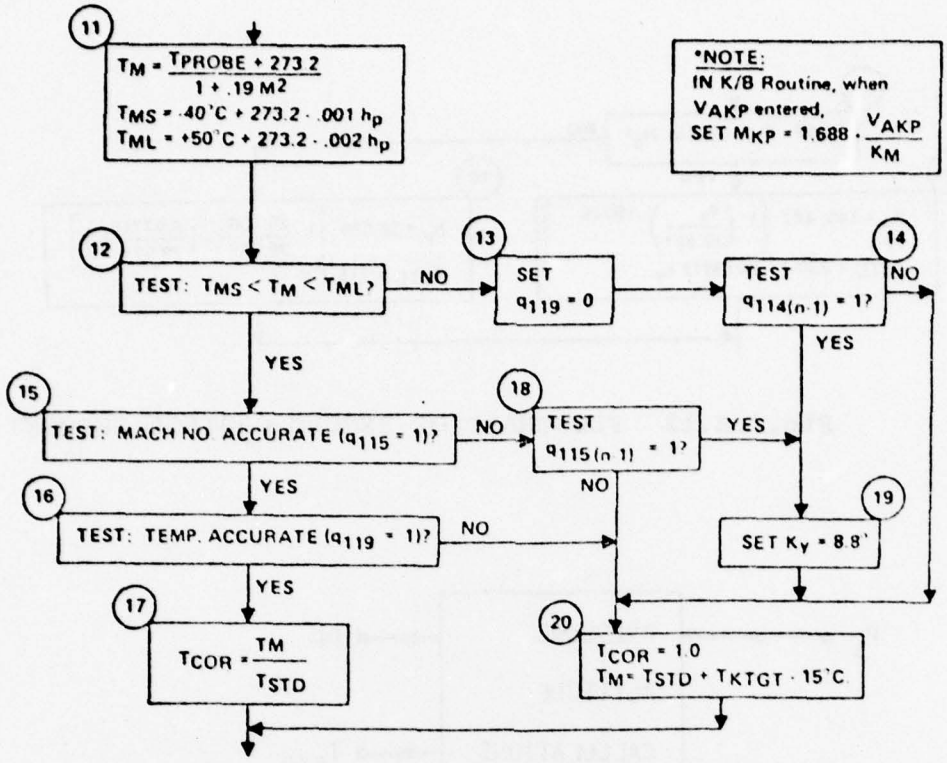


FIG.V.B.15 FLOWCHART OF CONTROL SEGMENT THREE

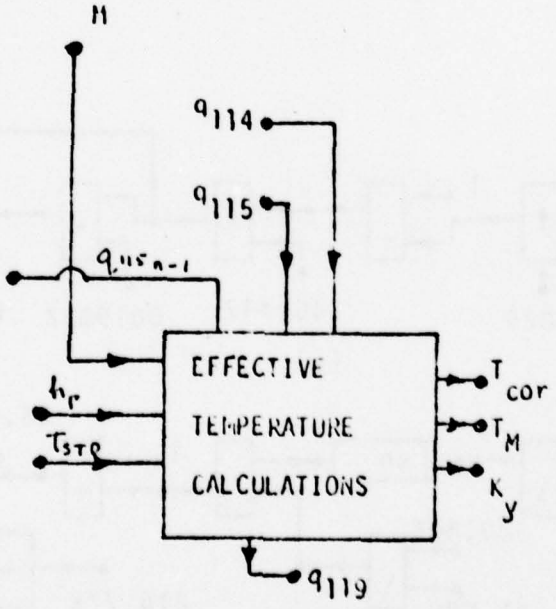


FIG. V.B.16 OVERVIEW DATA FLOWGRAPH OF SEGMENT THREE

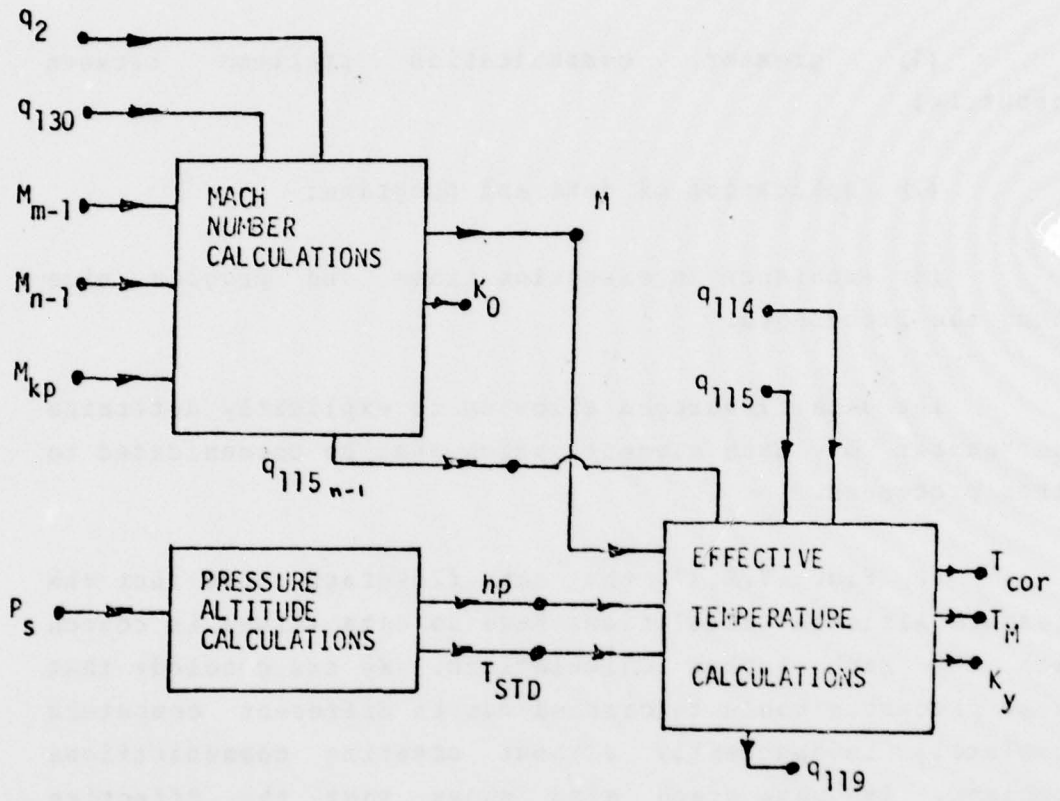


FIG. V.B.17 DATA FLOWGRAPH DESCRIBING THREE CONTROL SEGMENTS OF AIR DATA QUANTITIES-1

4. Analysis of Parallel Processing

The main concerns in distributing a process among several computers is the resulting inefficiencies introduced by the distribution:

(1) greater communication problems between computers;

(2) duplication of data and programs;

(3) imbalance in execution times and program size among the processors.

The data flowgraphs allow us to explicitly determine the number of data elements which must be communicated to other processes.

In Fig. V.B.17 the data flowgraph shows that the Pressure Altitude Calculations have no data values in common with the Mach Number Calculations. We can conclude that these processes could be carried out in different computers completely independently without creating communications problems. The same graph also shows that the Effective Temperature Calculations use two data values generated by Mach Number Calculations and two data values from Pressure Altitude Calculations. If a single computer must solve the problem, then eight data values are used as inputs and five values are used as outputs. If two computers are to solve the problem, and if Mach Number Calculations are done in one computer and the Pressure Altitude and Effective Temperature Calculations are done in the other, then the communication problems are increased by the two data values which must be communicated. Effective Temperature Calculations may not be

started before Mach Number Calculations are ready. This causes timing problems in addition to communication problems; however, data flowgraphs allow the analysis of both problems.

Data flowgraphs reduce the distributed system design problem to a graph partitioning problem with side constraints. Several effective algorithms exist for solving that problem. Reference [2] reviews the published literature on graph partitioning. The side constraints may be taken to be program size and maximal execution time. The problem then becomes: Partition the graph into a minimal number of partition elements so that the number of arcs cut is minimized, and the bounds on program size and maximal execution times are satisfied. The problem is completely analogous to the hardware partitioning problem for creating mother-boards in computer design. the mother-board design problem had to satisfy input-output constraints which permitted only a certain maximum number of input-output connections for each mother-board, as well as a certain maximum number of daughter-cards on each mother-board.

5. Test Case Preparation

As all of us who write computer programs know, a program is seldom correct when first executed. Therefore it is safe to assume that programs contain errors, and we would like to generate test cases which are:

(1) exhaustive enough to discover all errors;

(2) small enough in number so that we can effectively test the cases;

(3) suggestive of what must be done to make

corrections.

Computer programming is in many ways analogous to a lengthy derivation in solving calculus problems. The way in which one gains confidence in a lengthy derivation is by looking in the answer section to see if the result obtained agrees with the one given in the book. If the results agree, then we usually are satisfied. If not, then we try to establish equivalence. Failing that, we check algebraic manipulations, signs, differentiations, integrations, etc. until an error is discovered. If we work on an even-numbered problem which does not have an answer in the back of the book, we resort to different schemes. If we trust a friend we call him to find out what result he obtained. If the results do not agree, we check back to see when the results first disagreed.

In programming we use similar techniques. We write a program in some higher level language such as FORTRAN. If we write a special purpose routine to evaluate a trigonometric function, we compare our results with the corresponding library routine. How many test values do we use? If we have a polynomial of degree n, then theory tells us that n+1 points are sufficient to establish identity. By using a random number generator to generate a set of n+1 random numbers in the interval of interest and by finding that the results agree with sufficient accuracy, we become confident that no further errors exist. We remain confident until someone discovers that for a particular case the results are incorrect. We fix the program and add these particular cases to our repertoire of test cases. Usually the endpoints of finite intervals, zero value, or undetected underflows and overflows are a cause of trouble. How does the data flowgraph concept help us generate test cases?

We observed in the previous segment that the control

segment v_0 to v_5 had six different execution sequences and five different statement sequences. Each statement sequence gives rise to a different function. Therefore we must prepare at least five different data sets, one for each function. As shown in Fig. V.B.11 each function is a constant function, and hence theoretically, one data value for each data set would be sufficient to check identity to a previously computed constant function.

If one considers the flowgraph in Fig. V.B.7 in its entirety, then the total number of execution sequences between v_0 and v_{15} is 72. The number of execution sequences may be calculated by a vertex labelling algorithm which labels each vertex with the number of distinct access paths from the entry point.

Because several execution sequences give rise to the same statement sequence, only 50 distinct statement sequences exist. If each statement sequence yielded a unique function, we would have to construct at least 50 different data sets, each data set containing at least one set of values. In the A6-E tactical program there are 60 pages of flowcharts of control complexity with about an average of 20 statement sequences each. Therefore, there are approximately $(20)^{60}$ statement sequences in the entire program. If each statement sequence gave rise to a different function, we would have to generate $(20)^{60}$ data sets, each containing at least one set of data values. Clearly testing the program would become impossible.

The data flowgraph corresponding to the control segment (i.e. a program segment with a single entry and a single exit) from v_0 to v_5 in Fig. V.B.10 is disconnected

from the data flowgraph for the next control segment. This means that the two functions are independent and can be tested independently from each other.

Theoretically, the first control segment requires 5 data sets, the second 2 data sets. Altogether 5 + 2 data sets are necessary instead of 5*2 = 10.

The third control segment is a rational function containing 5 statement sequences. Hence 5 data sets would need to be constructed to test out that function.

In order that two rational functions be identical,

$$R1/R2 = Q1/Q2$$

it is sufficient to form the product polynomial,

$$R1 * Q2 = Q1 * R2$$

If the two product polynomials are equal for all values except possibly the points at which the denominators vanish, then we can conclude that the rational functions are identical. In the above example, four data values per data set would be sufficient. Thus 5 + 2 + 5 = 12 data sets are sufficient to test the identity of the functions calculated in the flowgraph in Fig. V.B.7 instead of the 50 data sets estimated on the basis of the statement sequences in the flowgraph.

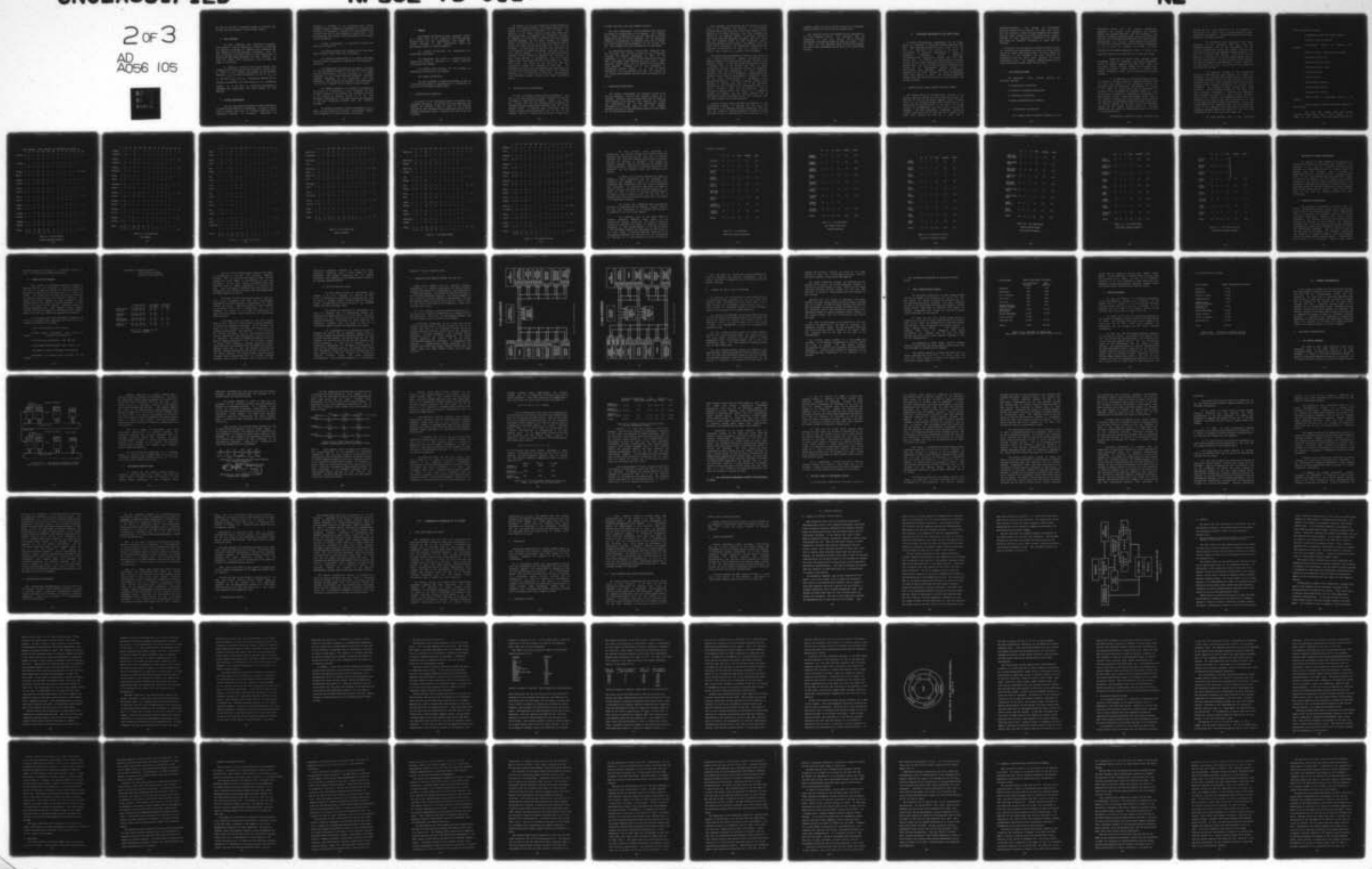
The flowchart page used in this example has average complexity. If we assume that the complexity of each page is on the average similar to this page and that data flowgraphs exhibit the same degree of independence as they do on this page, then only

$$60 * 12 = 720$$

data sets need to be constructed instead of $(20)^{60}$. Clearly

AD-A056 105 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 1/3
A STUDY OF ALTERNATIVES FOR VSTOL COMPUTER SYSTEMS.(U)
APR 78 U R KODRES , J D BUTTINGER
UNCLASSIFIED NPS52-78-001 NL

2 OF 3
AD
A056 105



720 data sets would be a reasonable number to construct even if each data set contained 10 sets of data values.

6. Error Analysis

The data flowgraphs lend themselves to numerical error analysis. Dorn an McCracken [5] present an elementary but complete treatment of roundoff error propagation for both fixed-point and floating-point arithmetic. They use data flowgraphs (or process graphs in their terminology) to develop the error-bound estimates for each function. We shall not repeat their presentation here but refer the reader to their book.

In addition to numerical error-bound analysis, data flowgraphs give strong indications of possible trouble spots or errors in the program. For example, in the statement sequence S_2 S_4 S_7 in Fig.V.B.1 variable M is set to M_{n-1} in S_2 and reset to M_{m-1} in S_4 S_7. Although the function may be correctly calculated, it is done clearly in an inefficient fashion. It is not hard to reconstruct the program to eliminate this inefficiency and still compute the same function.

7. Program Verification

For real time tactical systems, program verification is particularly difficult to accomplish. In order to verify that a program does what is intended, there must be an unambiguous statement of intention. Frequently the

statement of intention is in a flowchart form. Because flowcharts imply a sequence of statements and a sequence of controls, the statement of intention not only specifies what is wanted, but also in what sequence it should occur. This over-specifies the program, causes inefficiency, and removes useful flexibility.

Program verification, as described by Hantler and King [10], consists of:

(1) stating formally what assumptions are made about input variables before entry into a procedure;

(2) asserting algebraically or by logical statements what the output variables will contain when the procedure is completed;

(3) symbolically executing the program as described in some programming language to show that the results agree with the specification. The data flowgraph is a graphic expression of one or more algebraic statements. Therefore algebraic statements can be interpreted into data flowgraphs. This allows us to construct a data flowgraph or a set of data flowgraphs for the specification statement.

The symbolic execution of a program is equivalent to the process of constructing a data flowgraph for each execution sequence in the program. The verification process is the process of checking whether or not the data flowgraphs corresponding to the specification are equivalent to the data flowgraphs obtained from the execution sequences.

An automated process of showing equivalence is by no means an easy process to construct. Conceptually, however, data flowgraphs will enhance program verification.

8. Summary

This segment introduces the data flowgraph concept as a useful tool in the analysis of real time systems. Although there are more applications, this segment has focused attention on four applications which are particularly useful in systems analysis:

(1) process partitioning into subprocesses for distributed processing;

(2) generating test cases to establish that the program under design is identical to a known, correctly functioning test program;

(3) numerical error analysis and the analysis of inefficiences occurring in the program;

(4) program verification.

The data flowgraph is useful conceptually as well as in the practical domain of implementation of algorithms to carry out the automated analysis tasks.

C. EXECUTION TIME ESTIMATING

In Section V.A the theoretical aspects of execution time estimating were given. An analysis tool which generates the formula for execution times in terms of the linearly independent flow variables could be developed as part of the program compilation process. Presently this tool is not yet available.

Our analysis for the A6-E operational flight program was carried out by manual methods. The entire flowchart of the program is a collection of flowcharts one per page. The majority of the pages contain a single entry point and a single exit point. By looking at the operations indicated by formulas, it is not hard to determine among all possible execution sequences the one which consumes the maximum amount of execution time. It is that one which was used to estimate the worst case execution times for each page of flowchart. An upper bound for the worst case execution time can be obtained by summing the maximal execution times of each program segment. It may be that such an execution sequence is never realized in practice and therefore the upper bound measure is pessimistic. This is a way of insuring that the program execution never exceed the upper bound value. The values in the tables for the A6-E operational program were generated in this way. The breakdown into fixed point and floating point operations was chosen because floating point hardware units are presently available for LSI computers at a low cost. Using floating point arithmetic enhances accuracy and makes programming easier.

D. PROGRAM AND DATA REQUIREMENTS

In order to determine program length requirements, each page of the A6-E operational program flowchart was used and the number of instructions estimated. Because the actual A6-E program does not use floating point arithmetic the estimated program length differs somewhat from the actual program length. A floating point instruction does not require shifting operations because this is done in the hardware processor. Consequently the actual program length

is about 20% longer than our estimate predicts.

The data requirements for our estimates were based on the number of variabes used in the programs. The floating point format is assumed to be a 32 bit form used by the IBM computers as well as the LSI computers. For the presently available LSI computers, the floating point unit is treated as an input-output device accessible on the systems data bus. The constants are also accounted for in the totals.

For distributed systems, it may be more efficient from the execution time and bus use point of view to duplicate constants and function subprograms in each of the processors. In our analysis we have assumed that this is done. Furthermore, each processor will also have a small operating system which also is duplicated in each single board computer. Although this is not necessary in distributed systems design, it is done to create independence of each processor on the bus. The failure of any single board computer will not cause the failure of the system.

E. PARTITIONING METHODOLOGY

The logical cohesiveness of a program is shown by the data flow analysis. The data flow analysis allows us to determine precisely these variables which must be shared by other segments of the program. This analysis was carried out painstakingly for the A6-E operational program. Development of software tools to make this analysis automatic and part of the compilation process is very useful for the partitioning process.

Each variable was considered and its occurrance on any page of the flowchart was recorded. If the variable occurred only on one page, once as an output variable and at least once as an input variable, then that variable was defined as an internal variable to the page. A group of pages which have many variables in common with each other and few variables which are used externally, form a logically cohesive program segment. For example, the nine pages which constitute the navigational module have 143 variables in total and 50 of these are external to the module. Such a module is a logically cohesive module.

Our partitioning methodology was based on the concept of this logical cohesiveness. Logical cohesiveness is a natural byproduct of a well written program. The data flow analysis allows us to discover logical cohesiveness even in a program which is not written with that in mind. Furthermore, logical cohesiveness reduces the number of variables which are transmitted to other processes. Therefore, the bus use is minimized, or equivalently, the parameter count passed to subprograms is minimized. There may be other considerations of importance in the partitioning process. If we wish to design systems which continue to function even when one of the processors malfunctions, then a simple way of accomplishing this is to place a process in more than one processor. This is analogous to having a pilot and copilot on a commercial airliner for safety. All the vital functions can be distributed so that the failure of any one processor will not cause failure of any vital function.

Program and data length also must be considered in the partitioning process. If programs and data are to be distributed, it is important that the program and data fit into the computer. Although expansion of memory is easily achieved with the LSI architectures, the access time to the

expanded memory is slightly greater and there is contention among processors for the use of the system's bus.

The execution time of the process must be considered as the most important factor. The processes must be distributed so that each process can be executed successfully a prescribed number of times per second. This of course is a very important aspect of real time systems. All of the above considerations are used in the partitioning process.

VI. FUNCTIONAL DESCRIPTION OF THE VSTOL SYSTEM

We develop the functional requirements for the VSTOL system by examining in detail the avionics system for the A6-E. Other existing systems A7-E, P3-C, S3-A, F-15, and F-18 are contrasted with respect to the A6-E. We characterize the functional requirements in terms of arithmetic complexity, control complexity, intercommunication requirement, program size, and data storage size. We identify the core elements of tactical systems, that is, these functional elements which all of the systems share. Two of the VSTOL applications fighter/attack version and the antisubmarine warfare version are functionally quite different. However, the functional similarity of the VSTOL versions to their corresponding fixed wing cousins is great. The essential difference between VSTOL and fixed wing aircraft is in the takeoff and landing controls.

A. OVERVIEW OF THE ATTACK AIRCRAFT TACTICAL SYSTEMS

The primary purpose of the Navy attack aircraft is to provide close air support to forces operating on land. The A6-E and A7-E are the presently employed attack aircraft which use an on-board computer based tactical system. We have chosen the tactical system of the A6-E as a model because its documentation was most easy to use for our analysis. The most significant functional difference between the A6-E and A7-E is that the A7-E is manned by the pilot alone, whereas the A6-E has a pilot and a

bombardier/navigator. Both systems use functionally identical sensors to help navigate and track the target. They are designed to carry the same armaments which include free fall bombs, rockets and retarded acceleration weapons. Although the same ballistics problem has to be solved in both cases, the numerical techniques of doing that are different.

We describe the A6-E system in sufficient detail so that our estimates and extrapolation are based on reality rather than on an assumed hypothetical model. The A6-E system is documented in two documents, one which describes the flowchart [18] and the other which describes the system functionally [19]. The assembly language version of the operational flight program (OFP) was also used.

1. A6-E Tactical System

The operational flight program performs the following functions:

a) Navigational Calculations

b) Tracking and Ranging Calculations

c) Ballistics Calculations

d) Sensor Input/Output and Steering

a. Navigational Calculations

The sensors used to generate information to the

navigational system are: the inertial navigational equipment, the doppler radar equipment, the magnetic compass, the airspeed indicator, and the altimeter. The intertial navigation subsystem (INS) contains accelerometers which are able to measure accelerations along three mutually perpendicular axis (x,y,z). These measured accelerations are integrated by analog circuitry into velocity components, v_x, v_y, v_z which are converted by analog to digital converters into digital information which is placed into computers memory at the fixed rate of 20 times per second.

Similarly, the Doppler radar measures analog information which is converted into digital information, namely, the velocity component along the ground track, and the angle between aircraft heading and the ground track. Finally, the magnetic compass reading is converted into digital information along with the airspeed and altimeter readings.

It is important to distinguish between accuracy and precision. Accuracy would be a measure associated with how close the reading of the instrument would be to a carefully calibrated instrument. The precision is related to the number of binary digits which are generated. The precision of the instruments is 12 binary digits or less. The accuracy is dependent on calibration and generally is 10 binary digits or less. There are systemic errors such as the Schuler pendulum effect which depends on how carefully the gyroscopes are stabilized before the flight takes place. The digital systems can by used to aid in the calibration process and the Schuler pendulum effect is partially neutralized by a digital to analog signal which is fed back to the accelerometers.

Unfortunately errors are neither random nor are

they purely due to bias, therefore it is not possible to improve the ten binary digit accuracy unless the sensor instruments are improved. Increased accuracy and precision, however, is extremely expensive.

The navigational system operates in four modes depending on the confidence that the pilot and bombardier/navigator have in the instruments. If the inertial navigational system and Doppler radar are both operational and do not give mutually conflicting data, this mode is considered most accurate. If one or the other is suspected of inaccuracy, the two more modes result, inertial alone or Doppler alone. In case of failure of both systems, the magnetic compass and airspeed with a manual correction for wind velocity is used. The navigator is the backup for all four modes failing.

The computer program for the navigational function is subdivided into nine segments, each segment is documented as a flowchart page as well as two to five pages of assembly language program. We have extracted from each flowchart page the basic operation counts in the program. Because the AO-E computer (IBM 4Pi) has no floating point arithmetic, the assembly language program uses scaled arithmetic. In the flowchart, however, it is quite clear which variables correspond to floating point numbers and which variables are fixed point and/or logical variables. Table VI.1-6. contains the breakdown of each page of the flowchart with a count of each type of instruction as well as the number of calls to library subroutines. The two rows corresponding to the flowchart page are the instruction counts in the execution path which would be the most time consuming execution path in the upper row and the total instruction count on the page in the lower row.

The column headings refer to the instruction

types categorized as follows.

C-Conditional branch for integer operands

I/S-Load or Store an integer

CF-Conditional branch for Floating point operands

ISF-Load or Store a Floating point operand

FAD-Floating point ADd

FMU-Floating point MUltiply

FDV-Floating point DiVide

CC-COsine function

SI-SIne function

AT-Arc Tangent function

LN-Logarithmic fuNction

SQ-SQuare root function

MU-the number of independent cycles in the flowgraph

ES-the number of distinct execution sequences in the flowgraph

The last two columns deal with control complexity measures which have a close relation to the number of tests required to verify program identity to a

given prcgram. These concepts are discussed in Chapter V.

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
AIR DATA1	3	9	3	30	10	7	2	0	0	0	1	0	9	72
	6	12	3	40	11	8	5	0	0	0	2	0		
AIR DATA2	1	7	0	23	10	7	2	2 0	0	0	1	2	3	
	2	10	0	33	14	10	2	2	0	0	0	1		
AIR MASS	1	3	6	29	11	4	4	1	1	2	0	1	10	157
ANGLES	4	6	6	38	11	4	4	1	1	2	0	1		
DOPPLER	1	5	0	26	16	6	0	1	1	0	0	0	5	6
VELOCITY	2	6	3	33	17	6	1	1	1	0	0	0		
SYSTEM	4	2	0	37	10	10	1	1	1	1	0	3	6	11
VELOCITY	6	6	0	62	16	14	2	2	2	1	0	3		
BARO IN	2	2	3	22	9	6	2	0	2	0	0	0	8	42
VERT LP	4	6	5	52	11	7	3	0	2	0	0	0		
INERTIAL	2	3	0	27	11	8	2	1	2	2	0	2	1	2
ANGLES	2	3	0	27	11	8	2	1	2	2	0	2		
PLATFORM	1	3	2	36	14	13	5	1	0	0	0	0	2	4
CORRECT1	1	7	2	40	14	13	5	1	0	0	0	0		
PLATFORM	1	1	0	58	27	22	6	1	1	0	0	0	1	2
CORRECT2	1	1	0	66	31	24	8	1	1	0	0	0		
TOTALS	16	35	14	288	118	83	24	8	8	5	1	7	44	10^9
	28	57	19	386	136	94	32	9	8	5	2	7		

TABLE VI.1 A6-E NAVIGATIONAL

FUNCTION COMPLEXITY MEASURES

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
COMMAND	3	11	0	4]	7	3	2	3	1	0	0	0	4	9
STEERING1	4	30	0	44	7	3	2	3	1	0	0	0		
COMMAND	3	3	1	39	24	7	2	1	2	2	0	0	9	40
STEERING2	5	13	4	68	31	20	5	1	4	2	0	0		
COMMAND	3	13	3	35	5	6	5	0	0	0	0	0	6	32
STEERING3	3	21	3	49	6	7	5	0	0	0	0	0		
SAMPLE	0	52	1	0	0	0	0	0	0	0	0	0	0	2
INPUTS	0	56	1	0	0	0	0	0	0	0	0	0		
INTERRUPT	4	20	0	4	2	0	0	0	0	0	0	0	6	9
SERVICE	9	32	0	16	7	0	0	0	0	0	0	0		
STEERING	4	8	1	44	10	20	1	1	1	0	0	0	6	24
DISPLAY	5	10	1	48	12	21	1	2	1	0	0	0		
DISCRETE	0	86	2	2	0	0	0	0	0	0	0	0	2	3
OUTPUTS	0	90	2	2	0	0	0	0	0	0	0	0		
STEERING	2	10	0	6	1	1	0	0	0	0	0	0	8	10
KEY SEL1	5	31	3	22	1	1	0	0	0	0	0	0		
STEERING	5	6	0	18	0	2	0	0	0	0	0	0	11	22
KEY SEL2	10	19	1	66	0	3	0	0	0	0	0	0		
TOTALS	24	209	8	190	49	39	10	5	4	2	0	0	53	
	41	302	15	315	64	55	13	6	6	2	0	0		819

TABLE VI.2 A6-E INPUT/OUTPUT

AND STEERING

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
ROCKET	0	0	4	48	17	19	5	1	1	0	0	0	5	10
ATTACK1	1	5	4	52	17	19	5	1	1	0	0	0		
ROCKET	1	18	2	34	17	13	5	1	0	0	0	1	2	6
ATTACK2	1	19	2	36	17	13	5	1	0	0	0	1		
BOMB	2	4	0	16	2	5	1	0	0	0	0	1	3	4
ATTACK1	3	5	0	30	5	9	1	0	0	0	0	1		
BOMB	5	5	2	46	21	15	6	2	2	0	0	0	9	32
ATTACK2	5	5	4	90	23	17	6	2	2	0	0	0		
BOMB	2	2	0	78	43	39	4	3	3	0	0	2	3	4
ATTACK3	2	2	1	93	43	40	4	3	3	0	0	2		
BOMB	2	4	4	48	23	13	2	1	1	0	0	1	9	35
ATTACK4	3	5	6	58	23	13	2	1	1	0	0	1		
BOMB	2	2	3	17	3	2	1	1	0	0	0	0	9	12
ATTACK5	3	4	6	36	3	3	2	1	0	0	0	0		
BOMB	4	10	6	8	3	2	1	0	0	0	0	0	13	105
ATTACK6	6	14	9	27	6	6	1	0	0	0	0	0		
BOMB	4	4	3	33	11	12	4	1	0	0	0	0	15	102
ATTACK7	11	11	4	76	18	14	5	2	0	0	0	0		
COMMON DRAG	2	2	0	84	41	42	3	0	1	0	0	1	2	5
	4	6	1	85	41	42	3	0	1	0	0	1		
TOTALS	24	51	24	412	181	162	32	10	8	0	0	6	70	10^{11}
	39	76	37	583	196	176	34	11	8	0	0	6		

TABLE VI.3. A6-E BALLISTICS FUNCTION

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
GREAT CIRC	0	0	0	46	10	13	4	4	5	2	0	0	0	1
NAVIGATION	0	0	0	46	10	13	4	4	5	2	0	0		
TRACK RADR	3	7	5	23	6	6	0	0	0	0	0	0	9	10
TESTS	4	10	5	27	6	6	6	0	0	0	0	0		
DEPR ANGLE	3	4	7	21	4	3	1	0	1	0	0	0	10	147
TRACK-1	3	16	7	21	4	3	1	0	1	0	0	0		
TRACK SCAN	7	13	0	44	9	14	2	4	6	2	0	0	9	35
TESTS	9	21	0	51	10	14	2	4	6	2	0	0		
DEPR ANGLE	5	9	0	11	2	2	2	0	0	0	0	0	9	20
TRACK2	9	21	0	32	10	7	4	0	0	0	0	0		
LINE OF	2	8	4	31	9	5	4	3	1	1	0	0	8	72
SIGHT RNG1	3	12	5	50	10	6	5	4	1	1	0	0		
LINE OF	10	25	2	16	2	2	0	1	1	0	0	0	13	216
SITE RNG2	10	45	3	19	2	2	0	1	1	0	0	0		
SHRIKE	4	14	4	36	11	9	3	3	3	1	0	1	13	17
RANGING	15	39	0	25	9	3	3	1	1	0	0	0		
TOTALS	41	91	22	245	60	57	19	16	18	6	0	1	86	10^{13}
	61	193	25	323	74	64	30	17	19	6	0	1		

TABLE VI.4 A6-E TRACKING AND

RANGING FUNCTION

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
TARGET INI	6	6	0	120	4	0	0	0	0	0	0	0	5	18
	7	38	0	140	6	1	0	0	1	0	0	0		
TARGET POS	5	10	0	57	24	14	8	0	0	0	0	0	7	9
FILTERS1	7	14	0	73	28	18	11	0	0	0	0			
TARGET POS	2	4	0	16	4	0	0	0	0	0	0	0	2	3
FILTERS2	2	4	0	16	4	0	0	0	0	0	0	0		
SLEW	5	24	0	8	0	2	0	0	0	0	0	0	5	12
UPDATE1	5	26	0	20	0	2	0	0	0	0	0	0		
SLEW	8	12	2	49	16	10	4	1	1	0	0	0	13	136
UPDATE2	14	24	2	87	18	10	4	1	1	0	0	0		
ANGLE	2	4	4	44	12	13	5	2	1	0	0	0	7	16
RATES	2	7	5	62	13	16	6	2	1	0	0	0		
CURSOR	2	2	1	87	49	32	12	4	4	3	0	2	7	13
UPDATES	6	8	1	125	52	37	14	5	4	4	0	4		
RADAR	1	10	3	26	0	0	0	0	0	0	0	0	4	8
OUTPUTS	1	10	3	26	0	0	0	0	0	0	0	0		
TARGET POS	1	3	0	46	15	11	3	1	1	2	0	1	1	2
UPDATES	1	3	0	46	15	11	3	1	1	2	0	1		
TOTALS	32	75	10	453	124	82	32	8	7	5	0	3	51	10^9
	45	134	11	595	138	95	38	9	8	6	0	5		

TABLE VI.5. A-6E TARGET UPDATES

	C	L/S	CF	LFS	FAD	FMU	FDV	CO	SI	AT	LN	SQ	MU	ES
RESELECT	8	9	0	0	0	0	0	0	0	0	0	0	9	45
LOGIC1	9	31	0	0	0	0	0	0	0	0	0	0		
RESELECT	5	11	0	0	0	0	0	0	0	0	0	0	14	19
LOGIC2	14	30	0	4	0	0	0	0	0	0	0	0		
RESELECT	7	8	0	0	0	0	0	0	0	0	0	0	11	16
LOGIC3	11	25	0	3	0	0	0	0	0	0	0	0		
RESELECT	10	14	0	0	0	0	0	0	0	0	0	0	20	62
LOGIC4	20	36	0	0	0	0	0	0	0	0	0	0		
RESELECT	8	10	0	0	0	0	0	0	0	0	0	0	16	17
LOGIC5	16	30	0	0	0	0	0	0	0	0	0	0		
ATTACK	7	11	3	15	2	2	0	0	0	0	1	0	11	33
SELECT1	8	25	3	25	6	3	1	0	0	0	1	0		
ATTACK	6	23	0	22	5	5	1	1	1	0	0	0	10	72
SELECT2	10	40	0	38	7	5	1	1	1	0	0	0		
STEP OUT	2	2	7	15	6	1	1	0	0	1	0	0	19	102
OF ATTACK	9	19	10	18	6	1	1	0	0	1	0	0		
ATTACK	5	7	10	16	4	2	0	0	0	0	0	0	17	286
VALID1	6	34	11	17	5	2	0	0	0	0	0	0		
ATTACK	7	9	4	24	12	2	2	0	0	0	0	0	15	84
VALID2	10	20	5	39	18	3	2	0	0	0	0	0		
TOTALS	65	94	24	92	29	12	4	1	1	1	1	0	142	10^{17}
	113	290	29	144	42	14	5	1	1	1	1	0		

TABLE VI.6. A6-E ATTACK DECISIONS

We chose floating point operations to characterize the program because the essential reason why floating point operations have not been used in tactical computing has been that hardware floating point arithmetic has been too expensive to implement. The expense has changed radically with the LSI chips with the present cost range between $200 - $5000 for the floating point unit. The AN/UYK-14 and the AN/UYK-20 both have floating point options.

In Table VI 7-12 we have tabulated the number of FORTRAN or CMS-2 instructions (if the flowchart were translated into FORTRAN or CMS-2) in the categories of arithmetic (AR), conditional (IF) and control alteration (GO) statements. In addition we have given the number of assembly language instruction in the actual program and the number of bytes (8 bits) the program occupies in memory. The number of CMS-2 statements would be the same as in FORTRAN except that each variable in CMS-2 has to be defined in an additional statement.

We include the FORTRAN and CMS-2 versions for purposes of comparison with the assembly language program. We draw some conclusions as to relative efficiencies of programming at the end of this chapter.

One surprising fact is the large number of possible execution sequences which appear in the navigational program, namely 10^9. The number of execution sequences is related to the number of different functions calculated in this program segment. As is seen in Chapter V, the functional segments are independent of each other and the total number of distinct functions calculated by the navigational program is far less than 10^9. By reorganizing the program, it is also possible to reduce the number of

execution sequences.

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
AIR DATA1	19	9	6	34	118	242
AIR DATA2	20	2	1	23	87	168
AIR MASS ANGLES	15	10	2	27	124	272
DOPPLER VELOCITY	16	5	4	25	90	176
SYSTEM VELOCITY	26	4	4	34	115	232
BARO INRT VERT LOOP	25	9	6	40	127	270
INERTIAL ANGLES	16	3	2	21	78	168
PLATFORM CORRECTIONS1	17	2	2	21	100	200
PLATFORM CORRECTS2	28	1	1	30	243	488
TOTALS	182	45	28	255	1082	2216

TABLE VI.7. A6-E NAVIGATION

HIGHER LEVEL LANGUAGE COMPLEXITY

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
COMMAND STEERING1	33	4	4	41	109	242
COMMAND STEERING2	33	9	7	49	188	430
COMMAND STEERING3	28	6	5	39	99	318
SAMPLE INPUTS	52	1	1	54	272	482
INTERRUPT SERVICE	30	8	7	45	147	376
STEERING DISPLAY	23	6	4	33	127	252
DISCRETE OUTPUTS	46	2	2	50	254	552
STEERING KEY SEL1	18	8	5	31		
STEERING KEY SEL2	38	10	10	58	255	588
TOTALS	301	54	45	400	1451	3240

TABLE VI.8. A6-E INPUT/OUTPUT

AND STEERING HIGHER LEVEL

LANGUAGE COMPLEXITY

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
ROCKET ATTACK1	26	5	3	34	113	252
ROCKET ATTACK2	16	2	3	21	100	216
BOMB ATTACK1	13	3	3	19	62	136
BOMB ATTACK2	38	9	2	49	240	480
BOMB ATTACK3	29	3	3	35	261	532
BOMB ATTACK4	22	9	4	35	164	356
BOMB ATTACK5	15	9	7	31	73	168
BOMB ATTACK6	17	14	7	38	98	224
BOMB ATTACK7	38	15	11	64	229	498
COMMON DRAG	21	5	3	29	209	434
TOTALS	235	74	46	355	1549	3296

TABLE VI.9. A6-EBALLISTICS FUNCTION

HIGHER LEVEL LANGUAGE COMPLEXITY

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
GREAT CIRC NAVIGATION	14	0	0	14	82	164
TRACK RADAR TESTS	13	9	7	29	78	174
DEPR ANGLE TRACKING-1	10	12	10	32	102	250
TRACK SCAN TESTS	21	9	5	35	157	338
DEPR ANGLE TRACKING-2	22	9	9	40	112	268
LINE OF SIGHT RANG1	26	8	4	38	159	322
LINE OF SIGHT RANG2	19	13	7	39	107	244
SHRIKE RANGING	28	13	8	49	138	300
RADAR RANGING	23	13	10	46	131	284
TOTALS	232	104	75	411	1321	2932

TABLE VI.10. A6-E TRACKING AND RANGING FUNCTION HIGHER LEVEL LANGUAGE COMPLEXITY

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
TARGET INITIALIZE	50	6	5	61	152	306
TARGET POS FILTERS1	43	7	5	55	216	490
TARGET POS FILTERS2	10	2	1	13	15	36
SLEW UPDATE1	17	5	5	27	50	102
SLEW UPDATE2	46	16	8	70	189	396
ANGLE RATES	27	7	3	37	136	284
CURSOR UPDATES	38	7	3	48	316	678
RADAR OUTPUTS	15	4	3	22	86	230
TARGET POS UPDATES	18	1	0	19	88	176
TOTALS	264	55	33	352	1248	2698

TABLE VI.11. A-6E TARGET UPDATES

HIGHER LEVEL LANGUAGE COMPLEXITY

	AR	IF	GO	TOTAL	ASSEMBLY	BYTES
RESELECT LOGIC1	14	10	5	29		
RESELECT LOGIC2	24	15	12	51		
RESELECT LOGIC4	14	20	8	42		
RESLECT LOGIC5	10	16	6	32	583	1324
ATTACK SELECT1	29	11	4	44	161	390
ATTACK SELECT2	28	10	6	44	100	236
STEP OUT OF ATTACK	11	19	5	35	81	208
ATTACK VALID1	25	17	12	54	145	342
ATTACK VALID2	26	15	5	46	148	360
TOTALS	202	144	69	415	1218	2860

TABLE VI.12. A6-E ATTACK DECISIONS

HIGHER LEVEL LANGUAGE COMPLEXITY

2. Tracking and Ranging Calculations

The purpose of this segment of the program is to establish the position of the target with respect to the airplane. At the start of a mission up to four target positions in a sequential order can be inserted into the computer from the keyboard. Once the aircraft is sufficiently close to a target so that the landmarks which appear on the radar display allow the bombardier to place his cursor on the target, then the ranging and tracking calculations are made in a local coordinate system. It is necessary to determine the velocity of a moving target, the line along which the aircraft should move so that a released bomb or rocket would hit the target. Tables VI.3, VI.4, VI.5, and VI.6 give the complexity measure for these calculations.

3. Ballistics Calculations

The ballistics calculations determine from the initial conditions at release the down range travel and the time of fall of any particular weapon. From a mathematical point of view this corresponds to solving a second order non-linear differential equation with given initial conditions: the initial position of the aircraft and the initial air velocity of the aircraft. The numerical procedure used in the ballistics algorithm uses polynomial approximations rather than an integration method for solving the differential equations. This is done to reduce the execution time of the program. The ballistic calculations are described in Table VI.7 andVI.8. Table VI.9 andVI.10 contain the attack decision making process. Arithmetically this process is not demanding. However, from control

complexity point of view this is a difficult process to test. It contains 10^{17} execution sequences.

4. Sensor I/O and Steering

This segment of the program controls the analog to digital converter. The conversion is interrupt driven at the rate of 20 times per second. The data is gathered and smoothed for processing at the update rate of 5 times per second. The update rate of 5 times per second is used for all processes other than the data gathering function. The analog and digital outputs are also generated in this program segment. The digital to analog conversion is done under the control of the computer. Correction signals to inertial navigation unit, radar antenna control, display control and steering command are generated by this segment. Tables VI.11 and VI.12 contain the complexity parameters.

We can express the entire functional requirements of the A6-E operational flight program by the Table VI.13. The headings are described as follows:

S-Short instructions, 16 bit integers

M-Medium length instructions: load, store, and compare floating point quantities

L-Long floating instructions: FAD, FMU, FDV

X-Subprograms which calculate sines, cosines, etc.

INTG-number of integer variables in the program

REAL-number of floating point variables in the program

EXT-number of variables which are
used by other programs
external to the named one.

	Instructions				Variables		External	
	S	M	L	X	Int	Real	Int	Real
Navigational	51	302	225	29	18	125	3	47
Function	85	405	262	31	18	125	3	47
Tracking &	239	730	374	64	50	192	7	7
Ranging	433	954	439	71	50	192	7	7
Ballistics	234	552	420	28	20	170	18	16
Calculations	518	793	467	29	20	170	18	16
Sensor I/O &	233	198	98	11	87	103	17	16
Steering	343	330	132	14	87	103	17	16

Table VI.13 Summary of A6-E
Program Segments

Row one of each program segment expresses the number of instructions in the most time consuming execution sequence, that is, the worst case execution time. Row two contains the total number of instructions in the four categories. We use these values to estimate worst case execution times and memory requirements for program and data. From the number of variables used externally to the subprogram, we can estimate the worst case time requirements for data transfers on the bus.

We present the A6-E in sufficient level of detail so that we can extrapolate from this data the functional requirements for other aircraft, in particular, the VSTOL. Because the A6-E is a functioning system, it does represent a real system rather than a hypothetical one. If the armaments and sensors of the VSTOL are functionally the same, then we can expect close similarity in the operational flight program.

A few points about the nature of the A6-E program might be made. There is a large number of branch statements in relation to arithmetic statements. On the average, one third of all statements are branch (IF) statements in the FORTRAN implementation. More than half of the statements are control statements (IF, GO). This indicates that the control complexity is very high and hence testing the system is difficult. The ratio of FORTRAN statements to bytes of machine language program is 1 to 8. This is fairly typical of higher level languages, although scientific programs would have more code generated from one FORTRAN instruction. Another noted fact is that the maximal time execution sequence contains approximately 70% of the instructions in the entire program. The actual assembly language program contains 25% more instructions than the assembly language programs which contain floating point operations. This

difference is explained partially by noting that scaled arithmetic operations need a large number of shift operations. Also logical operations such as masking operations were neglected in the floating point version of the assembly language program.

5. The A7-E Tactical System

The A7-E computer system is functionally very similar to the A6-E. Although the computer, the ASN91 (IP-2) is different, its capability is nearly the same both in terms of maximum memory capacity (16K) and execution speeds. The word size is 16 bits, instead of 32, although double precision operations exist.

The sensors for navigation are functionally the same, the tracking radar is functionally similar, the ordnance and weaponry overlaps to a large extent. The major difference is that one man functions both as the pilot and navigator/bombardier. The display is quite different in appearance, although from the point of view of an operational computer program the differences are minor.

The programming for the operational program was designed and carried out by different people, hence a different analysis and different numerical methods were used to solve the same problems. We did not carry out the same analysis for the A7-E as we did for the A6-E because of the lack of resources. We would not be surprised to find considerable differences in the number of instructions to carry out the ballistics function. Two studies have shown that equivalent results are obtained by programs which are much shorter [6], [14] and integrate the differential equations directly rather than using polynomial approximations. The functional complexity is nevertheless

similar in terms of execution time.

B. OVERVIEW OF THE FIGHTER AIRCRAFT F-15 AND F-18

Figure VI.B.1 depicts the F-15 tactical system's organisation. A network consisting of the IBM AP1 mission processor surrounded by 9 microprocessors, each dedicated to a particular subtask. The network is connected together by the 1553 time multiplexed data bus. The mission computer is very similar to the PI-4 computer in architecture. The add and multiply speeds are twice as fast and the divide speed is four times faster than the PI-4 computer's. There is no floating point option.

The F-18 tactical system is shown in Figure VI.B.2 . The two mission computers, AN/UYK-14's, are surrounded by 12 microprocessors, each with its own special function. The 1553 bus again is used to interface the network.

A functional description of the system was not yet available for this study. From the Figures VI.B.1-2, it is evident that some of the functions carried out by the mission computer in A6-E are carried out by the peripheral computers: inertial navigation, radar tracking, display processing, air data and stores management. The flight control functions are carried out by the special dual-redundant dedicated processors. A maintenance data recorder, a communications systems controller are new features and a laser spot racker, forward looking infra red (FLIR) sensor are added sensors absent from the attack aircraft.

F-15A AVIONICS

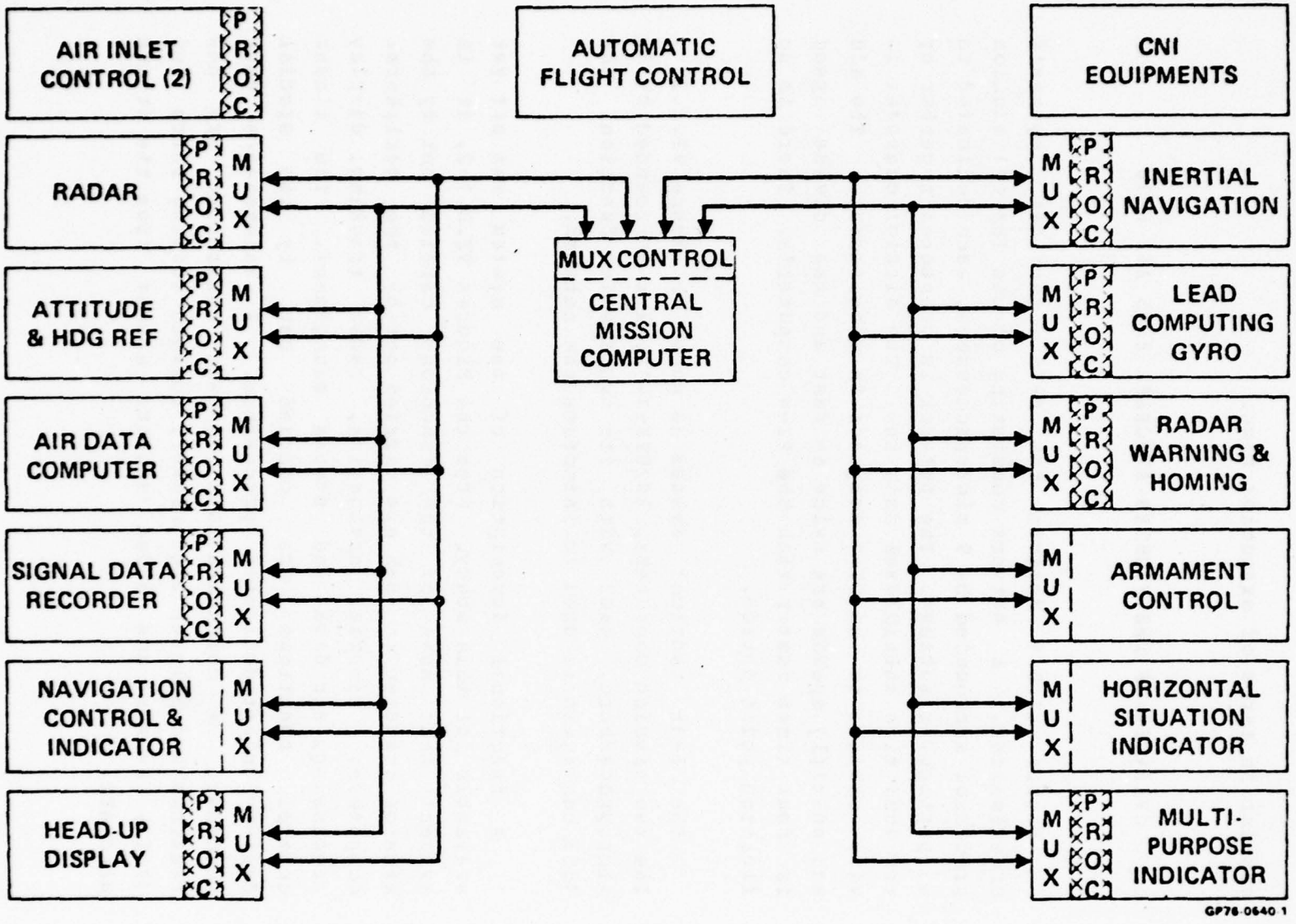


Fig. VI.B.1 The F-15 Avionics

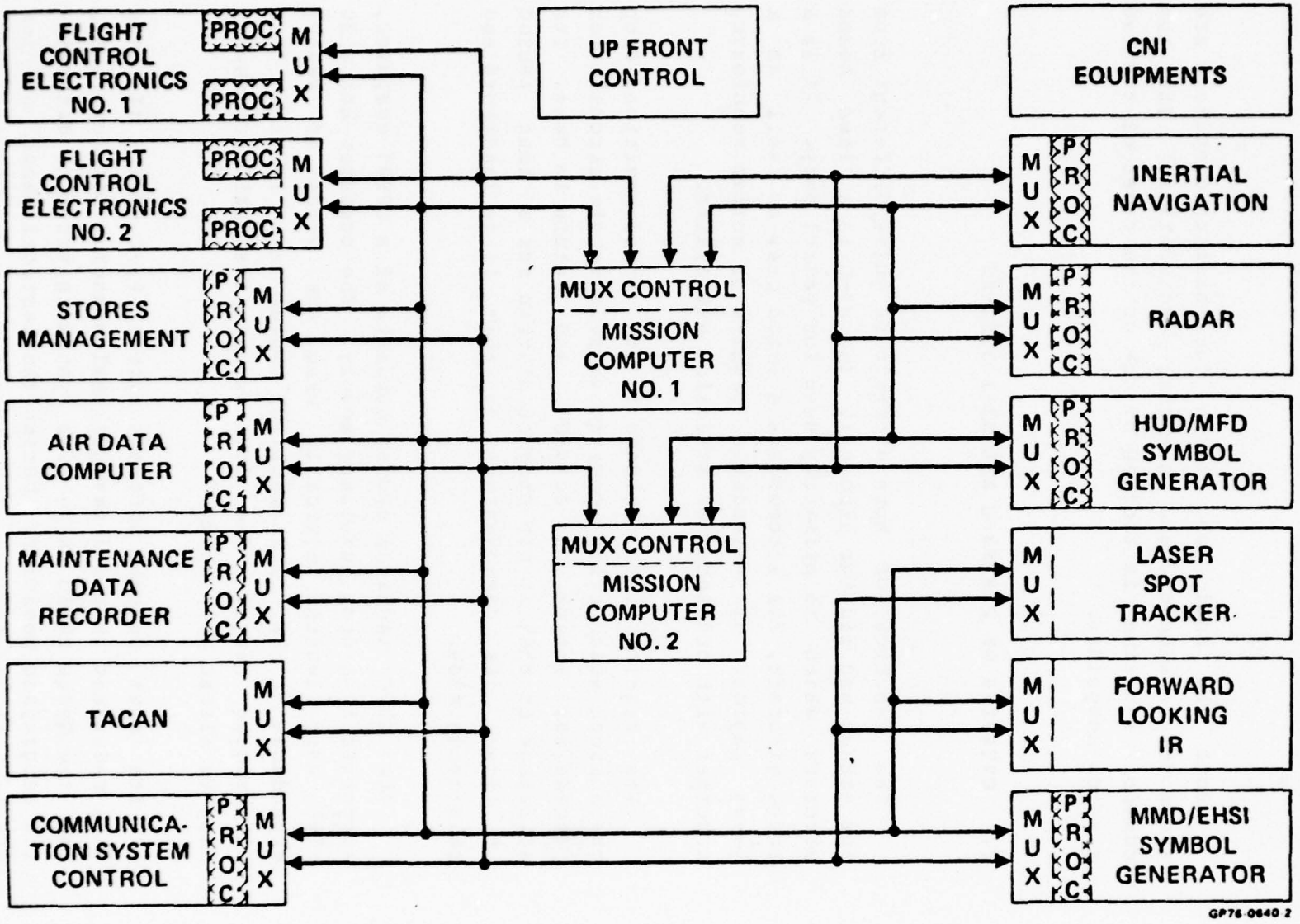


Fig. VI.B.2 The F-18 Avionics

Both the dual bus and the dual mission computers are there for redundancy and graceful degradation. The bus control function is handled by one or the other of the two mission computers.

C. OVERVIEW OF THE P3-C AND S3-A SUSTEMS

The function of both aircraft is quite different from the attack and fighter aircraft. The P3-C is a land based aircraft which is primarily used for patrol duty. It is a large aircraft, can accommodate a large crew as well as a heavy payload of expendable passive or active sonobouys, together with ordnance for attacking submarines.

Its mission is to navigate to its patrol position, drop the sensors which relay acoustic signals to the aircraft for processing, attack on command, and return to base. Its advantage is that it can stay on station for a long period of time: its disadvantage is that it is landbased and relatively slow.

The P3-C tactical system consists of a CP901 computer, supported by a drum auxiliary memory. The computer has a 30 bit word length, typically uses 48K words of memory expandable to 65K. Its execution speeds are in the 5-20 microsecond range. The drum expands its auxiliary memory size to almost 400K words.

Its most current software system, P3-C UPDATE II, uses dual-redundant inertial navigational sensors, Doppler radar, and the Omega radio navigation system in order to determine its geographic position. During the tactical phase of its mission it navigates with respect to the buoy field. The

computer can be used to release the buoys at the right instant to drop into a predesignated position. Thus the ballistics function is carried by the computer.

The signal processing details are classified and not included in the report. The stores management function is carried out by this computer as well. Except for the signal processing function, the P3-C carries out the very same functions as the A6-E. Functionally this "core" process is very similar.

Currently the P3-C system is execution time bound. Because the entire program cannot reside in memory, programs must be brought in from the drum, executed, data used for other processes, the program overlaid by another and so on. The operating system must therefore be more complex and a substantial amount of system's overhead arises because of the limited memory size and the overload on the central processor.

The S3-A is very similar in its functions to the P3-C. Because of its smallness, it can land on carriers. The smallness, however, limits its time on target, limits its crew to four and its payload. Because of its faster speed and shipboard base it can make up for some of the disadvantages.

The tactical system consists of a dual UNIVAC 1832 processor. In its maximum configuration it can have 256K 32 bit words of memory. The dual central processor configuration is used for both processing speed and graceful degradation. Floating point hardware is implemented and in its capability and architecture it is identical to the AN/UYK-7.

D. THE FUNCTIONAL REQUIREMENTS OF THE VSTOL TACTICAL SYSTEM

1. VSTOL fighter-attack version

The functional requirements for VSTOL fighter-attack version are assumed to be similar to A-6E, A7-E, F15, F18. The sensors are likely to include the presently used ones for navigation and target tracking. The major additions will be in the engine monitor-control and landing or takeoff flight control. There will be unforseen new sensor and effector development wich results in a need for designing growth capability into the system.

We shall use the A6-E program actual data as an estimate of the needs for these functions which are similar. Table VI.D.1 gives our estimate based on the data derived from the A6-E. This is compared to an estimate given in a Naval Weapons Center Report. Our estimates are somewhat lower, under the assumption that memory requirements for programs and data will not differ substantially from present values whatever computer is used for the implementation.

The comparison to Naval Weapon Center's estimate does not include their 25% safety margin, which brings their total estimate to 163,000 bytes of memory.

Our estimate amounts to a little more than twice the memory presently used in the A6-E. Of course, only time will tell whose estimate is closer to the value used. In distributed systems design it is not

Program Name	Memory Requirement Estimates	
	A6-E Program or Estimate	NWC Estimate
Executive	7600	7200
Navigation	2716	9400
Air to Surface	6620	7100
Air to Air	2800	4600
Data Link	2500	2260
Target Tracking Multiple Targets	7042	9000
Displays and Data Entry	10,000	25,900
Engine Management	10,000	16,000
Flight Control	8,000	-
Unforseen 25%	10,000	20,365
Total	71,598	101,825

Table VI.D.1 Estimates of Program and Data Length in Bytes (8 bits) for VSTOL (Attack Version)

crucial that the estimate be correct to the nearest memory module. If more processing is needed another processor with its own memory can be added. In case of a centralized computer, it is more important to leave enough space for expansion, because exceeding the available maximum memory size causes serious difficulties in systems design.

2. VSTOL ASW Version

To make an estimate of the functional requirements for the ASW version is somewhat more difficult. If the sensors will be similar to P3-C and S3-A then the major differences will exist in the display processing. We are assuming that the number of personnell carried by the aircraft will be less than or equal to the S3-A.

We shall use the P3-C as a functional guideline in our estimate of computer capacity needed for the ASW version of VSTOL. The processing rates will also be assumed to be those used for the P3-C. Table VI.d.2 contains the information for the program and data space estimated requirements.

We have not seen any program complexity estimates for ASW version of VSTOL. Our estimates rely heavily on the P3-C and S3-A present implementations. The execution rates of the functional segments vary and some are considerably lower for ASW applications than are fighter or attack aircraft. The operating systems in current implementations are more complicated because a large part of the operational program resides on drum and is brought into the computer's memory only when priorities permit. Therefore, the use of the functional segments which exist in the P3-C and S3-A systems and which are implementation dependent do not allow us to project into the future with the same accuracy as for

the attack/fighter versions.

Program Name	Memory Requirement Estimates
Executive	10,000
Navigation	7,000
Tactical Control	20,000
Communications	4,000
Tracking and Sensor Management	8,000
Displays and Data Entry	20,000
Engine Management	10,000
Flight Control	8,000
Signal Processing	80,000
Unforseen 25%	41,750
Total	208,750

Table VI.d.2 Estimates of Program and Data Length in Bytes (8 bits) for VSTOL ASW Version

VII. SYSTEM'S IMPLEMENTATION

In this chapter we assume that the applications programs have been designed, decisions on which computers to use have been made and the interconnection scheme has been determined. We shall look at alternative implementations from the point of view that the same application programs must be executed on the system with the same update frequency and with a numerical accuracy which is not degraded by the computational system. Although our study concerns itself with the future systems and therefore the computer systems of the early 1980's would be the ones used to implement the systems, we shall use presently available computers to demonstrate the feasibility of constructing homogeneous distributed systems from present-day technology. Again, LSI computers of the 1980's will simply mean fewer chips in the network compared to present estimates.

A. HOMOGENEOUS IMPLEMENTATION

1. The Systems Hardware

The system is built from identical single board processors such as the INTEL SBC80-20 or the Texas Instruments TM9900. The boards are connected by the INTEL MULTIBUS, II TILINE or the Digital Equipment's UNIBUS. A group of single board computers connected on a parallel bus is called an affinity group [4], as shown in Figure VII.A.1.

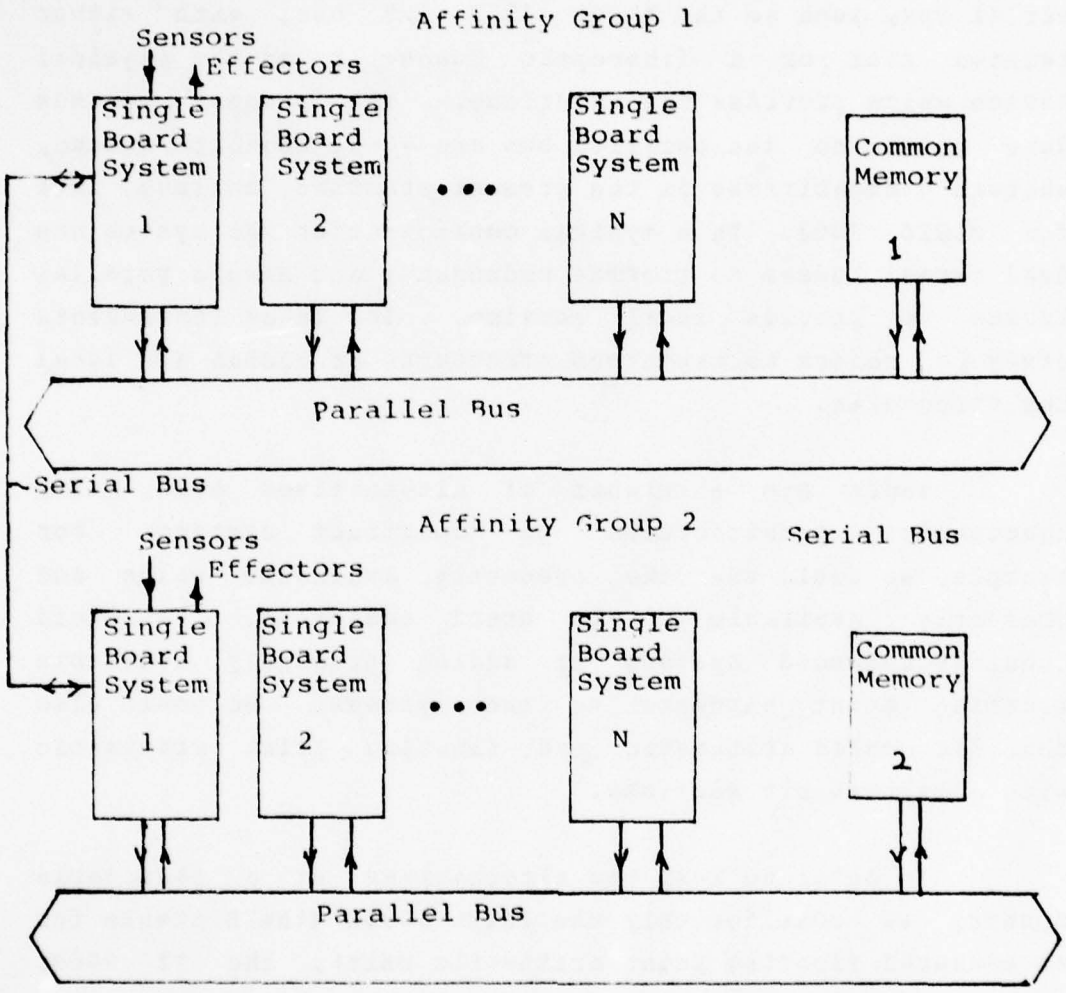


Figure VII.A.1 Two Affinity Groups of N Single Board Computers in a Homogeneous Distributed System

Physical remoteness of system's elements makes a parallel connection expensive and inconvenient. Therefore, the physically remote system's elements are connected by a serial bus, such as the MLSTD 1553 A/B bus, with either twisted pair or a fiberoptic connector as the physical device which provides the interface. The present maximum data rates on the parallel bus are 40-50 megabits/second, whereas 1 megabit/sec is the present standard maximum rate for MLSTD 1553. In a typical configuration the system has dual serial busses to provide redundancy and single parallel busses to provide local service. The Texas Instruments study [4] refers to these bus structures as global and local bus structures.

There are a number of alternatives even with homogeneous architectures to construct systems. For example, we could use the presently available chips and presently available single board computers. We could consider enhanced systems by adding presently available floating point hardware to the systems. We could also consider scaled arithmetic and floating point arithmetic with a sixteen bit mantissa.

In order to keep the alternatives at a reasonable number, we consider only the INTEL 8080E (the E stands for an enhanced floating point arithmetic unit), the TI 9900, LSI-11 each with a floating point unit.

2. Distributed Systems Design

We assume that the modular program design is complete so that each module has a single entry and exit point, called a control segment in Chapter V. To each control segment corresponds a data flowgraph which

explicitly identifies the input and output data and control variables. The input variables must be available to the process before the execution starts.

If a single computer is used to carry out the processes, as is the case with A6-E, then each process is executed in a predefined sequence. If a process is not needed, its execution is bypassed. Figure VII.A.2 shows the time-line of the processes for one execution of the A6-E operational flight program. The worst case execution sequence is less than .2 seconds for the entire program. There is a period of idle time before the next iteration is carried out.

Each process is a function which operates on a set of input values and generates a set of output values. In the language of Chapter V the function may be described as a data flowgraph. The number of output values which are used elsewhere in the program can be identified as vertices of the flowgraph which connect the data flowgraphs of the corresponding processes. Figure VII.A.3 illustrates the idea using hypothetical data flowgraphs.

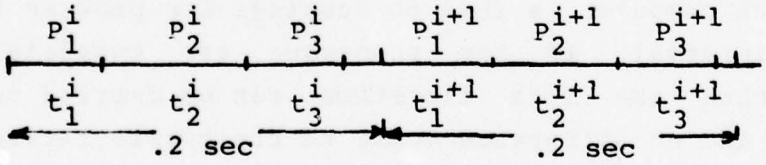


Figure VII.A.2 Time Line for the Single Computer A6-E Execution Sequence

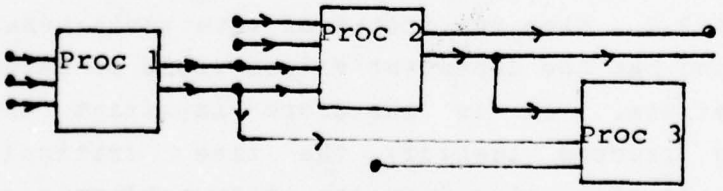


Figure VII.A.3 Data Flowgraphs for Three Hypothetical Processes

The same computational process can be carried out by a collection of slower processors. As an example, let us pretend that the processors we wish to use are approximately three times slower than the single processor which successfully is able to carry out the three processes. If computers 1,2, and 3 are assigned to processes 1,2, and 3, then their execution times become three times longer, as shown in Figure

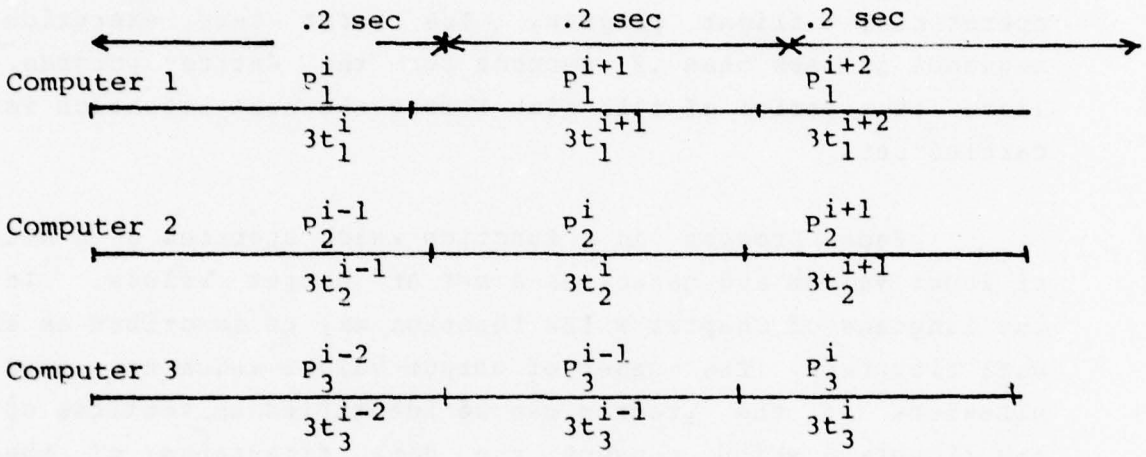


Figure VII.A.4 Time Lines for a Three Computer Implementation of the Distributed System

VIIA.4 Each computer is able to complete its process in the .2 second interval. If the processes are completely independent, then the i-th iteration can be carried out simultaneously and no difference would be observable between the single computer solution or the three computer solution. If the processes use each other's results as indicated in Figure VIIA.2.2., then the iteration rate would remain less than .2 second but the dependent values would be delayed by three iterations. It is therefore important that the partitioning process identify the time critical data flowgraphs and place them into the either the same computer or the same iteration interval.

Another aspect which becomes important is the transfer of data between computers. The parallel data bus allows several forms of data transfer at high transfer rates. The use of common memory for these data values which need to be transferred between computers is particularly efficient whenever relatively few such values are to be placed on the system bus. Computers would interfere with each other only when two computers simultaneously wish to access the common memory.

Other methods of message transfer, which require simultaneous attention from the transmitting and receiving computers, would give rise to more message transfer overhead. However, the maximum data transfer rates of 40-50 megabits per second on the system bus make the data transfer overhead almost negligible as seen from the calculations which follow.

To establish that such a system of single board computers can successfully solve the tactical problem, we first look at the A6-E program, which is analysed in complete detail, and the summary information is given in Table V.1. The worst case execution time estimates to carry out each of the processes at the rate of five repetitions per second is tabulated.

If we chose to carry out the A6-E process using three computers, one choice is to assign the navigational function to one computer, ballistic calculations, input/output, and steering calculations to another, tracking and ranging, and sensor calculations to a third. This assignment would tend to minimize the bus traffic because the number of data items to be transferred from the navigational computer is 47 floating point quantities and 3 logical variables. The ballistics computer would need to communicate 16 floating point and 18 logical variables. The

tracking, ranging, sensor, input/output, and steering computer will need to communicate 24 floating point and 23 logical variables. Since these variables are communicated 5 times per second, the total number of bits per second is

$$(44 + 87 * 4) * 8 * 5 = 15,680$$

If the communication takes place at the maximum rate of 40,000,000 bits per second, the total data communications time is .39 milliseconds every second. Added to this would be the time required to service and set up two interrupts from each of three computers five times per second, which as a maximum of 30 interrupt services per second. Interrupt service times are at worst about 50 microseconds per interrupt. This would add 1.5 milliseconds to the data transfer times. If the common memory concept is used for data transfers then each memory access requires an additional system bus access cycle of 200-300 nanoseconds, which would add a total of

$$(44 + 87 * 4) * 5 * 2 / 5 = 784$$

to the execution time every second. Therefore, a three computer system would in the worst case use .15% of the bus capacity. The total estimated execution times in seconds for each computer including communication time is listed in Table VII. A.5.

	8080 E	LSI-11	TI 9900
Computer 1 Navigation	.2786	.2685	.267
Computer 2 Ballistics & I/o	.3916	.370	.368
Computer 3 Track & Range	.683	.660	.658

Table VII.A.1 Total Estimated Worst Case Execution Times (Per Second) for the A6-E System

	Application Program	Functional Subprograms	Data Unique	Data Dupl	Operating System	Total
Computer 1 Navigation	2.2 K	1 K	.6 K	.2 K	2 K	6 K
Computer 2 Ballistics I/O	6.6 K	1 K	.9 K	.1 K	2 K	10.6K
Computer 3 Track & Range	9.5 K	1 K	1.2 K	.1 K	2 K	12.8K
Totals	17.3 K	3 K	2.7 K	.4 K	6 K	29.4K

Table VII.2 Amount of Memory Required (Bytes) For the Distributed System

Computer 3 is used most heavily at about 66-68% capacity. The difference between the perfcrmance estimates of the three LSI computers is almost negligible. This is due to the fact that the most time consuming operations are the floating point operations. The floating point unit performance is nearly the same for all three computers. Distributed systems' implementation using private memory requires that certain programs exist in each computer namely, the programs required for data transfers, the functional subprograms (SI, CO, CN, AT, SQ), and some input-output handlers. If the systems use private memory, then the data items shared by the computers will be copied from one memory to another and in the worst case three copies of the shared data will occur.

The estimated number of bytes of memory required for the programs is assumed to be nearly the same in all three LSI computers. We estimate the program length in the PI-4 computer to be nearly the same as in the LSI computers because of the similarity in architectures. The estimated memory requiremens for the distributed system is given in Table VII-3. We have tabulated in Table VII.3 a

three computer architecture in which memory is only shared for common data values. Consequently, functional subroutines, some data and the operating systems are duplicated in each computer. This architecture has the drawback of requiring more memory than the corresponding single computer system. The advantages are that single board computers which are slower but much less costly, smaller in size, weight, and power requirements, can be used to implement systems which require more computational capability than provided by one single board computer.

Projections of the future indicate that the capability provided by the prsent-day single board LSI computers will be available on the single chip by 1985. The present computational requirements for the A6-E will therefore be satisfied by a single board on which three single chip computers provide the computational capability and the remaining chips are used to interface the system to sensors and displays. The cost, weight and power requiremens of such a system will be a small fractions of their present values. We are not naive to believe that the computational requiremens will remain fixed. We expect that increased capability of the chips will encourage an increase in computational requirements. We believe that the most cost-effective solution to these problems is a distributed system of these chips which are most widely used and hence sell at a cost closest to the recurring production costs which are measured in dollars per chip.

3. The Distributed Homogeneous System's Implementation of VSTOL

We wish to estimate the number of single board computers needed to implement the VSTOL systems using presently available LSI computers. Because the VSTOL aircraft will be comparable in performance to the S3-A and P3-C, the present iteration rates for the processes are assumed to be sufficient. Because the presently used computers are in the best case ten times faster than the currently available LSI single board computers, certainly no more than ten computers are necessary to give the same iteration rate as the present systems. Therefore, the basic issue in trying to estimate how many single board computers are needed becomes the question of memory size.

At this time the single board computer which contains the most memory has 8K bytes of program memory and 2K bytes of RAM. Six months from now both of these numbers will almost certainly double because 16K bit chips are already on the market. Therefore, with present technology 10 single board computers connected by a parallel bus into a system which contains a common memory for the variables needed to be shared is an upper bound estimate. This would allow us to build a system with a total of 116K bytes of memory which is well above the estimated 102K bytes stated as an estimated requirement by Naval Weapons Center [20] for VSTOL attack version.

It is superfluous to state that within a year the system's memory size could be 200K bytes and as our estimates indicate, this system in 1985 will be composed of 10 chips on a board instead of 10 boards in a card cage.

4. Software Issues of Distributed Systems

In the previous segment we have outlined a method to

use existing single board LSI computers to construct a system which solves a tactical problem of the complexity which occurs on the A6-E. In order to bring such systems into existence, a program must be generated. In most airborne applications, the program development, testing and maintenance costs have been paid for each new airplane. Typically, the major airframe contractor assumes the responsibility for the computer program. Closely related functional aircraft,(A-6E A-7E),(P3-C, S3A), have different computers connected to similar sensors, displays and weapons. In the past, programs were written in assembly language to make program size small and execution time as short as possible. The programming effort was repeated for each new project. The human-intensive programming effort has been expensive in the past and is continuing at the same level. The decreasing hardware costs encourage greater use of computers, hence more programming of nearly the same cost per instruction. Therefore, not surprisingly, the software cost to hardware cost ratio is approaching 80/20 and likely to continue its growth.

Both users of computers and computer manufacturers recognized early that it is to both of their advantage to make the lifetime of programs as long as possible. The development of higher level languages (FORTRAN, COBOL, ALGOL etc.) not only increased the productivity of programmers, but also allowed the programs to be transferred from one computer to another, from one generation to another. When disk systems replaced tape oriented systems, many of the programs did not survive intact. They had to be substantially rewritten in order to make efficient use of the system.

In airborne applications, the computer systems have always been operating near their capacities both in speed and memory size. The peripherals: the sensors, displays

and effectors have not been interfaced to the computers with standard interfaces. Therefore, much of the program will have to change, whenever changes are made in the peripherals. Furthermore, the real time tactical systems are characterized by program control complexity which is much greater than programs of similar size in other applications. Therefore, the testing of programs becomes a lengthy and complex task which is subject to change whenever the peripherals change. The combined effect of all the causes is that not much effective transfer of programming effort has taken place from one project to another. Only in systems updates, such as P3-C Update II, has there been substantial saving of reprogramming effort.

The present Navy policy is to try to enforce the use of both the standard higher level languages: CMS-2, SPL-1 as well as standard computers. It is hoped that enforcing these standards will make the lifetime of tactical programs longer than one generation, as well as allow transfers of programs between projects. The use of standard airborne computers, namely the AN/UYK-20 compatible AN/UYK-14 is helpful in both hardware maintenance and ability to use program development tools, such as compilers, assemblers, etc. which have been already developed for the AN/UYK-20.

This policy on the surface appears to enable the saving of the large investment in the stockpile of computer programs which the Navy has generated for its tactical systems. Examination shows that this policy alone is not sufficient to allow tactical programs to be carried from on generation to the other. Whenever the systems architecture changes, or changes in the sensors or displays occur, only a minor portion of the programming effort would carry over to the new system. For example, the introduction of the phased array radar into a system, would not only change the sensory interface but the entire tactical system because radically

new information with different accuracies and different information transfer rates becomes available. Our choice is to either deny the benefits of this new sensor and pretend that it is a traditional search radar in order to preserve our software, or rewrite the majority of the program. When disks replaced tapes, there were many attempts to preserve the tape program by thinking of disks as tapes. These temporary fixes have long since been phazed out and the programs have not survived the change in generations.

The Navy fleet in mothballs in the various harbors is another example of an attempt to preserve something which was extremely expensive to build and buy and therefore to throw it away seems such a waste. The changes which have occurred in shipbuilding are happening at a much faster rate in the electronics industry. Therefore, attempts to preserve expensive applications software from one generation of tactical systems to another is not realistic with our present state of knowledge and ability to predict future changes.

Successful preservation of software from one generation of computers to another has been accomplished in these applications which are processor configuration independent. By expressing the program in a higher level language such as FORTRAN, it becomes useable on any computer which has a FORTRAN compiler. The FORTRAN compiler itself can be made more easily transferable from one generation to another if the so-called "intermediate code" emitted by the compiler is general enough so that the only change necessary to adapt the compiler to a new computer is the rewriting of the final phase of the compiler, namely the "code emitter". The scientific subroutine packages, compilers, assemblers, editors, and other programs which only depend on general purpose input-output devices are examples of programs which have been successfully preserved from generation to

generation.

The present Navy policy to force the contractors to use the "Navy standard airborne" computer AN/UYK-14 has the following consequences

1) It creates a narrow branch of "Navy standard airborne" computers and thus prevents the Navy's participation in the "LSI revolution" of radical cost reductions in hardware and software by the use of defacto "industry standards," the DEC'S LSI-11, INTEL's 8080E, TI's TMS9900.

2) It tends to create heterogeneous systems consisting of the "Navy standard AN/UYK-14's" surrounded by a variety of different "industry standard" microprocessors. The F-18 and F-15 designs are good examples.

3) It will not solve the problem of preserving the applications software from one generation to the next for the reasons mentioned in the previous pages.

4) It will allow the CMS-2M compiler to generate code for the AM/UYK-14 and thus the expense of rewriting the compiler would be saved.

At this time we cannot cite successful single board distributed systems designs. Much of this design work is presently going on and reports are yet to be written. From our own experience, the design and static testing of applications modules is the most time consuming effort. Very accurate predictions of maximum execution times can be obtained by writing and executing the programs on existing developmental systems. Most of program development can be accomplished conveniently on systems which are specifically designed for program development. Existing timesharing

systems which have editors, compilers, simulators and debuggers can be very successfully used to develop, test and debug the application programs.

The decisions of which particular system is the target implementation hardware can be delayed until virtually the last minute. Decisions of how to "package" the modules into single board computers can be made at any time when sufficient performance data for both the computer and the interconnecting bus is available. Analysis software which determines execution times and data transfer requirements directly from the programs is a useful tool which would make the task of assigning modules to computers easier.

The dynamic testing of the modules which belong to a single board computer can then be done with using in circuit emulation systems. (INTEL's in circuit emulator, ICE, is one example of such systems provided by the manufacturer to permit the monitoring and final debugging of a system which directly interfaces with peripheral circuits.) Each computer can be tested independently to check whether or not the predicted performance corresponds to the "in circuit" performance.

Software tools to monitor the interaction of several computers on a data bus are presently under development by the distributed LSI computer manufacturers. These software tools at the final integration tests are useful to resolve interaction difficulties.

In summary, the software development, testing, debugging and maintenance is basically no different for distributed systems than it is for single computer systems. Only in the final integration phase, the monitoring and debugging offers some new aspects. A way of monitoring and

recording the information on the bus would be to dedicate one single board computer to be the monitor-recorder. Our recommendation is to use high level languages which are not only standard in the Navy but extend to at least the Defense Department, preferably beyond. Use of defacto standards, namely languages which have become standard because of high degree of use are preferable to decreed standards. Our recommendation is to phase out CMS-2 because it no longer offers significant advantages over defacto standards such as FORTRAN. With the use of floating point arithmetic, the scaled arithmetic which CMS-2 supports is no longer needed. The language BASIC appears to be becoming the defacto standard in the LSI computer applications. The LSI environment encourages simple, small languages which are easy to learn and require a small memory for compiler/interpreter execution. Because distributing the computing removes critical execution time problems and the inexpensiveness of memory removes the necessity for being parsimonious with memory, higher level languages are certainly appropriate for airborne computing. The only serious drawback of BASIC is that it does not support "structured programming form." The language PASCAL, PLM, and some other "structured languages" may eventually win the popularity race for LSI computing.

E. HETEROGENEOUS IMPLEMENTATION

The heterogeneous implementation is patterned after the F-18 system's concept. A dual "mission" processor is used to increase system reliability, supported by a myriad of smaller processors each possibly of different arithmetic capability. Figure VI.B.2 describes the implementation using that alternative.

The main difference between the homogeneous and the heterogeneous system is that the homogeneous one consists of identical LSI computers which communicate on a parallel bus, if the computers are physically close and on a serial bus, if the units are physically remote. The heterogeneous system may contain two or more different types of computers connected on a serial bus. The same type of computers may be connected on a parallel bus, as is the case of the dual-processor F-18 system where some memory may be shared by the two AN/UYK-14's.

There is one major reason for heterogeneous systems, namely the feeling that the capability of "microcomputers" is not sufficient to carry out the calculations required of an airborne tactical system. The dual system in the F-18 design is justifiable from the point of view of reliability. If one system malfunctions, the other continues to operate the system in a degraded mode. The so-called "mission computers" are in the "minicomputer" class and may (in case of AN/UYK-14) or may not (in case of F-15) be constructed from LSI chips.

There is a strong pull towards this form of systems architecture. The subcontractors who provide the sensors have experience and knowledge of one microcomputer system which is used to make their sensor "smart." Different subcontractors would naturally be attracted to different microcomputers. Use of a "standard" microcomputer would cause redesign and a higher price. The contractor at this point has no experience with homogeneous distributed systems. There is considerable risk from his point of view especially if the time pressure of the contract requires immediate action. Therefore, the contractors choice is a minicomputer, much like the ones he has used in previous contracts. If the program has to be documented in CMS-2, the "mission" computer must already have a CMS-2 compiler.

Hence, even if a contractor would be able to generate a homogeneous distributed system of LSI computers as the least cost solution, someone would have to provide a CMS-2 compiler, which by itself would cost an estimated $4.9 million to develop [3]. He has no choice other than the heterogeneous system.

Minimization of aquisition costs tends to create a heterogeneous system. As noted earlier, the subcontractors who are not using the Navy "standard" microcomputer would have to redesign their systems. This would result in higher aquisition costs.

The major penalty of heterogeneous systems for the Navy comes after the system has been aquired. The documentation of several different microcomputers, together with different assembly languages, or special microcomputer higher level languages will cause educational problems for the personnell. The cost will be proportional to the number of distinct microcomputer systems present.

Spare parts which allow on line system's servicing will grow in number in proportion to the number of distinct types of computers.

We will note that if the system is errorfree and if the mean time between failures reaches 100,000 hours, as discussed in the next segment, then the maintenance penalties for the heterogeneous systems are in total value small compared to the total life cycle cost of the system. Only experience will demonstrate this.

C. COMMUNICATIONS PROTOCOLS

The communications protocols are dependent on the bus structure. Typically the protocol is stratified to several levels. At level zero is the hardware protocol which is usually different for different manufacturers. There are some standards which are of importance (IEEE 448,CAMAC) . At level 1 is software support provided by the manufacturer so that the user does not need to concern himself with developing software. Typically the manufacturer also provides subroutines at level 2 which enable the user to carry out data set transfers, process synchronization etc. At level 3 are usually higher level language subroutines written in FORTRAN, BASIC, PASCAL, etc., which permit the applications programmer to make higher level language statements which cause data transfers to target subsystems. The applications programmer does not need to know any more detail than how to use the subroutine. There are attempts to standardize the protocols even at this level. At level four are the global operating systems (if any). For heterogeneous systems such develcpments would depend on the system's configuration, which computers are used and how they are configured. For a homogeneous system, such operating systems are provided by the systems's manufacturer. (INTEL's RMX 80 Real Time Executive system.)

The system's manufacturer provides the baseline or a kernel or such an executive. The user builds a customized level of software on top of the kernel. In present tactical systems, the development costs of executive systems for tactical applications has been entirely paid be the Navy. Widely used LSI homogeneous systems executives aquisition costs will be measured in thousands of dollars rather than millions of dollars used to develop operating systems.

VIII. COMPARISON OF RELIABILITY OF THE DESIGNS

A. MOST LIKELY ERRORS AND FAULTS

The experience with LSI chips dates back only a few years and therefore statistical data about reliability as a function of time is not yet complete. Accelerated life testing data is available only on some systems. The commercial version of the INTEL single board computer SBC 80/10 has undergone reliability analysis. The accelerated life test reported in [11] gives the mean time between failures (MTBF) as 91,739 hours at 90% confidence. If the equipment is operated 24 hours per day at 25°C then the expected life of the system is 10 years. This corresponds closely to field data, which indicates an MTBF of 90,845 hours. Operating the system at higher temperatures, 55°C, reduces the MTBF to an extrapolated 25,000 hours. The single board computers which are made up of military standard components have not been studied. Higher reliability would be expected for these computers.

In airborne systems, the sensors, radio communications equipment, displays and other peripherals, and power supplies are currently the least reliable systems components. A reliability improvement in the computer part of system by an order of magnitude will exaggerate this problem. Therefore, reliability improvements to the total system are not likely to arise noticeably by increasing the reliability of the computer portion, although distributed computer architecture will give the systems designer

cpportunities to do this. The observed mean time between failure of all sensory instruments and displays for the F-4E was 48.7 hours (UEDPS, 66-1 Field Data) Page 64, [9]. Therefore, a system using LSI technolcgy will fail so infrequently in comparison to the peripheral instruments that the well known Maytag television commercial, in which the serviceman complains because he has nc trouble calls, will indeed be the case.

E. TESTABILITY

Testing for malfunctions in a single ccmputer system is done initially before takeoff. Thereafter, if the computer fails in flight, the operator may again invoke the test rcutines. Typically the computer is tcc busy tc do selfchecking.

In a distributed system the time pressure is not as great and selfchecking can be incorporated into the pericdic execution sequence. In a system which is hcmogeneous, cnly cne test program needs to be written. In a heterogeneous system each distinct processor needs its cwn test prcgram. In a distributed system external checking may be accomplished. In crder for a computer to dc selfchecking, it needs to be functioning to some degree. An external computer can be effective in testing after selftesting is no lcnger feasible. Again homogeneous systems have an advantage over netercgenoeous systems. Distributed systems have an advantage over single computer systems.

C. DIAGNCSIS OF ERRCFS

A single computer system is usually under such timepressure that it can afford to do very little in diagnosing errors. Only minimal error diagnostics are typically implemented in such systems. In a distributed system, the timepressure is less critical, hence more sophisticated error diagnosis can be done. We generally wish to protect ourselves against the most likely sources of errors, namely the sensors. If independent measurements using different instruments agree with each other, we usually can trust the results. If there is disagreement, then we look for inconsistencies, large changes in small time intervals, etc. Small biases in instruments are particularly difficult to diagnose. However, with digital systems, calibration problems are easier to solve because it is possible to record information over time periods and make small adjustments. For example, if a digital watch is off one second a day consistently, then if we have access to the count which determines the displayed unit of time we can alter that and correct the bias. Distributed systems have a advantage and homogeneous distributed systems have a added advantage because of uniformity.

D. ERROR TOLERANT FUNCTIONING DURING MISSIONS

The present systems implementations have already a large degree of error tolerance. In the A6-E and A7-E the navigational instruments can gradually fail. There are typically four modes of navigation: inertial-Doppler, inertial alone, Doppler alone, air data alone. However, if the computer malfunctions, then only manual backups remain. In a distributed system there are many options for the designer. For example, there may be a spare computer which can be activated and which either has in its memory programs for the vital functions or can read the program into its

memory from an auxiliary memory.

Another solution would be to have the vital functions in two computers so that if one fails, the other can carry on, much like a pilot and copilot function on transport aircraft.

E. GRACEFUL DEGRADATION

Graceful Degradation refers precisely to the concept that one or more failures in the system degrade the performance but do not totally cause the system to collapse. A distributed system, particularly if it uses optical interfaces between components, has the ability to degrade gracefully. If systems elements are connected electronically they are not as well isolated and a serious failure in a system connected to the bus can cause a failure on the bus, which makes the entire system inoperable. An optical bus is not nearly as vulnerable because of the light signal isolates the systems electronically.

A systems designer has many options to create a system whose total failure is very unlikely. Duplicating or triplicating systems elements would be an obvious way.

IX. ECONOMIC ANALYSIS

A. SUMMARY OF ECONOMIC ANALYSIS METHOD

When estimating the costs of alternative engineering implementations using a still emerging technology, like large scale integrated circuitry, performing the economic analysis begins by making projections into the future about a set of significant variables. In the case of LSI, some of these would be propagation delay, gate/chip, cost/gate, failure rate, cost/connection, etc. There in fact exist several of these significant variables that could be identified, each one having a point, and an interval estimate of their value forecast at certain steps along the future time line being considered. An exhaustive analysis would seek to take all possible combinations of all possible values of the variables to measure the costs and uncertainties in those costs of the alternative engineering implementations. This type of analysis can quickly lose any intuition it might have built for a decision maker in a mass of data points.

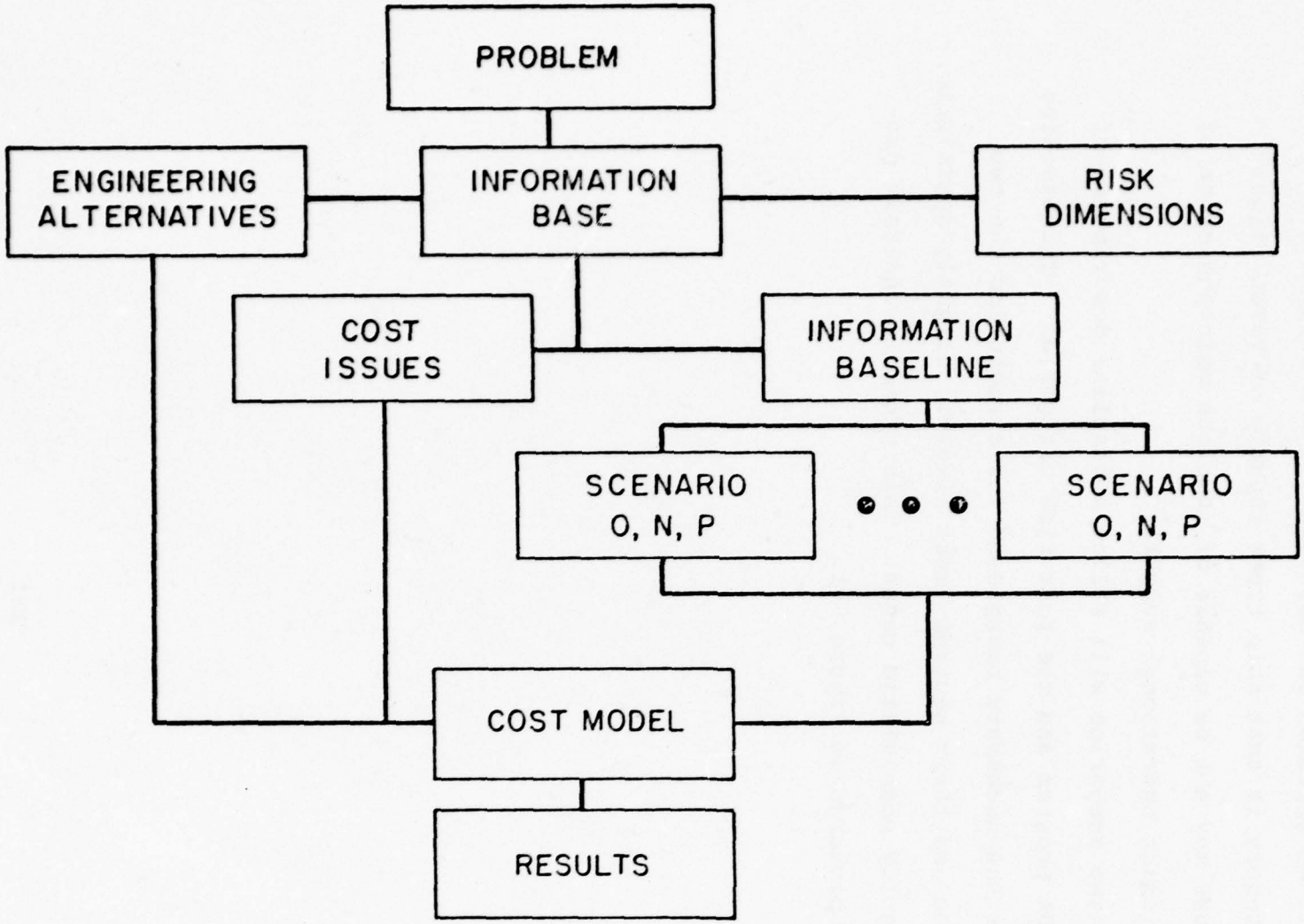
An alternative approach, and the one selected for this report, is to first create a set of broad risk categories that relate to the problem being analyzed, and to generate from them scenarios each with a neutral (or most likely), a slightly optimistic, and a slightly pessimistic case. The outcome of these three cases is a set of three values for certain key variables. The cost of the alternative engineering implementations is then based on this outcome. What

this narrative scenario approach accomplishes for a decision maker is a projection into the future structured around some managerially meaningful categories. The constants across the scenarios are the competing engineering implementation philosophies and the risk category structure. In structuring the scenarios it is important to free the implementation alternatives from the risk categories to be able to see how they then stand against each other across a spectrum of risk. A decision maker can then see how each competing alternative would fare cost-wise in possible futures that have more meaning than might be drawn out of a mass of data points.

For the economic analysis of this report alternative avionics computer architectures, each embodying LSI technology, will be placed in the context of two possible scenarios developed on the milestone path of the VSTOL aircraft. Each scenario structures the risk inherent in the future into four broad categories. The first category will be called the semiconductor industry as related to technological changes in and the marketing of microcomputing devices. The second category is the systems acquisition strategy for future avionics in light of OMB circular A109 and with regard to source selection and contract incentive structure. The third category is the maintenance-manpower system which entails repair-discard, level of maintenance, and labor mix possibilities. The last category, aircraft employment, addresses not only the number of VSTOL aircraft projected as a base for operating and support costing and the rate of aircraft production, but

also their method of employment, i.e., from which ship types they will be operated and maintained. The implication of the last category is that ship types capable of VSTOL flight operations may not be capable of complete maintenance or of multi-mission operational support.

The two scenarios will follow a baseline description of the VSTOL program and the four risk categories. The baseline develops the necessary background for structuring the two scenarios and their neutral (most likely), slightly optimistic, and slightly pessimistic cases. This economic analysis process is pictured in Figure 9-1.



ECONOMIC ANALYSIS METHOD

Figure 9 - 1

B. BASELINE

As stated, the four categories of risk briefly are, the semiconductor industry, the systems acquisition strategy, the maintenance-manpower system, and the employment possibilities.

1. The Semiconductor Industry as Related to Technological Advance and Marketing of Microcomputing Devices

The first chapter of this report generalized the impact of large scale circuit integration on computing technology. A more detailed account with particular reference to technological change and marketing of microcomputing devices will be discussed here.

The primary effect on the market of LSI has been to open up new applications areas for microcomputing devices. In particular, is the opening up of three areas that can be broadly classed as the process control market, the consumer market, and the small business or hobbyist general purpose personal computer market. The thread common to all of these markets is the implementation of what was previously electromechanical analog logic and/or custom designed electronic circuitry by general purpose digital computing logic integrated to one or a few semiconductor chips.

Before briefly reviewing the history of these microcomputing devices, reflection on the life cycle of computer products in general reveals a reasonably standard sequence of events. Technological advance in circuitry design occurs

first, followed by exploitation in the form of a product made possible by improvements in manufacturing technology. The product, if accepted in the market, begins to find further applications. It is gradually enhanced in performance to meet these new found applications, creating a family of computers. Peripheral equipment and software development tools are developed by the parent company. If the product really makes an impact in the market, like the DEC PDP8 did in terms of helping to create the minicomputer market, other companies specializing in peripherals, maintenance, or software support and applications spring up. The basic product is upgraded until it is no longer economic, and then emulated as transition to a replacement computer product occurs.

With this in mind, concentrating on the traditional Von Neumann building blocks of the digital computer, the processor, the memory, the input/output, and the timing circuitry provides a framework for describing the technological change-marketing paths that have appeared to date and that are projected to appear in the two scenarios of this report for LSI computing devices.

A semiconductor company, Intel Corporation, already a producer of LSI memories was the strongest initiator in the microprocesser/microcomputer device market. Their Intel 4004, a four-bit word length device, was the first commercially successful microprocessor. It was and is used in limited scope process control and some consumer products like pong games. As a designer's tool it required a read only memory

chip to provide space for the controlling program. Timing circuits, and input/output interface devices were also required. As more circuitry could be put on a chip because of semiconductor manufacturing technology improvements, the question presumably became for Intel, "For commercial success, should all the building blocks of a digital computer be placed on one chip or should the single chip processor be made more capable in terms of word length, processing speed and instruction set?" Empirically the evidence shows that the processor was made more powerful first, with the introduction of the Intel 8008, an 8-bit word length processor, and then with the most commercially successful microprocessor to date, the Intel 8080. Several other microprocessors entered the market, of course, the AMD 2900, the Motorola 6800, the Zilog Z-80, the LSI11, and now the 16-bit Texas Instruments TI9900, to name a few. The significance of the last three examples named is that they are enhancements over the 8080 and that they came prior to the Intel 8048 family, the first complete computer-on-a-chip on the market and also prior to the TI9940, also a computer-on-a-chip. Also of significance is the fact that Intel announced the 8085, a fifty percent speed enhancement of the 8080 at the same time as the 8048 family.

Several reasons can be postulated as being behind this technological change-marketing path. The first is that semiconductor read only memory chips to hold the applications program were already available from separate vendors. Another is that when a process control is designed in the

framework of digital computing logic it is done so with the traditional Von Neumann computer building blocks in mind. The process is first described by the design engineer in terms of data flow, i.e., input signals being transformed by functional operators into output signals. The job is then sized in terms of execution times and memory space as separate constraints. The result is separate implementation. As applications are worked out using digital computing logic, memory size can be incremented by adding memory chips in much the same way that memory is added to a general purpose main frame or minicomputer as applications are developed. Now that the use of single and few chip microprocessors has made its initial impact on the applications market, the computer-on-a-chip with its fixed resident read only memory fits the applications that are becoming more defined in the literature as to execution time, memory size, and instruction set manipulation.

Restated another way, the microprocessor impacted the market, the market in turn then impacted the microprocessor development path. As a company makes technological headway as did Intel with the first microprocessors, that technology is exploited for the payback to the investment as outlined by the $S \cdot N \geq CNR + CR \cdot N$ formula of the early chapters in this report. After the first device hits the market the strategy generally is to follow with performance enhancement as the users become more familiar with the implementation. As other companies come into the market with even faster performing

devices as did Zilog, DEC, Texas Instruments, et al., then the lead company, in this case Intel, generally begins to discriminate its product line by the support system, and then by some technological leap-frog that is intended to put distance between itself and the others. The last few paragraphs describe some of the several reasons, besides those of semiconductor manufacturing technology, that most probably were behind the chronology of introduction of the Intel 8048 single chip computer, i.e., after the enhancements to the original successful microprocessors.

One other aspect of the market that bears mentioning is the development in the licensing of a product's technology (a concept pointed out in the Computer Family Architecture report, referred to in the next section, as most probably holding the key to much of the cost differential in most actual negotiated product selections). In the earlier days of computing, and to this day, IBM held near and dear to its technological secrets as it gained its dominant position in the market. Now companies will license the use of trade secrets to competitors. The most relevant examples are the license agreement Norden, a division of United Technologies, has with Digital Equipment Corporation to produce the military versions of the PDP/LSI11's, the ROLM license with Data General Nova Division, and recently the AMD license agreement with Intel to use Intel design masks to produce 8080A's. Two other

important ones stand out. The first is to obtain a quick return for the parent company on a technological implementation in a high rate of technological change area in the face of the 10^6 dollar development cost for the device, referred to early in this report. The second is to increase the base of use of a product both in number of devices and number of sources and hence gain a wider acceptance of the product as an industry standard.

In summary, a major point in this technological change-marketing path baseline is the development of the understanding that the applications market has an influence on the technological form of a product along with the physical laws governing the shape of that product and the manufacturing technologies that produce it. What the scenarios will try to structure then from this risk category is two possible continuations of the established trends so as to be able to structure the available technology packages and their support systems for VSTOL avionics with a probable baseline freeze of 1985.

2. The Systems Acquisition Strategy

The previous risk category primarily addressed the producers. This category addresses the user side. The thrust from users in any high technology area is to somehow create a set of standards so as to minimize repetitious investment in capital equipment and cost of ownership. A buffer of standardization is created to protect the user from the rapid pace of technological change.

Currently several different movements amongst the users of computing devices exist. First, within the military services, the Military Computer Family Architecture (CFA) committee has established an analysis which outlines what a military computer family requirement would be with regard to a standard instruction set and software support and then goes on to show with sound argument that the commercial PDP11 commercial family satisfies the requirement.

Within the Navy the Assistant Secretaries of the Navy for Installations and Logistics and for Research and Development have sent out a joint memo dated 30 March 1977 for comment, mapping out a short term and long term strategy to cope with the proliferation of computing devices. This memo establishes for the near term the continuation of the UYK7 and 20 in the surface Navy as the standard computer family, and the AYK14, a machine made in bit slice fashion from the LSI AMD 2900's and compatible with the UYK 20, as the Naval airborne interim standard. It calls for all future microcomputer/microprocessor acquisitions to be of devices capable of implementing these

respective instruction sets. In the longer term it calls for evaluation of the CFA proposal in light of the competing AYK14, UYK7 and 20 interim standards.

The AYK14 currently has possible homes in several Navy airborne projects (9-1).

Project	AYK14's
F18	1600
LAMPS III	205
AV8B	350
IEWS	300
TACOMJAM	80
HARM (Avionics)	361
TASSES	24
WIDEBAND DUAL MODE	1000
URAIDS	VARIABLE
DIFAR	350
CAINS IA	1000
LINK 11	285
PROTEUS	500
	6255*

*while not exact or official, these numbers are representative.

Most significant are the F/A-18 which will have two AYK14's together on a mil standard 1553 type bus, the LAMPS Mk III helo which will have one AYK14, the P3C Update III which may have an AYK14 in network to offload the at-capacity central computer, and the AV8B, the advanced Harrier for the Marine Corps. The importance of these projects will be seen in the scenarios. With the exception of LAMPS Mk III, the final direction of each of these projects is not yet firm as of this writing. The ultimate outcome of each of these projects is in some measure tied in with the VSTOL concept. Any one of a number of outcomes could occur, affecting the ultimate

base number of AYK14's in the Navy system. This will be a key factor in costing out the alternative computer architectures in this report via the scenario--neutral, optimistic, pessimistic case idea. That number, along with the following totals for existing airborne computers in the Navy, has considerable meaning when reflected on in light of the $S \cdot N \geq CNR + CR \cdot N$ equation inherent in LSI computing device development expressed earlier in this report (9-1).

Type of aircraft	Number of computers per aircraft	Number of aircraft	Total number of computers
E2C	3	36	108
A7D	1	400	400
A6E	1	200	200
S3A	5	165	825
P3C	1	150	150
F14	6	200	1200
			2883*

*while not exact or official, these numbers are representative.

The connection of the VSTOL project with the National Aeronautics and Space Administration also has significance in terms of the possible acquisition strategies which might come to pass. In the absence of a general aviation standardization committee, like the kind ARINC provides for commercial aviation, NASA has taken it on itself to provide such a forum. In late 1977, it will initiate a Request for Proposal (RFP) for a general aviation computer based avionics demonstrator with award to be made in mid 1978 and delivery to occur in 1980-81 for evaluation. Currently postulated answers to the RFP based on preliminary NASA work indicate a multiple computer network on a

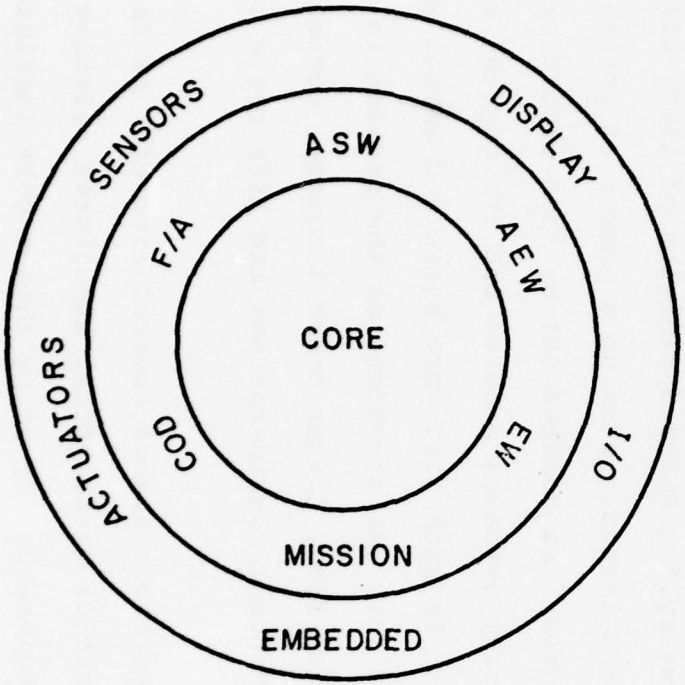
bus like mil standard 1553A or IEEE 488, with a multifunction display--not dissimilar with the F18/F15 solutions and the homogeneous alternative of this report. Form, fit, and function standardization is being sought to allow for advances in technology much like ARINC standards for commercial aviation. Although the environment for a general aviation aircraft may not be exactly the same as for VSTOL (although corporate type jets are in use by the Coast Guard and reach, in commercial use, the edge of the speed and maneuverability envelopes of their military sub-sonic counterparts), the degree if any to which the NASA effort links with the VSTOL effort could influence the cost of the implementation alternatives. This effect will be addressed in the scenarios. Even if the actual hardware is somewhat different, the concept of implementation and the interface standards might extend across general aviation and military applications.

Another very significant point in the acquisition strategy revolves around OMB circular A109. This systems acquisition circular among other things requires that early attention of industry be invited by a statement of mission needs vice specific hardware specifications in the RFP. The impact of this is that concept definition and validation are essentially obtained via industry participation with specific hardware implementation not delineated. Prior to OMB A109 a definite piece of hardware and consequently its inherent technological implementation became locked, very early in the acquisition process, into detailed specifications. In interviews with

airframe manufacturers and Navy software support activities, much of the "software cost problem" is viewed as really a problem of correct software but for a hardware specification overcome by the events of technological change and the resultant specification changes.

A position of this study made apparent by its very nature and more specific in the scenarios, is that if the Navy lab structure, i.e., the labs and their research arms in college campuses, private think tanks, and public institutions are considered as "industry," a wider definition of A109, without losing its spirit and intent, opens up a number of possible acquisition strategies. As pointed out by this report, the nature of the aircraft avionics problem and its digital computer implementation is such that very basic and useful work can be done via the Navy lab structure on the problem in terms of data flow, control flow, higher order language, coding of algorithms, architectural considerations and life cycle cost modeling.

If one first abstracts across several aircraft, as is done in this report, a set of mathematical structures relating the flow of data is obtainable. By doing this a core (navigation, ballistics, etc.) is obtained that would be common to all the mission variants of an aircraft like VSTOL. Each aircraft begins with that core and adds to it the mission variants. Beginning with that idea the process can be pictured as emanating from the center of concentric circles (Figure 9-2). Eventually the implementation of these functions occurs. In



CONCENTRIC VIEW OF VSTOL COMPUTATION REQUIREMENTS
Figure 9 - 2

the past in planes like the F4 it was in analog fashion. Now these mathematical structures are implemented by digital computing logic. This core would establish the way in which the mission variants and particularly what are called the embedded processing elements would interface, both initially and then across time as the system grew to meet added requirements.

The significance of this appears when, from empirical evidence, it is seen that the aircraft industry is organized by airframe manufacturers, engine makers, and various avionics and electrical shops that relate to specific mission variants or avionics functions. There are flight control shops, radar shops, electronic warfare shops, weapon systems shops, etc. Each shop as technology users will be exploiting LSI technology and replacing large amounts of circuitry by microprocessors and read only memory, in what is called embedded fashion, in their own piece of equipment. By going back to the concentric circles it can be seen that several different approaches to acquisition can be taken. The avionics shops, airframe and engine manufacturers, can be allowed to de facto decide what the core looks like by impinging toward the center of the concentric array from the outside. Or the user can define the core first as ARINC does for the commercial aviation world, and allow the circles to grow outward in an orderly fashion from the central core. Looking closely at commercial aviation, ARINC, a third party created by mutual consent of the airlines defines this core and an orderly fashion for interfacing to it.

NASA is now attempting to do this for general aviation. If the VSTOL project managers are thought of as the users, then the Navy lab structure could be viewed as the third party in the acquisition strategy to define that core. Note that this does not mean definition in terms of specific hardware, on the contrary it would be in more abstract terms of control and data flow, execution times, memory requirements, bus interface protocols, HOL algorithms, etc. This would structure then how the mission variants of the plane and embedded processing elements would interface with each other and the core. As with ARINC, this would be free of hardware implementation and allow technological change to be captured up to the point of baseline freeze. What the scenarios and their neutral, optimistic, and pessimistic cases will do is hypothesize possible acquisition paths and see how the cost of the two architectural implementations of this report would be affected.

3. Maintenance-Manpower System

In an economic analysis it is generally intended that the fallout of the analysis is the repair-discard, level of maintenance, and labor mix results. However it is often the case that organizational inertias or organizational optimization of other criteria than the economics of competing engineering alternatives prevail over the analysis results. This risk category therefore lays out some considerations that are later structured into possible outcomes in the scenarios.

As exhibited in articles in the electronic engineering trade journals, there is concern over job description because

of the fact that electrical circuit implementation is becoming in many areas, particularly process control, one of digital computing logic. The emotions attached to this can be expressed by the quote of one aerospace industry engineering head who stated emphatically that he didn't have any computer programmers, he had aerospace engineers that were called upon to program. The trade magazines such as Digital Design, Electronics, Spectrum, etc., have begun to run articles that address the electronic engineers' outlook with respect to this microprocessor/microcomputer phenomenon.

The relationship to the Navy maintenance-manpower system becomes visible through the viewing of businesses that are recognizing that their very organizational structures are being impacted by LSI technology in digital computing logic form. The separation between the computer science departments and the other engineering design departments is being obscured as the electronic engineers are being required to know digital computing logic and programming, and vice versa. Although the Navy maintenance-manpower system is a user vice designer system, a knowledge of the implementation of circuitry by digital computing logic is a requirement for understanding its maintenance. It is reasonable to expect that just as the electronics engineer design trades are impacted by LSI technology, so will the maintenance trades.

The meaning of this to the Navy is simply, or not so simply, NEC and rating restructuring. The not so simply comes in the sense that a maintenance-manpower system is not created

overnight. Just as the electronics engineer is concerned with his educational preparation and the protection of this human capital investment in the face of LSI digital computing technology, so will the maintenance man. LSI digital computing technology might dictate via economic analysis that a solution to the maintenance-manpower system structure be one group of people that understands built-in-test design and operation at the organizational level of maintenance, another that understands circuit board logic testing at the intermediate level of maintenance, and those that understand digital computer program design, analysis and troubleshooting at the depot level. Restructuring NEC's and ratings to accomplish this, independent of whether the system is in an airplane, a ship, or a submarine, cannot be done instantaneously. It may not even be recognized as a positive goal by the manpower-personnel planning systems since the separation of maintenance ratings by warfare specialty creates other positive intra- and inter-organizational relationships. The scenario method is used therefore to address the possible relations of LSI to the maintenance-manpower system and the costing fallout, vice assuming the normative case.

Existing evidence that there is recognition of the possible effect of LSI on the maintenance-manpower system is the joint service symposium on Automatic-Test-Equipment held in June 1977. Each service presented its programs in the area, soliciting inputs from industry as well. Some possible relevant developments that could occur out of this tri-service program will be considered in the scenarios.

Another aspect of maintenance of no small significance is the software development and support of the digital computing logic applications software. There is more than one way to develop and support the software. If the aircraft avionics hardware and applications software procurement is bundled, support of that software could come from that source. If the procurement is unbundled, a software vendor or avionics shop could provide it. The Naval Air Systems Command has created its own applications software support activities in the Naval Air Development Center at Warminster, Pa., the Missile Test Center at Point Mugu, California, and the Naval Weapons Center, China Lake. They handle the operational flight programs (OFP) for the ASW, high performance fighter, and attack aircraft respectively. The current thrust is to pass to these software support activities control of the applications software after a certain point of testing in development. This task becomes more difficult as the use of digital computing logic implementation and its required software spreads throughout the aircraft in the form of microcomputer/microprocessor based systems.

The impact addressed in the scenarios is the effect in terms of the type of software development and support systems that might be decided upon as outlined in the generalization of cost issues in this report.

4. Employment

The VSTOL aircraft as with the LAMPS, and its drone predecessor the DASH, is more closely intertwined with the parent

ship that supports it operationally and for maintenance. The question of how much processing and display equipment the ship and aircraft should host respectively is heightened in the case of VSTOL because of the take-off gross weight restrictions of the VSTOL technology. Any discussion of VSTOL ultimately becomes a discussion of the Navy surface ship force structure, again no simple discussion.

Currently the Navy force structure is debated around three force levels, a 400, 500, and 600 ship Navy; and two force mixes, sea control and power projection. To understand the two mixes one must first understand the two most basic roles of the Navy. The first is the projection of strike power inland as embodied in the strike carrier task forces and the nuclear strike cruisers. The second is the control of the sea lanes as embodied in the smaller escort vessels and the hypothesized sea control ship now called the VSS or VSTOL support ship, about the size of the Amphibious Assault Helicopter Carrier (LPH). Alternative force structures were developed by this author from Navy testimony in Congress, a National Security Council study, and two Congressional Budget Office working papers.

The outcome of these alternative forces goes into providing the base number of VSTOL aircraft that could reasonably be expected to exist in the 1991-2001 time frame and hence the number of airborne computers. The maintenance structures that could reasonably be expected to exist were also developed from these alternative force structures.

C. SCENARIO (AIRBORNE STANDARD)

This scenario is meant to be interpreted in the metaphorical sense. Its intent is to relate events in time so as to establish how, most likely, slightly optimistic and slightly pessimistic cases might be generated from a collection of realistic events.

The scenario begins in the fall of 1977. The Naval Postgraduate School's report on distributed LSI microcomputing for VSTOL avionics has been received with interest. It is read in several places which include the Naval Air Systems Command Avionics and Software Support Offices (NAVAIR 533), the VSTOL Program Office (PMA 269), and as an input to the ASN IL/RD memo of March 1977. In November of 1977 the VSTOL RFP for conceptual studies goes out. The NASA RFP for the general aviation computer based, multiplexed bus, and multifunction display demonstration also goes out at about the same time.

In December of 1977 Norden, a division of United Technologies, makes first deliveries of the LSI11M, a military version of the DEC LSI11, announced in the summer of 1977 as an embedded use companion for their PDP11/34M. The military Computer Family Architecture committee which recommended the PDP11 family for the military form, fit, and function specification uses this to add emphasis to their analysis. The LSI11M instruction set is a subset of the PDP11/34M, would be compatible for embedded use in a bused network with the PDP11 family, and uses the existing PDP11/DEC family software

development and support systems available from a multitude of commercial sources and already in place in many military activities.

Intel Corporation's commercial success with the 8080 microprocessor family continues. The 8080 family as enhanced in the summer of 1977 by the speed improvements (a factor of 2) of the 8085 is marketed as the ad hoc industrial process control and consumer market standard. This marketing position is furthered in the fall of 1977 by the licensing and second sourcing of the 8080 family by National Semiconductor and AMD in the form of chip sets families and single board computers. Other successful entrants in the microprocessor/microcomputer market in terms of accumulated experience (devices sold) are the Motorola 6800, an 8-bit word length microprocessor, and the Texas Instruments 9900, a 16-bit word length microprocessor, all single or two chip microprocessors which also provide the base for those companies' single board microcomputers.

From the fall of 1977 to mid 1979 when the RFP for VSTOL avionics advanced development (concept definition and validation) is made, the technological-marketing path the microprocessor/microcomputer industry has taken is one of exploitation of circuit integration at the level of the classical computer building blocks of processor, memory, input/output, and timing devices. Memory technology in particular progresses on paths different than semiconductor processor technologies so that magnetic-bubble, floppy disc, and holigraphic memory forms provide alternatives for implementation, particularly for

random access mass or block memory. Because of systems designers' desire for flexible memory size, the single chip semiconductor computer in mid 1979 has not become the dominant industry implementation for process control, the consumer market, or the general purpose microcomputer.

The single chip semiconductor computer is however a limited commercial success by mid 1979 at the 4, 8 and 16-bit word length with a resident memory capacity of 4K words ROM and a rated throughput of 0.5 million instructions per second. By 1985 it is anticipated that the single chip computer will have replaced the microprocessor plus separate read only memory chip as the implementation for process control and the consumer market at the 8 and 16-bit word length with a resident memory capacity of 16K words ROM and a rated throughput of 1 million instructions per second. It will not however be the primary implementation for the general purpose microcomputer. Intel, Motorola and Texas Instruments remain the dominant forces in these markets and their 8080/8048, 6800, and 9900/9940's the dominant families.

The ASN IL/RD in mid 1978 accepts with minor modification the mid term and long term strategy indicated in their memo of 30 March 1977. The Computer Family Architecture committee with the growing use of the LSI11M receives endorsement of the PDP11 family in mid 1978 as the basis for a military computer family. The AYK14 and UYK7 and 20 programs receive a DOD variance endorsement however to proceed in parallel, as already established, for their respective Navy shipboard and airborne

communities. A further evaluation point for the ASN IL/RD memo is set for mid 1979. This timing coincides with the RFP for VSTOL advanced development (validation and definition).

The structure of the Navy shipbuilding program is coalescing with the President's budget for FY80, moving in the direction of a 500 ship Navy. Thirty-two ships will be capable of landing and launching VSTOL aircraft. The avionics conceptual studies proposals, replies, and awards show general agreement that the following ship types, CV, CVN, and VSS (LPH size ship), are under review as organizational and intermediate level maintenance and multi-mission operations platforms. The strike cruiser CGSN is under consideration for single-mission operations and organizational level support. The maintenance strategy is generally being conceived as one of built-in-test (BIT) monitoring and line replaceable units (LRU) removal and replacement at the organizational level with the discard or repair of modules decisions made at the intermediate level of maintenance, and further repair made at the depot level. LRU size is accepted as measured in ATR (ARINC air transport racks) dimensions with modules measured in Standard Electronic Modules (SEM2A) which are increments of ATR's.

The defense budget being formulated in mid 1979 for FY81, funds completely the F/A-18 program production. The implication of this is that the AYK14 will have a permanent home in the Navy fighter/attack aircraft inventory with 800 F18's programmed for by 1990 and about 100 F18's programmed to be

off the production line by early 1980. Additionally, the P3C Update III is to be implemented by an AYK14 networked with the existing central processor on a mil standard 1553 type bus. Both of these decisions are in keeping with the near term strategy of the ASN IL/RD memo. The AV8B Harrier is dropped however in favor of the A4M and F/A-18, and the coming VSTOL program.

The mid 1979 reevaluation point for the ASN IL/RD memo is faced with three complications. First, the CFA committee findings opting for the PDP11 family is solidly backed by that families continued commercial success, particularly the LSI11 subset. Second, the growing AYK14 program has created a significant investment in that family. Third, the microprocessors plus memory chips and single chip microcomputer have proliferated in embedded use in weapons and tactical systems even in the face of the March 1977 memo. The proliferation is at the point that several types already exist in the Navy inventory supporting such diverse tactical and weapons system implementations as tactical aircraft landing gear retraction, Marine Corps electronic battlefield devices, and guidance devices for remotely piloted vehicles. A non-exhaustive list of these types are the AMD2900, LSI11M, the Intel 8080, 8085, 8048, 8748, the Motorola 6800's and the TI 9900, 9940's. The implication here is the proliferation of their respective software development and support systems and interfacing requirements. Also complicating the issues laid out in the memo and its mid 1979 reevaluation, is the

inconclusive nature of the direction taken by the automobile industry from the fall of 1977 to mid 1979. Although expected to be very large quantity users and hence prime movers in the standardization of the microprocessor/microcomputer industry, the competition between the big three and also between their respective model divisions in terms of exploiting the LSI microcomputer/microprocessor technology has held up this standardization. An ad hoc standardization has occurred within model divisions. This standardization is on the basis of an 8-bit word length (with the exception of Chrysler) by late 1978 and then on the Motorola 6800 at Ford, the Intel 8080 at GM, and the TI 9900 at Chrysler as their respective standard instruction sets by mid 1979. An IEEE 488 multiplex bus and its protocols and interface standards are in the concept stage for production implementation by the 1983-1985 model year time frame.

The importance of the automobile industries' position with respect to the military is in the fact that the temperature range faced by electronic circuitry in the environment of the automobile engine is a sustained -75°C to +450°C (9-2),300°C more than that required by mil specs for LSI/IC's. Hence, the automobile industry as a potential 10 million unit per year user of microprocessor/microcomputer based process control systems could by the volume of their use remove the extra factor of 2 to 3 the military currently pays for its microprocessor/microcomputer devices. Additionally, the automobile makers could create the instruction set, interface, bus and

protocol, and power standards in volume which general aviation and the military could accept as their own.

The NASA contract for a demonstration general aviation avionics is let in April 78 in answer to the late 1977 RFP. By 1980-81 the demonstrator is in operation using a mil standard 1553 type of bus, and a distributed network of Intel 8080 family microprocessors plus ROM chips type of architecture, and a multi-function display.

In light of these events, in particular the growing AYK14 base, and the influence of the ASN RD/IL memo, the RFP for VSTOL advanced development avionics definition and validation is structured so as to indicate selection of the LSI bit slice AYK14 family as the airborne computer in VSTOL. An additional anticipation in mid 1979 from the RFP is that the HSX, and VPX would be built from the same AYK14 family as the VSTOL. As a minimum then, a NAVAIR standard airborne computer family emerges. It is also anticipated that the embedded microprocessors and single chip computers, even if procured from separate sources, be made to conform with the AYK14 standard as to word length, instruction set, and interface on a mil standard 1553 type bus with either a shielded twisted pair or fiber optic implementation. It is further outlined in the RFP that this AYK14 family be targetable from the DOD HOL. With this in mind, the VSTOL program avionics advanced development definition and validation proceeds in early 1980.

Additional events between 1980 and 1985 affect the results of this process. One is the Navy part of the Tri-Service Ad

Hoc Automatic Test Equipment Project. It is decided to continue with the VAST system concept incorporating the AYK14 family into it.

Electronics/avionics technicians, like their commercial counterparts, begin to develop skills in the fundamentals of digital computing as a part of the recognition by the Naval Training Commands of the similarities in engineering/maintaining microcomputing devices that cross traditional rates and equipments. Although no rating consolidations take place in this 1980-85 time frame, the climate is set and some NEC (Navy Enlisted Classification Codes) are consolidated between the ET, AT, and DT rates.

This scenario will be used to cost the two architectural alternatives, the heterogeneous with AYK14 LSI bit slice computers in the avionics core networked with AYK14 subsystem front end microprocessors, and the homogeneous with a distributed network of commercial microcomputers made to meet mil specs in both the core and subsystem front ends. Each system concept will be assumed developed by an avionics computer shop during the VSTOL concept validation advanced development stage, rather than by the lab structure or the VSTOL airframe manufacturer. The most likely or neutral case, slightly optimistic, and slightly pessimistic cases of this scenario will develop costs around hypothesized annual rates of growth of the AYK14 standard family and commercial microcomputers.

D. SCENARIO (COMPUTER FAMILY ARCHITECTURE STANDARD)

This scenario is meant to be interpreted in the metaphorical sense. Its intent is to relate events in time so as to establish how most likely, slightly optimistic and slightly pessimistic cases might be generated from a collection of realistic events.

The scenario begins in the fall of 1977. The Naval Postgraduate School's report on distributed microcomputing for VSTOL avionics has been read by the Naval Air Systems Command Avionics and Software Support Offices (NAVAIR 533 and NAVAIR 360), the VSTOL Program Office (PMA 269), and as an input to ASN IL/RD memo of March 1977. Another interested reader is the NASA study group working on the RFP (to be released in late 1977) for a general aviation computer based, multiplexed bus, and multi-function display avionics demonstrator. The NASA RFP for the demonstrator and the Navy VSTOL RFP (to be released in late fall of 1977) for conceptual studies, although independently conceived, are each reciprocally tracked in recognition of the similarity of their purpose.

The Military Computer Family Architecture committee also recognizes the similarity of purpose in the NASA and VSTOL RFP's. Norden (a division of United Technologies) begins delivery of the LSI11M in December 1977, a military version of the DEC LSI11, announced in the summer of 1977 as an embedded processing companion for the PDP11/34M. The LSI11 is a multichip processor containing a subset of the PDP11/34M instruction

set, supported by the same software, and capable of being used in embedded fashion and/or in a bused network with the PDP11M family.

The two RFP's and the LSI11M are used as evidence by the Military Computer Family Architecture committee to further the position of their analyses for a set of form, fit, and function specifications around the PDP11/LSI11 family. The point made is that this family will capture the existing commercially available software development and support systems for not only military applications but also for general aviation as well.

Intel Corporation's commercial success with the 8080 microprocessor continues. The 8080 family as enhanced in the summer of 1977 by the speed improvements (a factor of 2) of the 8085 is marketed as the ad hoc industrial process control and consumer market standard. This marketing position is furthered in the fall of 1977 by another second sourcing of the 8080 family by National Semiconductor and AMD in the form of single chip processors and single board computer packages. Other successful entrants in the microprocessor/microcomputer market continue to be the Motorola 6800, and the Texas Instruments 9900, both single chip microprocessors.

From the fall of 1977 to mid 1979 when the RFP for the VSTOL avionics advanced development (definition and validation) is made, the technological change-marketing path the microprocessor/microcomputer device industry has taken is one of rapid continuation of circuit integration to the point of

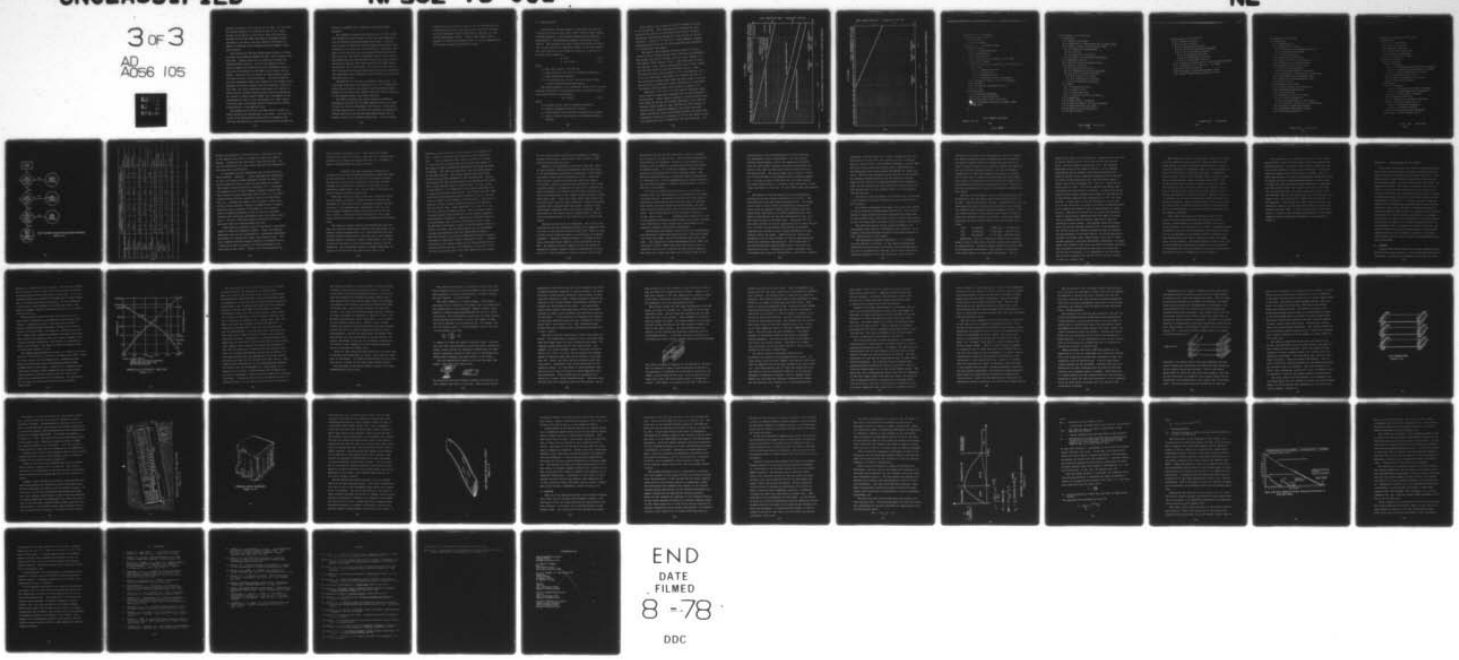
having all of the classical computer building blocks, processor, memory, input/output, and timing devices on one chip in the form of the Intel 8048's, TI 9940's, and a Motorola product as yet undesignated. Although alternative memory implementations such as the magnetic-bubble, floppy disc, and holigraphic forms exist, the single chip MOS implementation dominates them by mid 1979, providing more than enough memory flexibility and speed for applications designers. Hence, the single chip computer becomes the dominant implementation for process control, the consumer market, and the general purpose microcomputer market, replacing the microprocess plus ROM chips arrangement. By mid 1979 it has an 8-bit or 16-bit word length and 16K or 8K bytes of ROM respectively and a rated throughput of 0.5 million instructions per second. By 1985 this has increased to an 8-bit or 16-bit word length each with 64K bytes ROM and a rated throughput of 1 million instructions per second. This single chip computer continues to dominate the process control, consumer, and general purpose microcomputer market by 1985. In fact, the hand held calculator and general purpose microcomputer market has merged because of this by 1985. It is not uncommon to see from 4 to 16 of these computers-on-a-chip, each mounted in separate or stacked chip carriers mounted on a reflux soldered ceramic board linked with the parent companies bus system by 1985. Motorola, Intel, and Texas Instruments are the strongest entrants in this segment of the semiconductor industry.

The ASN RD/IL in mid 1978 accepts with some important modifications, the mid and long term strategy indicated in their memo of 30 March 1977. Influenced by the rapidity with which heavy industrial process control is being implemented by single chip computers, particularly in the automobile industry, and also by the success of the PDP11M/LSI11M, the strategy endorses the idea of a military computer architecture family with the PDP11/LSI11 family as the leading candidates but reserves comment on the ultimate long term standard to review the progress of the single chip computer.

Because they are already established, the UYK7 and 20 programs receive endorsement for continuation as the interim standard for shipboard application. However, when the F18 program and the AV8B Harrier do not receive the production go ahead decision for Fiscal 79 in favor of increased emphasis on VSTOL (with the A7/A4M/F14 filling the gap), the AYK14 loses the substantial part of its base as the airborne standard computer. Because of this, two other major AYK14 programs, the P3C Update III and the LAMPS Mk III program, are in doubt as to how to proceed. The P3C update does not happen with the AYK14 networked to the central computer but rather with a redesign of its software, particularly the Omega subroutine. A complete internal overhaul of the computing system based on a distributed network of elements of the military computer family (when selected) is projected as the ultimate solution. The LAMPS Mk III project, further along, uses the AYK14's already contracted for at a projected base of about 300. A

AD-A056 105 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 1/3
A STUDY OF ALTERNATIVES FOR VSTOL COMPUTER SYSTEMS.(U)
APR 78 U R KODRES , J D BUTTINGER
UNCLASSIFIED NPS52-78-001 NL

3 OF 3
AD
A056 105



END
DATE
FILMED
8-78
DDC

further evaluation point relevant to the AYK14 for the Naval Air Systems Command of the ASN RD/IL memo is not needed because of its demise in the F/A-18, AV8B, P3C decisions. The mid 1979 RFP for VSTOL therefore is developed without reference to the AYK14 and only refers to the desire to adhere to elements of the ultimate military computer family when selected.

The structure of the Navy shipbuilding program is coalesing with the President's budget for FY80, moving toward a 600 ship Navy. Seventy ships will be capable of landing and launching VSTOL aircraft. All will be capable of multi-mission support and organizational and intermediate maintenance with the exception of the CGSN strike cruiser and the DD963H. These two will be capable of single-mission support and organizational level maintenance. The return from the avionics conceptual studies and tracking of the Automatic Test Equipment Tri-Service Project show that advances in built-in-test (BIT) will mean preventive maintenance will consist of monitoring BIT program indicators and corrective maintenance can consist of faulty module (SEM2A size) replacement at the organizational level or line replaceable unit (LRU) removal of the ATR size, with module discard taking place at either organizational or intermediate level.

The demise of the F/A-18, and AV8B Harrier in favor of VSTOL and hence the reduced base of the AYK14, plus the rise in use of the PDP11M/LSI11M family has not stemmed by mid 1979 the proliferation of other microprocessing/microcomputing

devices in embedded use in mission oriented and sensor equipment.

The automobile industry has from the fall of 1977 to mid 1979 followed a decisive path in their use and standardization of microprocessors/microcomputing devices. After an initial time of competition between model divisions they have responded to the rapid pace of integration of the entire computer to the chip level so that by mid 1979 they have developed form, fit, and function standards based on word length, multiplexed bus type and instruction set by corporation, with Ford choosing the Motorola 6800 family, GM the Intel 8048 family and Chrysler the TI 9940 family. The result is that by mid 1979 distributed processing demonstrators have been established with production units numbering in the millions to go into the 1981 model year cars.

The NASA RFP of late 1977 is awarded in April 1978. By late 1980, early 1981, the results are in, the TI 9940 based family being chosen as the general aviation standard family because of its 16-bit word length, wide usage base by Chrysler in distributed fashion, and second sourcing.

The VSTOL validation and concept definition advanced development proceeds with the NASA demonstration as an input to the core avionics development, particularly in the multifunction display area. The Navy lab structure is awarded the concept definition and advanced development phase vice an avionics shop or the airframe manufacturer. The two leading

candidates costed from this scenario are the PDP11M/LSI11M in a heterogeneous network and either one of the Intel, Motorola, or Texas Instruments single chip computer families in a completely homogeneous network. (The TI 9940 family will be chosen for illustration purposes since it is also hypothesized to be chosen by NASA for general aviation.)

E. COSTING RESULTS

Illustrative costing results are generated from the two scenarios for each of their three cases--neutral (most likely), slightly optimistic, and slightly pessimistic--for both the heterogeneous and homogeneous computer architecture alternatives. Two equations are used in conjunction with the narrative of the scenarios to determine several of the cost numbers. The first equation is the production learning or industry experience curve formulation.

$$y = ax^b \qquad (9\text{-}1)$$

$$b = \log S/\log 2 \qquad (9\text{-}2)$$

where

y = unit cost (price) of the $x^{\underline{th}}$ unit

a = cost (price) of the initial accumulated quantity, A

x = total accumulated quantity

S = % of previous cost (price) that cost (price) drops to when accumulated quantity doubles.

The second equation is the compounded annual rate of growth equation used often in the business world to describe alternative future product sales outcomes.

$$mA = A(1+g)^T \qquad (9\text{-}3)$$

where

m = multiple of the initial accumulated quantity

A = initial accumulated quantity in units

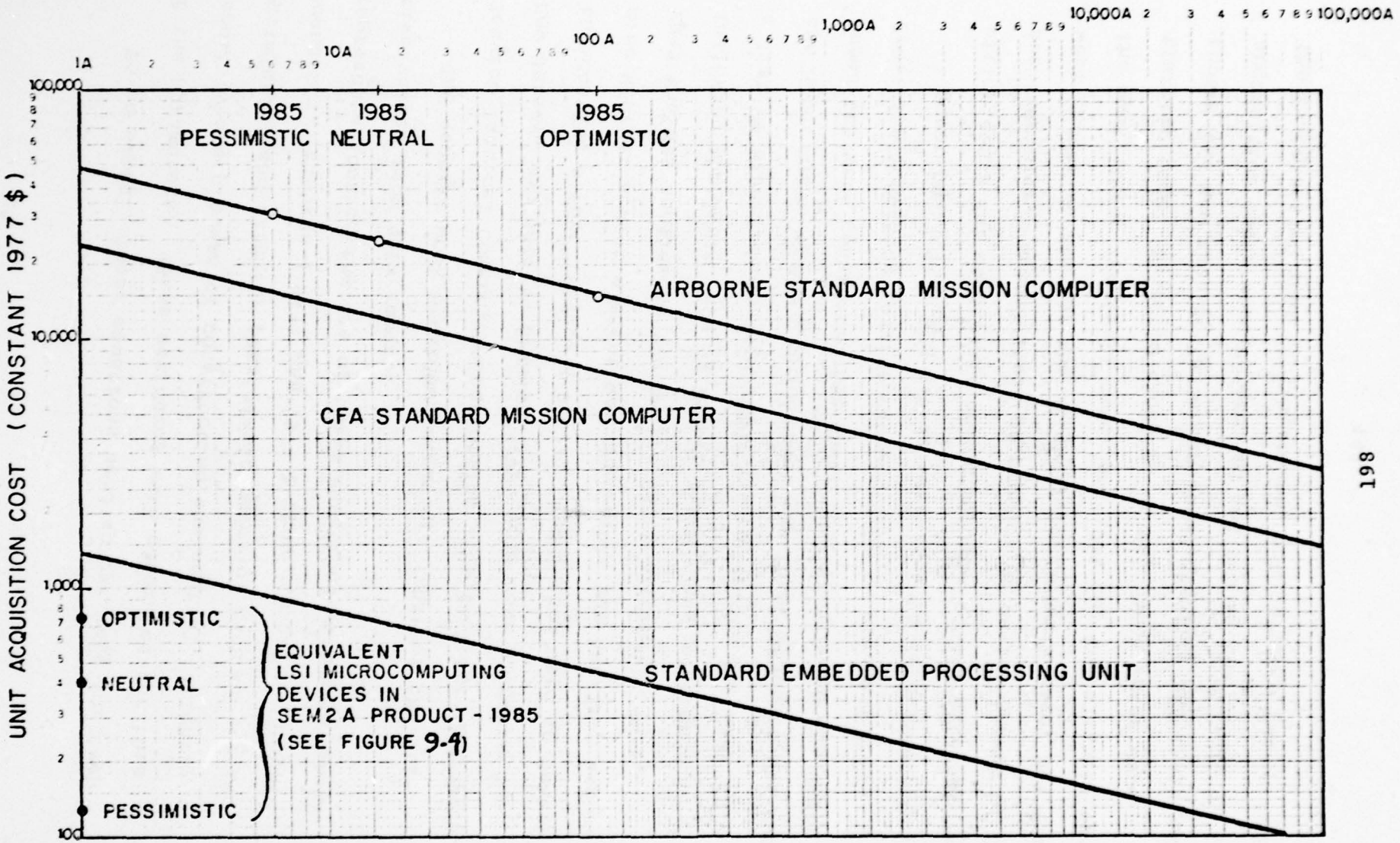
g = annual growth rate in percent expressed as a decimal

T = years to attain the multiple of accumulated quantity desired.

To be useful, these equations need an estimate for each of the parameters. These estimates were generated for this report by the process of constructing the neutral, slightly optimistic, and slightly pessimistic cases of each of the two scenarios. Figures (9-3) and (9-4) show composite, graphic representations of the use of these equations with representative values from this report.

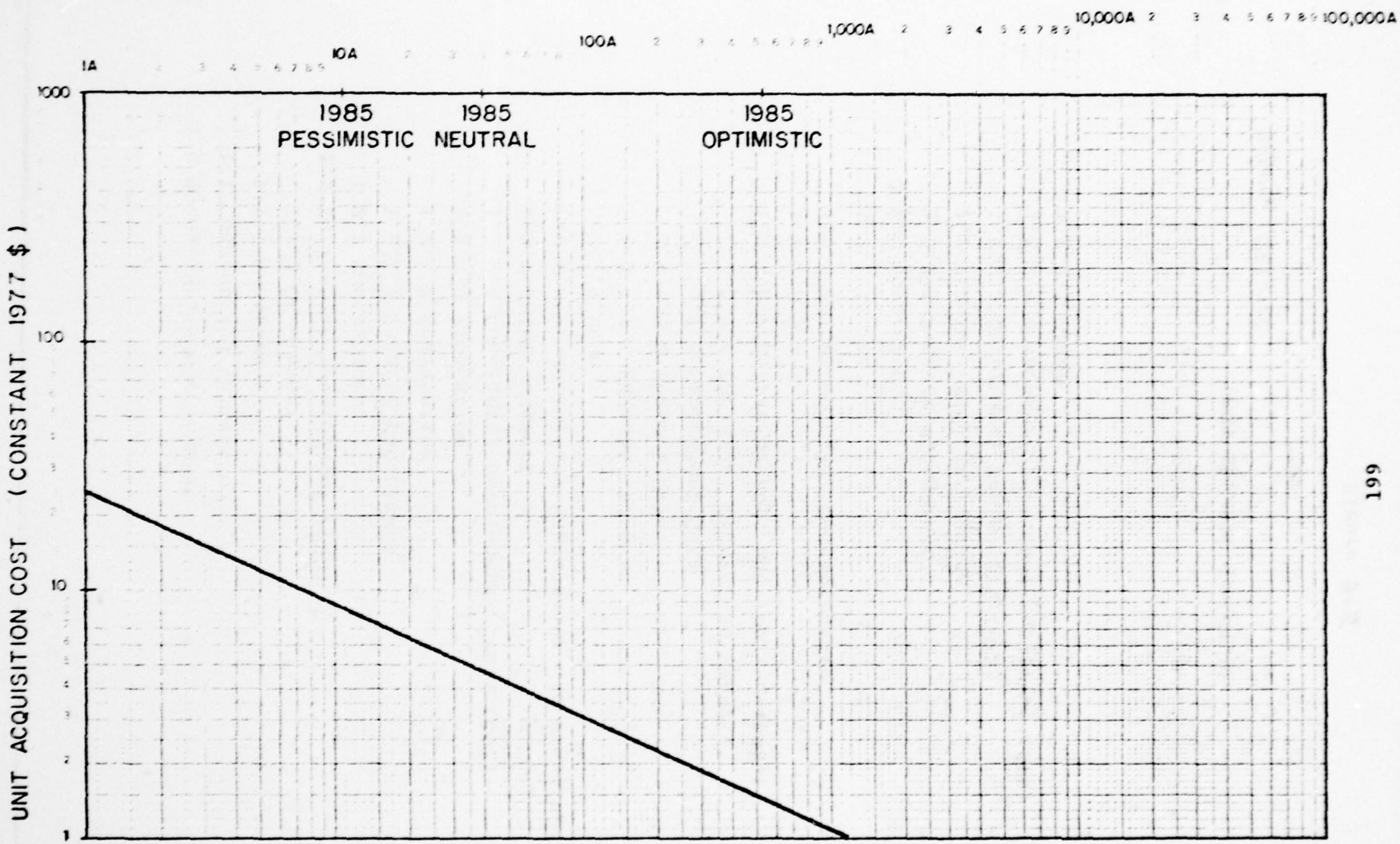
The generic work breakdown structure (WBS) of the Tri-Service Tactical Communications office (TRI-TAC) Fort Monmouth, New Jersey, was used to develop life cycle cost elements for the costing effort (Figure (9-5)). The method used to determine whether a cost element was applicable was to first assume that the WBS applies to the entire VSTOL aircraft. Then, following the logic flow of Figure (9-6) from reference (9-3), a list of significant, applicable cost elements for each of the alternative architectures under the three cases of the two scenarios were considered for their contribution to the VSTOL. Preliminary costing results are pictured in Figure (9-7).

Equations (9-1) and (9-3) were used to determine acquisition costs for each basic computing unit of the alternative architectures. Annual rates of product or device growth of use were chosen as 30% for the slightly pessimistic case, 50% for the most likely (neutral) case, and 100% for the slightly optimistic case. Selected examples of why these rates were chosen and how the initial accumulated quantity and their unit acquisition costs were determined and led to the values in 1985, the hypothesized year of comparison, are outlined below. All



EFFECT OF ALTERNATIVE GROWTH RATES OF STANDARD COMPUTING DEVICES ON ACQUISITION COST WITH 85% EXPERIENCE CURVE.

Figure 9-3



EFFECT OF ALTERNATIVE GROWTH RATES OF MICROCOMPUTING DEVICES ON AQUISITION COST WITH 73% EXPERIENCE CURVE.

Figure 9-4

1.0 Research and Development
  1.1 Concept and Validation
    1.1.1 Contractor
      1.1.2 Government
  1.2 Full Scale Development (FSD)
    1.2.1 Contractor
      1.2.1.1 Program Management
      1.2.1.2 Engineering
      1.2.1.3 Fabication
      1.2.1.4 Contractor Development Tests (CDT)
      1.2.1.5 Test Support
      1.2.1.6 Producibility Engineering and Planning (PE
      1.2.1.7 Data
        1.2.1.7.1 Engineering Data
        1.2.1.7.2 Support Data
        1.2.1.7.3 Management Data
        1.2.1.7.4 Technical Orders and Manuals
      1.2.1.8 Peculiar Support and Test Equipment
      1.2.1.9 Other
      1.2.1.10 General and Administrative
      1.2.1.11 Fee
    1.2.2 Government
      1.2.2.1 Program Management
      1.2.2.2 Test Site Activation
      1.2.2.3 Government Tests (DTE/IOTE)
      1.2.2.4 Government Furnished Equipment (GFE)
      1.2.2.5 Other

COST ELEMENT STRUCTURE

Source: TRI-TAC

Figure 9-5

2.0 Investment (Non-Recurring)

2.1 Contractor

2.1.1 Program Management

2.1.2 Producibility Engineering and Planning (PEP)

2.1.3 Initial Production Facilities (IPF)

2.1.3.1 Production Engineering

2.1.3.2 Tooling

2.1.3.3 Industrial Facilities

2.1.3.4 Manufacturing Support Equipment

2.1.4 Technical Support

2.1.5 Initial Spares and Repair Parts

2.1.6 Initial Training

2.1.6.1 Training Facilities

2.1.6.2 Training Devices and Equipment

2.1.6.3 Initial Student Training

2.1.6.3.1 Operator Training

2.1.6.3.2 Maintenance Training

2.1.6.3.3 Instructor Training

2.1.7 Data

2.1.7.1 Engineering Data

2.1.7.2 Support Data

2.1.7.3 Management Data

2.1.7.4 Technical Orders and Manuals

2.1.8 Leaseholds

2.1.9 Common Support Equipment

2.1.10 Peculiar Support and Test Equipment

2.1.11 Other Non-Recurring Costs

2.1.12 General and Administrative

2.1.13 Fee or Profit

Figure 9-5 (continued)

2.2 Government (Non-Recurring)

2.2.1 Program Management

2.2.2 Initial Training

2.2.2.1 Training Facilities

2.2.2.2 Training Devices and Equipment

2.2.2.3 Initial Student Training

2.2.2.3.1 Operator Training

2.2.2.3.2 Maintenance Training

2.2.2.3.3 Instructor Training

2.2.3 Production Acceptance Test and Evaluation (PATE)

2.2.4 Operational Test and Evaluation (OTE)

2.2.5 Test Site Activation

2.2.6 Government Furnished Equipment (GFE)

2.2.7 Other Non-Recurring Investment Costs

Figure 9-5 (continued)

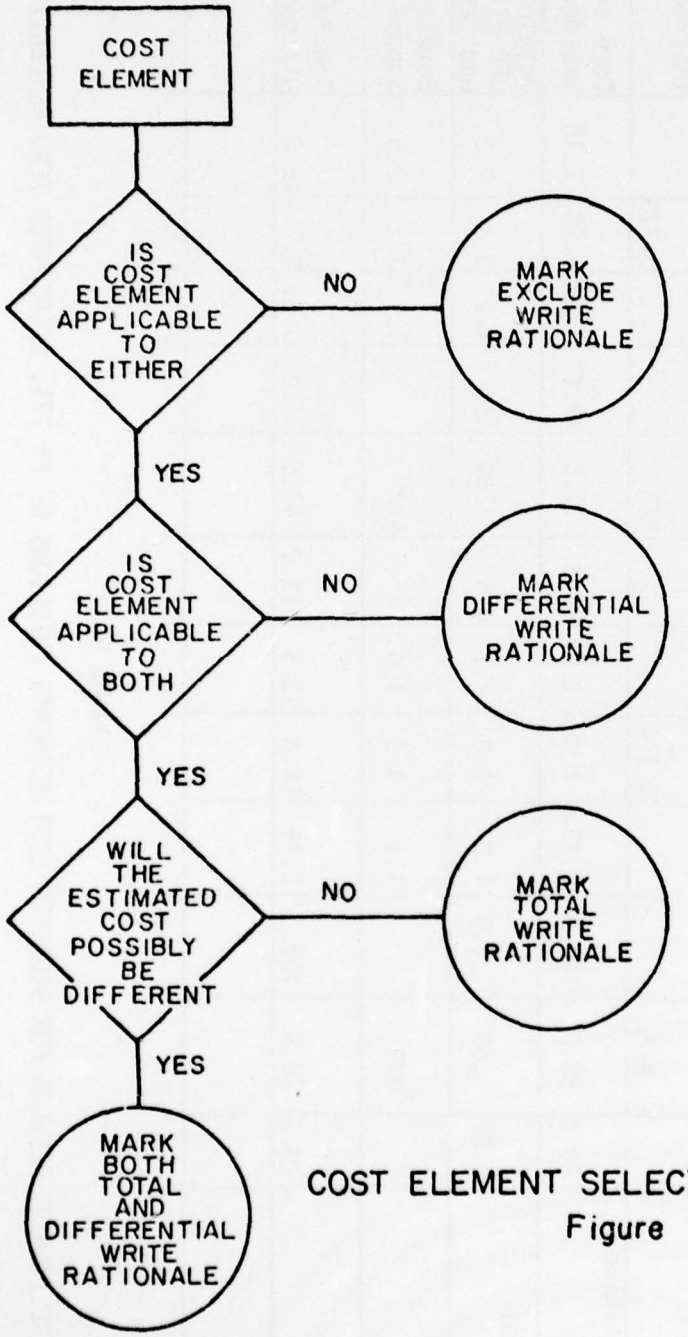
3.0 Investment (Recurring)
- 3.1 Contractor
  - 3.1.1 Manufacturing
  - 3.1.2 Production Material
    - 3.1.2.1 Purchased Equipment and Parts
    - 3.1.2.2 Subcontracted Items
    - 3.1.2.3 Other Material
  - 3.1.3 Sustaining Engineering
  - 3.1.4 Quality Control and Inspection
  - 3.1.5 Packaging and Transportation
  - 3.1.6 Operational/Site Activation
    - 3.1.6.1 Site Construction
    - 3.1.6.2 Site/Ship/Vehicle Conversion
    - 3.1.6.3 Assembly, Installation and Checkout
  - 3.1.7 Other Recurring Investment Costs
  - 3.1.8 General and Administrative Costs
  - 3.1.9 Fee or Profit
- 3.2 Government (Recurring)
  - 3.2.1 Quality Control and Inspection
  - 3.2.2 Sustaining Engineering
  - 3.2.3 Transportation
  - 3.2.4 Operational/Site Activation
    - 3.2.4.1 Site Construction
    - 3.2.4.2 Site/Ship/Vehicle Conversion
    - 3.2.4.3 Assembly, Installation and Checkout
  - 3.2.5 Technical Orders and Manuals
  - 3.2.6 Government Furnished Material
  - 3.2.7 Other Recurring Cost

Figure 9-5 (continued)

4.0 Operating and Support Costs (O&S)

- 4.1 Operations
  - 4.1.1 Electrical Power
  - 4.1.2 Special Materials
  - 4.1.3 Operator Personnel
  - 4.1.4 Operational Facilities
  - 4.1.5 Equipment Leaseholds
  - 4.1.6 Other Operations Costs
- 4.2 Logistic Support
  - 4.2.1 Maintenance
    - 4.2.1.1 Personnel
      - 4.2.1.1.1 Organizational Maintenance Personnel
      - 4.2.1.1.2 Intermediate Maintenance Personnel
      - 4.2.1.1.3 Depot Maintenance Personnel
    - 4.2.1.2 Maintenance Facilities
    - 4.2.1.3 Support Equipment Maintenance
    - 4.2.1.4 Contractor Services
  - 4.2.2 Supply
    - 4.2.2.1 Personnel
      - 4.2.2.1.1 Organizational Supply Personnel
      - 4.2.2.1.2 Intermediate Supply Personnel
      - 4.2.2.1.3 Depot Supply Personnel
    - 4.2.2.2 Supply Facilities
    - 4.2.2.3 Spare Parts and Repair Material
    - 4.2.2.4 Inventory Administration
      - 4.2.2.4.1 Inventory Management
      - 4.2.2.4.2 Inventory Holding
    - 4.2.2.5 Transportation and Packaging
  - 4.2.3 Other Logistic Support Costs

Figure 9-5 (continued)



COST ELEMENT SELECTION DECISION PROCESS
Figure 9-6

Cost Element	Description	Airborne Standard Scenario						CFA Standard Scenario						Remarks
		Heterogeneous			Homogeneous			Heterogeneous			Homogeneous			
		O	N	P	O	N	P	O	N	P	O	N	P	
1.2 1.2	FSD Engineering		N/A			1.5			N/A			1.5		Sem 2A Product Development
1.2 1.2	FSD Engineering		245.0			26.0			245.0			19.0		A/C Design Weight Penalty
1.2 1.8	FSD Peculiar S/T Equip.	0.06	0.15	0.24	0.02	0.03	0.05	0.06	0.15	0.24	0.02	0.03	0.05	ATE Programming
2.15	INR Initial Spares		N/A		0.07	1.2	8.6		N/A		0.04	0.6	4.3	One Year Supply
2.16	INR Initial Training		N/A			0.114			N/A			0.129		Instructors/Students
3.1.1 3.1.2.1 3.2.6	IR Acquisition Cost	48.4	80.0	100.0	0.495	1.65	2.97	18.8	31.5	40.6	0.37	1.20	2.16	Core and Embedded
	Applications Software	0.57	1.60	3.0	1.2	3.8	7.2	0.57	1.60	3.0	1.2	3.8	7.2	Original Attack OFP Plus Three Updates
	Software Tools		N/A		1.1	4.2	7.6		N/A		1.1	4.2	7.6	Development and Support
4.0	O/S Total	29.3	70.6	102.8	2.19	14.8	93.5	15.9	40.5	53.2	0.66	9.3	46.8	Ten Years, 77$ Discounted at 0%

FIGURE 9-7

PRELIMINARY RESULTS FOR SELECTED COST ELEMENTS (MILLIONS OF FY 77$, DISCOUNTED ZERO PERCENT)

dollars are expressed in current dollars. Inflation, the rise in the general price level, is assumed to affect all inputs equally. Discounting was done using a zero percent rate. Using the standard DOD 10% figure would only make the homogeneous case even more cost-effective.

To determine the unit acquisition cost of the naval airborne standard computers of the heterogeneous alternative in 1985, the number of these computers procured for naval airborne systems by 1985 as a first cut was chosen to be about 6,000 based on the projections of reference (9-1). Interviews with members of the military computer industry indicate that their industry average experience curve is 85%, i.e. as accumulated quantity doubles, acquisition cost to the user comes down 15%. Currently the annual rate of growth in units sold of the military computer industry is conservatively (slightly pessimistically) projected at 30% annually (9-4). Using an initial accumulated quantity of 500 for the naval airborne standard (from unofficial procurement plans and industry interview), a 30% and a 50% annual growth rate bracket the aforementioned 6,000 unit base projected for 1985.

Three things are worthy of note at this point in the development of these and subsequent figures. First, the industry experience curve can be interpreted in a negotiated procurement environment as a production (labor) cost learning curve. The naval airborne standard computer would be in production run on a fixed price type of contract for costing purposes in the time frame of this report. Second, the curve represents the

unit learning (experience) curve. The cumulative average learning curve would lie slightly above the unit learning curve, generating an average unit acquisition cost for a purchase in quantity slightly above the unit cost curve.

Finally, note that sustaining engineering is included as one of the recurring manufacturing costs that leads to the acquisition cost to the user by the computer industry. The meaning of this is that new models of the naval airborne standard would continue to incorporate innovations in LSI technology, bubble memory, etc., as the costs of the naval airborne standard continued to come down.

Returning to how the total of the unit acquisition costs were developed, the number of VSTOL aircraft in the buy was estimated at 1,000 from the force structure analysis referred to earlier. The heterogeneous architecture calls for two naval airborne standard computer units per aircraft for the core avionics. 2,000 units would be procured at unit acquisition cost of $30,000 for a total of $60,000,000 for the VSTOL fleet.

The airborne standard scenario has hypothesized that the Naval Air Systems Command would undertake the development of an airborne family microprocessor, not a microcomputer since the scenario assumed integration of circuitry only to the computer building blocks of processor, memory, input/output, and timing circuits at the point in time a development decision was hypothesized. The development program is assumed undertaken beginning in mid 1979 with the reevaluation point of the

ASN(RD/IL) memo referred to in the scenario, and completed in 1980. Based on a nonrecurring development cost of $150 per gate and an equivalent gate count of 10,000 for a single or a few chip LSI processor made from the military family of LSI chips (9-5),the nonrecurring development cost would be about $1,500,000. Note that this would be a sunk cost with respect to the VSTOL program. The initial accumulated quantity and acquisition cost was determined in the following way. Pricing two existing military microprocessors, the LSI11M and the AYK30, their single quantity price is about $2,500. In lots of 500 or more there is a 15% reduction in price which would make the in quantity unit price about $2,125. The growth rates for these two military microprocessors were estimated for up to the next couple of years at 100% annually from interviews. The price in quantity of each of these military microprocessors would be about $1,500 by 1980, using equations (9-1) and (9-3), an 85% curve and the information above. This is assumed used as a design-to-cost figure by Nav Air in the airborne standard microprocessor development. Based on the $1,500,000 nonrecurring development cost and the $1,500 design-to-cost figure, the developer would need at least an initial quantity order of 1,000 to at least cover his nonrecurring cost of development and make him competitive with the LSI11M/AYK30 products. This was chosen as the initial accumulated experience quantity available in 1980 after a year's development effort. A modest projection of the annual growth rate of microprocessor/microcomputer devices annually as an industry is projected at 50% for the next several years (9-6). This growth rate was assumed

for the naval airborne standard microprocessor in airborne systems as the neutral (most likely) case to begin in 1980 when the device would be ready.

Based on the hypothesized 1,000 plane VSTOL buy, and 10 of these airborne standard microprocessor devices per plane in the heterogeneous architecture alternative, a unit acquisition cost of about $850 was estimated by 1985. Since each naval airborne standard microprocessor would need memory (16K words), parallel and serial bus interfacing, and a floating point package to accomplish its task even in embedded use, these would have to be costed into the basic processing unit since a microprocessor is not capable of doing anything without these other items. Using the LSI11M, AYK30, ROLM Corporation and other price lists for these components, costing by analogy was done with a multiplicative cost factor of about 4 being surprisingly consistent across the various companies. This would mean for the neutral case, for example, about $3,400 for each microcomputing unit at the front end of an avionics subsystem.

Two other costs need to be considered. The first is the nonrecurring cost of programming Automatic Test Equipment (ATE) like that in the Versatile Avionics Shop Testor (VAST) for the printed circuit cards that would be in each piece of computer equipment. Although a standard computer is already in existence, each application requires a new test program for the ATE. The standard airborne computer has on the average 10 boards, each assumed "complex" under the description of reference (9-7). A 90% comprehensive median (neutral) cost of ATE

programming would be $15,000 per board, a total of $150,000 for the unit in the neutral case. The airborne standard microprocessor plus the equipment to make it a microcomputer is assumed to be six boards in a Navy Standard Electronic Module 2A (SEM2A) product. At $3,400 per airborne standard microcomputer of six boards that is about $600 per board. Each board was assumed moderate in complexity under the description of reference (9-7). At 90% comprehensiveness, median (neutral) cost for ATE programming of $5,000 a board would be a total of $30,000 for the product.

The last cost reviewed at this point is a recurring investment cost, the VSTOL flyaway cost attributed to the computing system weight. Reference (9-5) relates that every pound of operational systems weight contributes 6-8 lbs. to airframe, fuel system, and other supporting systems weight, and that a modern fighter aircraft costs out at $500 per pound of flyaway costs. This concept applied to the 50 pounds (2 units at 25 lbs. each) of the standard units versus the 2-4 lbs. of the homogeneous computing module makes for an order of magnitude differential in the flyaway costs attributable to the respective computer architecture alternatives.

The operating and support costs for both the heterogeneous and homogeneous alternatives were generated using the TRI-TAC Life Cycle Cost Model, and the B-K Dynamics Navy Billet Cost Model. The parameters of mean-time-between-failure (MTBF) and mean-time-to-repair (MTTR) were taken from existing data on comparable units for the airborne and CFA standard computers. The values were 2,200 hours MTBF with 20 minutes MTTR at the

organizational level of maintenance and two hours MTTR at the intermediate level of maintenance. For the airborne standard microprocessor, and for the homogeneous microcomputer the most likely, slightly optimistic and slightly pessimistic case idea was used since field reliability data is not mature enough. The values chosen were 2 times, 10 times, and .5 times the MTBF of the airborne and CFA standard computers. A figure of about 25,000 hours for the AYK30, LSI11M, and TI9900 demonstrated in a simulated field environment is the figure developed by interview. The same MTTR's were used throughout.

The costing in the CFA scenario for the heterogeneous alternative was also based on equations (9-1) and (9-3). The PDP11/34M and LSI11M were used as representative. The number of CFA units in use by 1985 was generated by starting with annual military computer industry sales employing LSI technology being conservatively (slightly pessimistically) estimated at $100,000,000 for 1977 and $250,000,000 by 1981 (9-4) This would be about a 30% growth rate. Reference (9-4) and interviews indicate an initial acquisition cost of about $25,000 for a 1/2 ATR, 16K work memory PDP11/34M and that Norden conservatively expects to grow to about $50,000,000 in sales annually within five years. Coupled with the initial unit acquisition cost of about $25,000, an initial experience base of 500 was considered representative. This yields about a $12,500,000 nonrecurring development cost if the first run amortizes this cost. Although not directly available, this development cost figure is representative. With an 85% industry

experience (learning) curve the average acquisition cost for the CFA unit for the VSTOL core avionics of the 1/2 ATR size would be about $15,000 in the slightly pessimistic case by 1985. With two of these units per aircraft and a 1,000 plane buy, the acquisition cost would be about $30,000,000 for these core units. The front end processing units would be as indicated in the airborne standard scenario with the exception that the factor of four would not be applied because of the assumption under the CFA scenario that the direction of integration is to more rapidly bring all the building blocks of a microcomputer into one LSI chip.

The costs of ATE programming, and flyaway cost attributable to the computing units were computed as in the airborne standard scenario.

No software development and support tool costs were costed to either scenario's heterogeneous alternatives. This is best casing of these alternatives since interviews have shown that in many projects these tools aren't used or require modification to suit the input/output structure of the particular application.

The VSTOL Operational Flight Program development and maintenance costs would be a differential cost element. It is computed using the technique outlined in the generalization of cost issues section of this report.

The homogeneous equipment acquisition cost is computed for each scenario as follows. In the airborne standard the architecture would consist of 24 single chip microcomputers in the core in two SEM2A sized modules, and 20 single chip microcomputers embedded as front end processors in the avionics

sub-system, ten boards of two microcomputer chips to a board. The airborne standard scenario hypothesizes that integration of computer building blocks to a single chip will not be as rapid in terms of memory size, processing speed, and input/output as in the CFA standard scenario. For the CFA scenario, integration to single chip computers progresses at a faster pace so that only 12 single chip microcomputers in one SEM2A sized module is needed in the core, with ten 10 boards of two microcomputers each as front end processing units. These numbers are representative in view of the technical information of this report.

The cost per microcomputer chip in either alternative was estimated for the most likely case of each scenario to be $5 by 1985. This was obtained as follows. References (9-6, 9-8) and interviews were used to obtain and check dollar sales volumes for industrial LSI microprocessor/microcomputer devices. The unit prices attached to each dollar sales volume to obtain a figure for quantity produced in each year is the Intel 8080A unit price for those periods since it was the industry price leader.

1974	$22,000,000	@ $500 each ≃	50,000 units
1975	50,000,000	@ $100 each ≃	500,000 units
1976	150,000,000	@ $ 25 each ≃	6,000,000 units

Admittedly, this is a rough way to obtain these figures; however, they do check with interview sources close enough to be representative. Available volume information in units sold of microprocessor/microcomputer chip devices is unreliable since these numbers are proprietary information. 1976 is

chosen as the base year of experience. Reference (9-6) predicts an annual industry growth rate of 50% for the next several years. This was taken as the neutral case. The industry experience curve is well known to be 73%, i.e. as industry accumulated quantity doubles, cost and price falls by 27%. Using the chosen initial experience accumulation of 6,000,000, the price at that quantity of $25, and the neutral annual growth rate of 50%, equations (9-1) and (9-3) yield the $5 device price by 1985. This figure checked with Delphi questionnaires and other forms of interview. It also falls in the range of total annual industry sales volume of $500,000,000 to $1,000,000,000 predicted from references (9-6, 9-8) for these devices. A final way in which the feasibility of such massive volume checks out, at least in an order of magnitude sense, is to consider the possible types of applications for these devices. Reference (9-6)quotes that of some 25,000 potential types of applications considered within the realm of single chip microcomputer process control, only 10% of them are currently in active design. Applications range from automobiles (10 million vehicles annually with 2-3 microcomputers per vehicle being the industry strategy to meet the emission and mpg standards of the early 1980's), 40 million appliances annually (microwave ovens, washer-dryers, TV's, etc.), point-of-sale terminals, precision measurement instruments, to arcade games, and children's toys (into the 100's of millions annually). With automobiles and appliances accounting for nearly 100 million devices annually, it is reasonable to see the other applications accounting for 200-300 million devices annually by 1985 as a neutral case.

These chips must still be placed into a system or product. Using the SEM2A size module with two chips per board and the reflux solder on ceramic technique described earlier, an estimate of nonrecurring development costs for such a product is about $1,500,000 based on industry interviews. Acquisition costs for such a product were obtained from interviews, literature (9-5), and checked against available price lists. The results indicate that a factor of about 2.5 times the sum of the acquisition cost of the LSI chips in the product can be used to account for the contribution of the printed circuit boards, interconnects, chassis, and power. Another factor of 2-3 accounts for the cost of making the product suitable for a severe environment. While seemingly rough factors, they check out surprisingly consistently across products in the minicomputer and microcomputer range both in terms of available data and industry interview.

Using these two factors and the neutral case of 50% annual growth for microcomputing devices which yields $5 devices, a SEM2A module of six boards with two devices per board would have an acquisition cost of $450. Two modules would be needed in the core of the homogeneous architecture alternative to be equivalent in performance with the airborne standard heterogeneous architecture alternative under the assumptions of that scenario. One module would be needed in the core of the homogeneous architecture alternative to be equivalent in performance with the CFA standard heterogeneous architecture alternative under the assumptions of that scenario.

This concludes the discussion of how the costing results were obtained. While not every case of the two scenarios was detailed, hopefully the reader gained enough to understand how the costing results of Figure (9-7)were obtained. A narrative summary of these results would be put as follows. For airborne applications, the rate of technological change and growth of general use of LSI microcomputing devices is great enough to offset any cost advantages of standardization of computers and software development and support tools even of the form, fit, and function type. The three significant areas of cost that manifest this are the acquisition cost, the operating and support costs, and the aircraft flyaway cost attributable to the computers. Even without the flyaway cost considered, the homogeneous case is the cost-effective alternative under both scenarios and all cases. If these results hinged on one condition, it would be the continued understanding of distributed computing and the continued development of the software to effect it.

Appendix A. GENERALIZATION OF COST ISSUES

This section presents for the reader a narrative tutorial on the cost issues involved in systems engineering with LSI based circuitry, particularly microcomputing devices. A few controversial things should be immediately pointed out for the initiated reader as being conjectured by this report, and supported elsewhere. The first is that while useful to an extent, cost per gate or cost per bit may no longer be as relevant a parameter for systems engineering with microcomputing/microprocessor devices as cost per device plus cost per interconnect of devices(2-1). Second, the cost of special hardening of a microcomputing/microprocessing device against radiation, shock, heat, and other environmental factors by the use of sapphire substrates, etc., may be overkill when its contribution to overall aircraft survivability is assessed. And finally, the actual cost contribution of software may not be minimized by adherence to standard computing devices nor even paramount when total systems engineering costs are considered (specifically aircraft weight penalty from standard computing devices in this report's analysis). With these ideas brought forward, some tutorial cost generalizations will now be described.

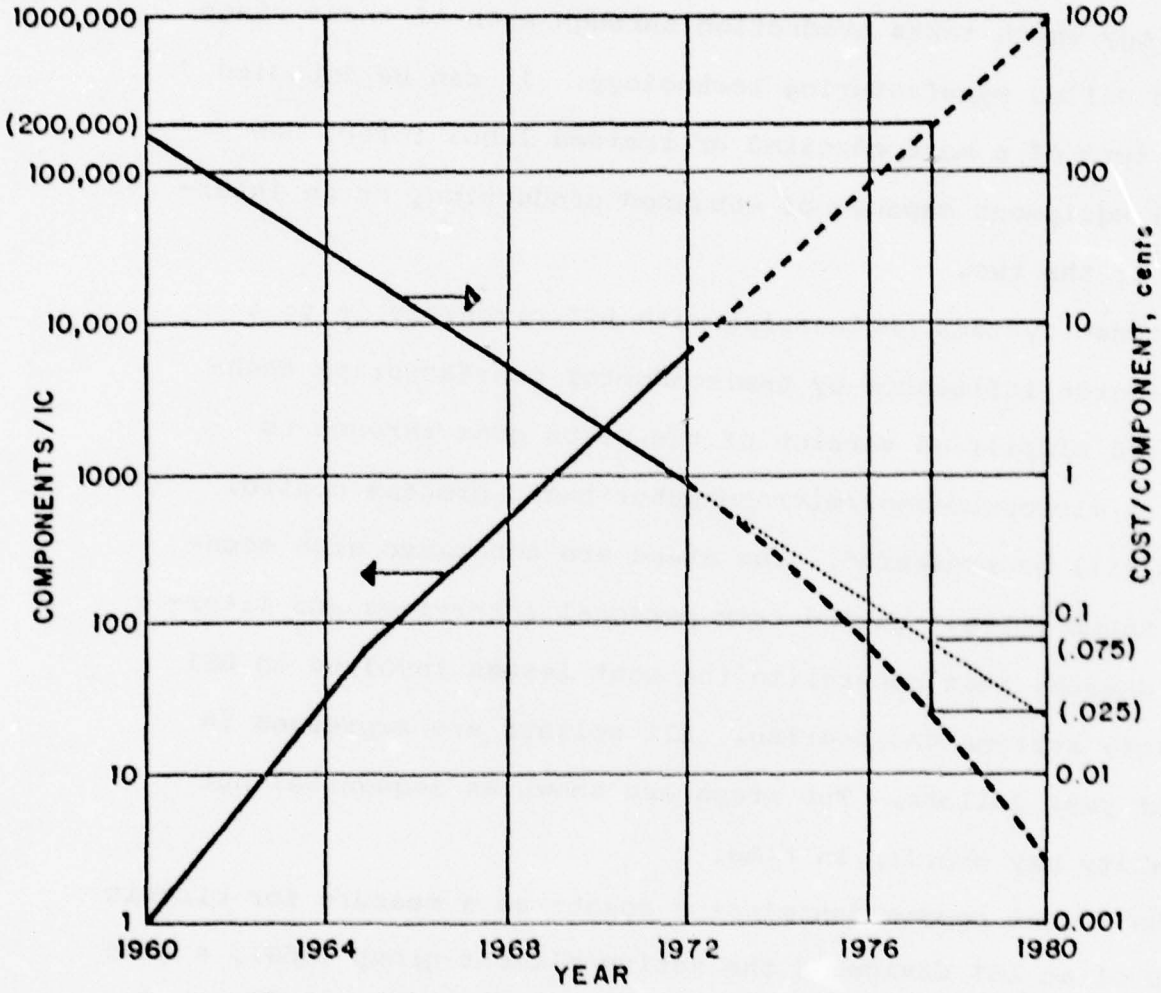
A. HARDWARE

A key to the economic aspects of systems engineering with LSI circuitry based systems is in the associated manufacturing technology. Manufacturing technology is the means by which

something is fabricated by the producer. Typically any product will go through several stages of processing from raw material to finished system with value added at each stage. It is the technology which takes production through each of these stages that is called manufacturing technology. It can be embodied in the form of a more educated or trained labor force, new capital equipment capable of enhanced production, or an interaction of the two.

Because systems engineering with LSI circuitry is to a great degree influenced by semiconductor manufacturing technology, a simplified version of the steps gone through to create a microprocessor/microcomputer based process control system will be presented. The steps are annotated with economic "thumb rules," gained from personal interviews and literature search, that generalize the cost issues involved in LSI circuitry systems engineering. All dollars are expressed in current year dollars. The steps are shown as sequential but in reality may overlap in time.

The lowest common denominator sought as a measure for circuit design of an LSI device is the active element group (AEG); a gate for logic units, a bit for memory units. Measures of complexity and recurring device manufacturing cost are made parametrically on AEG counts. Figure A-1 is a composite overview of the trends in device complexity and recurring manufacturing cost in the last several years for the silicon metal oxide semiconductor device technology. The description of the steps that follow will develop how Figure A-1 is constructed.



Source: Hittinger, William C.
"Metal-Oxide-Semiconductor Technology"
Scientific American, August 1973,
Vol. 229, No. 2, page 28

TRENDS IN LSI COMPLEXITY AND COST
Figure A-1

The first step in the LSI manufacturing process however is not included in the computations to obtain Figure . This first step is the creation of the circuit logic design and accounts for the major share of the non-recurring costs of production. In the case of an original design which works significantly into a semiconductor manufacturer's corporate strategy (like the Intel 8080, Motorola 6800, and Texas Instruments' 9900), the cost of this creative, labor intensive activity is generally in the millions of dollars, and one to two years of calendar time. For the 8080, 6800, and 9900, the figure was probably around $5,000,000 each. A reasonable, although large, range for this non-recurring cost is from $100,000 to the few millions. The large variance comes from a number of factors ranging from corporate financial structure, and marketing, to existing design tools. Separating out this latter factor as one which could be applicable industry wide, reveals two significant developments that have the potential of reducing this non-recurring cost. These are computer-aided-design, and the use of general circuit design arrays that are customized in the final masking and diffusion steps (to be explained more fully later). Some companies have reported out in the trade magazines reduction in calendar time for development of a logic design to 2-3 months and a reduction in non-recurring cost of design into the tens of thousands of dollars from the hundreds of thousands of dollars. (A question which remains unanswered but will be addressed later is if the non-recurring cost of the design of

the testing procedures for these custom LSI chips has been included in this reduced non-recurring cost.) Being the first step in the process and particularly if tied heavily into the corporation's strategic planning, working out a logic design which later proves faulty, which has problems in production and testing, and which has missed the market, can quickly put a source of LSI circuits into financial straits. To a high level military budget planner, thinking in terms of hundreds of thousands of dollars or even a few millions of dollars may not be significant. We are numbed by large numbers almost daily. However, even a casual reading of a magazine like Business Week gives one an appreciation for how in a fiercely competitive business like semiconductors, risking and losing from a few hundred thousand dollars into the few millions of dollars and the concurrent few months of time can set a corporate strategy on its head and leave companies as big as Texas Instruments trying to overcome the effects (9-6).

After the logic design has been worked out, the next step is drafting and documenting the design. This generally takes from 2-3 man-months. If computer-aided-design was used, this step can be a fallout of the actual logic design process.

The next part of the process entails growing a very pure sausage-shaped silicon ingot.

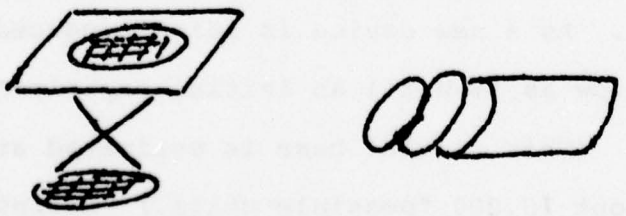
The silicon raw material is a miniscule part of the cost. Other materials used are silicon-on-sapphire (SOS) (obviously more expensive in terms of raw material), and most recently the use of GR-AS, allium-Arsenide.

The ingot diameter is a crucial number. The diameter has grown from 1-2 inches to 3-4 inches, with 5-6 inches being the leading edge of the technology. Beyond 6 inches, while possible, creates warping problems when the ingot is sliced into wafers. The crucialness of increasing the ingot diameter is the multiplicative effect it has on the number of chips that can be created in one process set up. For example, the ratio of surface area on a 4-inch diameter ingot to one with a 3-inch diameter is simply

$$\frac{S_4}{S_3} = \frac{\Pi (\frac{4}{2})^2}{\Pi (\frac{3}{2})^2} = \frac{16}{9} ,$$

or roughly 1.8 times the number of possible chips. Note however that this ratio decreases as the ingot grows successively larger, another reason why expanding incrementally, by heavy capital investment, beyond a 6-inch ingot is considered very leading edge technology and very unlikely in the future.

The next manufacturing phase is to slice the ingot into wafers, each about 0.03 inches thick.



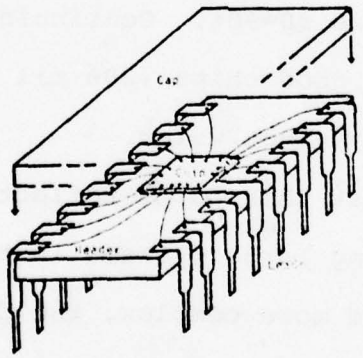
The wafers are then sent through a masking and diffusion process anywhere from three to ten times. This process entails

miniaturizing and replicating the circuit design on the wafer. A photolithographic technique is used for masking. As miniaturization beyond optical resolution levels occurs, an electron beam technique is used. Masking basically lays out the negative of the circuit pattern, and diffusion "etches" the pattern onto the surface of the wafer. (The generalized circuit array process mentioned earlier etches out several clusters of common circuit elements and these wafers are put "on-the-shelf." A final masking and diffusion will connect up only those elements desired for the "customized" circuit design when a user orders it up. This customized design however must still undergo testing procedures developed for it and it alone.)

The first, and most significant of three test points now occurs. The etched wafer is inserted into a test probe where some very simple tests are used to determine bad chips. The industry average wafer yield is only about 30-40%. For example, if a 4-inch diameter ingot is used and a 200 mil x 200 mil (0.2 inch x 0.2 inch) chip is desired, the maximum number of chips per wafer that can be etched is a few less than 300. Hence, only about 100 are acceptable after this check point. Furthermore, that 30-40% wafer yield is the industry average. As a new device is being produced, wafer yield may be as low as 5% until an initial experience base has accumulated. (This initial base is estimated at 32 wafer runs or about 10,000 "possible" chips.) Learning occurs and the yield will eventually reach the 30% average, and for

some designs and in some companies, wafer yield may reach as high as 40-50% for a sustained run of wafers. However, for some logic designs or for some companies the industry average of 30% wafer yield might never be reached, particularly if the production run is not sustained over time.

After going through the test probe and marking of the bad chips, the wafer is scribed, i.e., the wafer is cut up into individual chips. The bad chips are discarded and the good chips are sent to be packaged. Three methods are currently used to package the chips. The oldest method, the flat pack, has been largely replaced by the most prominent method, the dual-in-line (DIP) package. The third method, the chip carrier, is the leading edge packaging technology and will be discussed shortly. In the DIP, the chip is placed on a lead frame by a worker using a special machine and a microscope.



The leads are bonded by pressure to the chip and to the pins. This is generally a labor intensive process and the cost of DIP packaging is considered as an increasing function of the number of pins (leads off the chip), with 40 pins being the break point where the cost increase begins to become more severe. (This aspect is beginning to get more attention in

systems engineering with LSI chips.) Most semiconductor companies will send the chips overseas to be packaged and assembled into a product. The mounted chip is inspected for good connections. The industry average yield of good mountings is about 80-90% of the number of chips that pass wafer yield. This yield is called the packaging yield. Returning to the example, that means 80-90 good chips (200 mil x 200 mil) from the 4-inch wafer at this point. A cap of ceramic or plastic is installed on these good packages. For high reliability uses, the cap is hermetically sealed. The hermetic seal is the primary reason given for the more expensive price, by a factor of 2-3 over commercial chips, of military temperature spec DIP's. For other uses the cap is molded on. The capped package is extensively tested at this point by automatic testing equipment. The ratio of good chips to the total in this final test yield is 80-90%. Continuing the example, that means about 64-81 good chips (200 mil x 200 mil) from a possible of 300 from a 4-inch wafer.

The reader should at this point reflect on the philosophy for designing large systems in this report. As chips become larger and more complex, the design of this final test can approach that of the logic design of the chip itself, i.e., this non-recurring cost of final test design can be on the order of tens of thousands of dollars, if not into the hundreds of thousands of dollars. Even if computer-aided-design and final mask and diffusion customizing can reduce the non-recurring logic design cost, a non-recurring final

test design cost must still be incurred for every peculiar logic design conceived of. Computer generated tests can aid this process. An obvious cost advantage could be had by the user if this large test design non-recurring cost is spread over as many other users as possible.

Because of the labor intensiveness of packaging, and its cost as an increasing function of pin count, some forms of packaging are being developed, other than the DIP, which lend themselves to easier productizing (interconnecting) from a chip set. One which appears in a couple of different forms is called the chip carrier. Using a completely automated procedure, the LSI chip is suspended and encapsulated in a pinless package. The leads are bonded to the chip and joined to only a solder point on the edge of the carrier.

Up to this point what has been discussed is the manufacturing process of logic design of packaged chips for a microprocessor/microcomputer device. In summary, the non-recurring manufacturing cost of logic design is from the tens of thousands of dollars into the few millions of dollars. The same magnitudes are true for the non-recurring cost of logic design testing procedures. The recurring costs of packaged chip manufacturing have followed the trend in Figure (A-1) and are currently between about .01-.1¢ per active element group. That means that for a microprocessor chip on the level of complexity of the Intel 8080A, about 4,000 gates per chip, the recurring cost of manufacturing for a sustained run is between about $.40 to $4.00 per chip with a selling price around $15-20.00. A military temperature spec chip based on a sampling of price

lists is about 2-3 times that price. A chip of the complexity of the 16 bit word length TI 9940, a recently announced single chip microcomputer of about 10,000 gates, including 1-2,000 memory bits and I/O points, would have a recurring manufacturing cost of up to about $10.00 per chip. However, the selling price, as initially guessed by competitors, is at about $500.00 for the TI 9940 because Texas Instruments will still be amortizing the non-recurring costs and still be coming down the experience curve.

The area to be addressed now is the process of going from an LSI chip set to a finished microcomputer product. This process is best conceived of as involving three stages, interconnecting the packaged chips to a board, interconnecting the boards to accommodate for power, signal, and ground, and finally, interconnecting boards into a product case like an ARINC Air Transport Rack (ATR) or a Naval Standard Electronics Module (SEM2A). There is a wide variety of methods to accomplish each of these steps and it is probably fair to say that these steps are not as amenable to as rapid a pace of technological change in manufacturing technology as is the fabrication of the packaged LSI chip. Again, it is volume that will determine the amount of automation that will exist in the manufacturing process. Systems engineering trade-offs are made between performance, and reliability/maintainability which impact the manufacturing technology of productizing from chip sets, rather than the manufacturing technology impacting the systems engineering as with the chip making process.

The non-recurring cost of product design from chip sets is on the order of thousands of dollars into the few hundreds of thousands of dollars but again, possibly into the few millions of dollars for some corporate strategy intertwined items like the first programmable calculators or militarized processors and computers like the AYK30, LSI11M, PDP11/34M, AYK14, and ROLM products.

The parameter for recurring cost estimation and also for reliability/maintainability measurements when productizing (interconnecting) an existing LSI chip set is the number of connections or contacts that have to be made between chips and supporting active element groups like power, ground, and signal connections within the product. Although the contributing failure mechanisms of LSI chips to the product are being explored, it is the interconnects that are most significant in the failures of a product and that may reflect the reliability overkill of the LSI chip.

These recurring costs for a sustained production run range from 5¢ to 10¢ per connection and slightly more depending on the interconnect method used, production volume, and automation level of manufacturing. A few of these interconnect methods will be described. Following this will be a description of three techniques that go directly from chip carriers (vice packaged chips) to product. When projecting to a VSTOL program with the concept definition and validation occurring in about the 1981-1985 time period, it will probably be one of these three techniques that will end up as the interconnect technique.

Interconnection of chips in computers in the early 1970's was accomplished with point-to-point wiring of chips mounted on molded plastic blocks throughout the system. This is labor intensive but provides the ultimate in flexibility and would be now used primarily for breadboarding of prototypes or low volume products that don't justify the capital expense of automated fabrication machinery. More common today is the use of printed circuit boards (PCB's), which have solder runs to provide the means of current conduction. The DIP's are pressed manually or automatically into female connectors aligned on the PCB. These boards, called daughter boards, still must be connected into a backplane, called a mother board as in Figure (A-2) from reference (A-1).

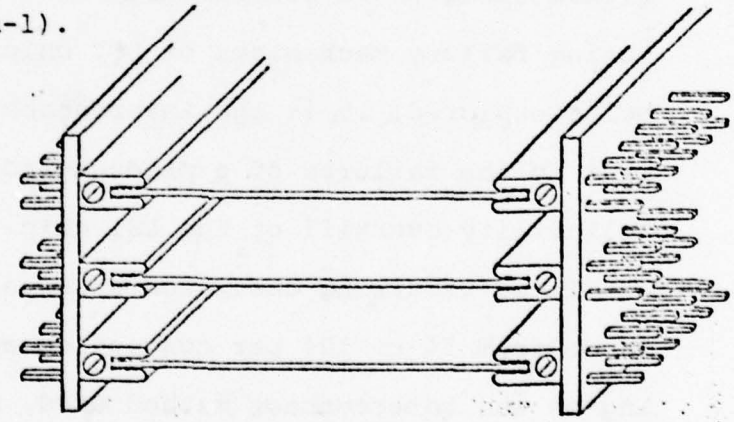


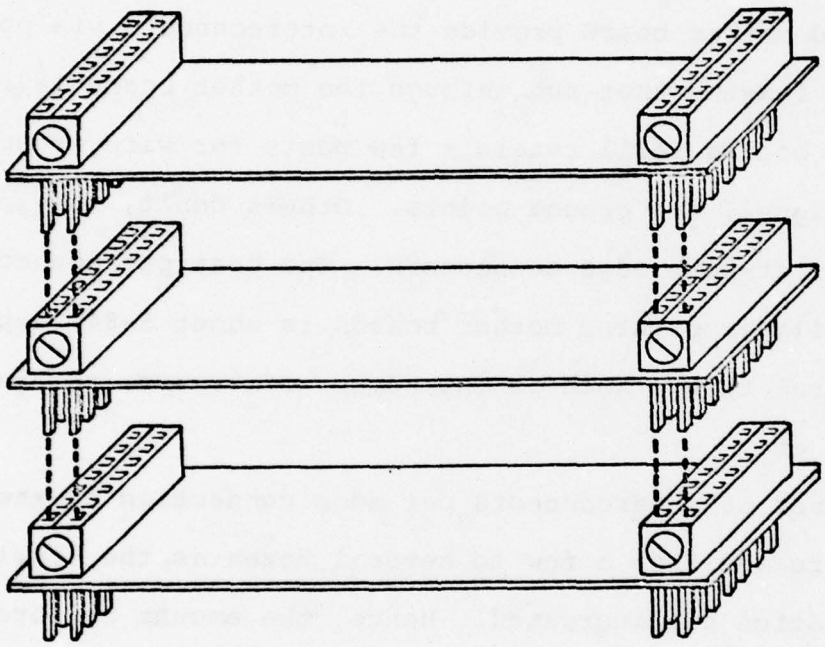
Figure A-2

Originally, the mother boards were metal backplanes and had rows of female post connectors into which male edge connectors on the daughter boards were plugged. Posts on the mother board served as leads into which hand wire-wrapping (more than 10¢ per connection) or automated wire wrapping (about 10¢ per connection) were accomplished on the reverse side of the backplane. The wire wrapping provides flexibility in that the

board can be rewrapped if a daughter board is updated. In very large quantity designs where flexibility is not so important, a multilayer printed circuit epoxy-glass or fabric mother board is now used. The printed circuit daughter boards, often two-sided, are pressed into the edge connectors on the mother board, also often two-sided. The printed circuits on the multilayered mother board provide the interconnects via posts of selected lengths that run through the mother board layers. Some mother boards still retain a few posts for wire wrapping of power, signal, and ground points. Others don't, relying instead on flat-wire edge connectors. The cost per connection on the multilayer printed mother boards is about 5-8¢, depending on several things such as the number of layers, remaining wire wraps, etc.

The number of interconnects per edge connection on the PCB has increased from a few to several dozen as the level of chip integration has increased. Hence, the amount of force required to insert a daughter board into a mother board has increased considerably. This has led to Zero Insertion Force (ZIF) connectors. There are several ZIF designs. They require no force for insertion because all the female connectors are initially free of resistance. Once the PCB board is in place, then a mechanical action like a lever or machine screw is actuated to close all the female contacts simultaneously.

The latest form of the ZIF connector is to dispense with the mother board idea all together and stack two-sided daughter boards via ZIF connectors that themselves act as active electronic elements (Figure A-3).



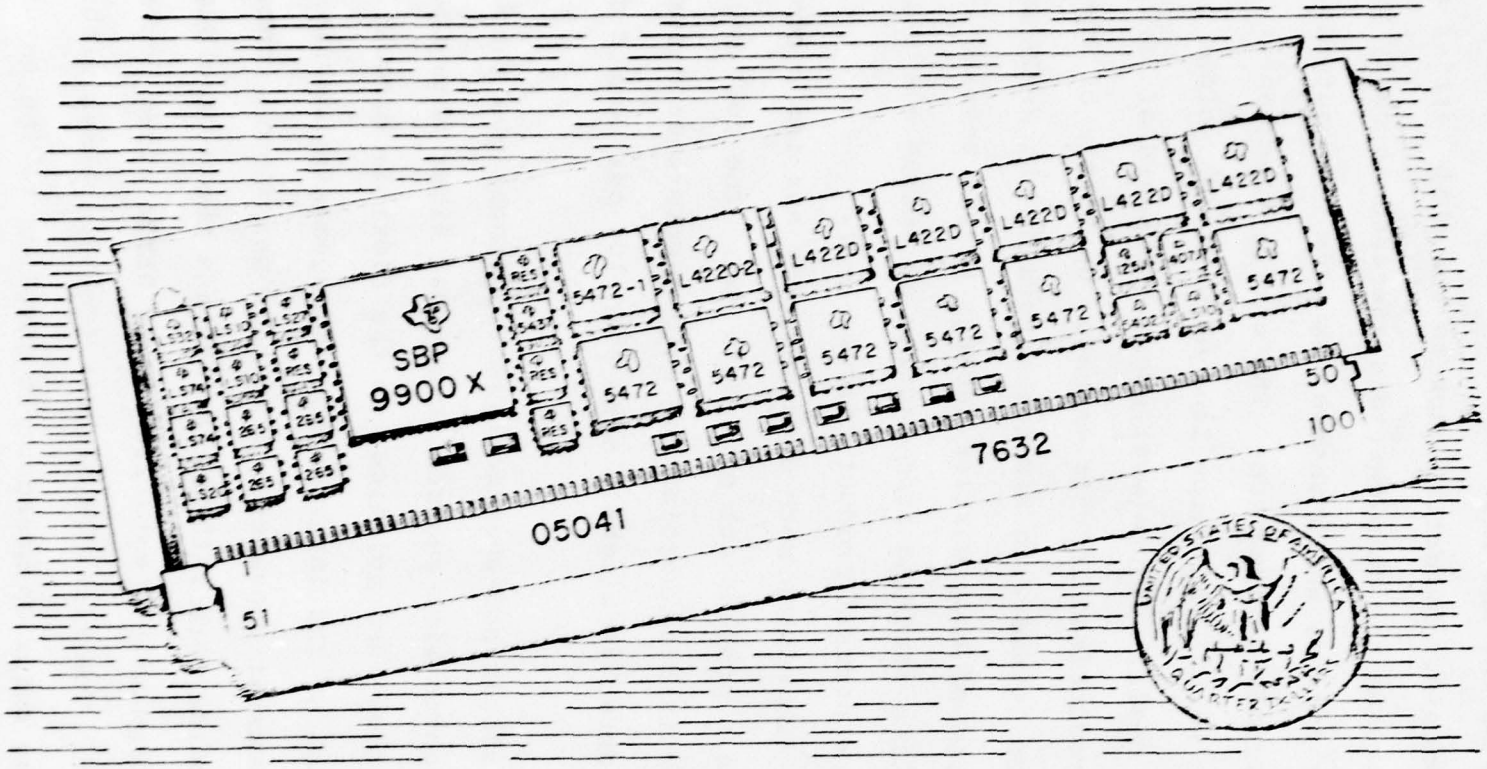
ZIF CONNECTORS
Figure A-3

The trend, as can be seen by the ZIF interconnect technology, is toward making the interconnects themselves active electronic elements. Referring back to the chip carrier concept, in some cases the chip carriers are automatically placed on cold solder points, which themselves have been automatically placed on a ceramic layered PCB. The entire arrangement is refluxed in a kiln. The chips line up with the solder points as the solder cools. Figure(A-4) shows one board of SEM2A size module productized in this manner. Up to six of these boards would be joined with ZIF connectors in the SEM2A module.

Another type of chip carrier package stacks several carriers together with the stacking frame acting as active electronic elements in the overall product (Figure A-5).

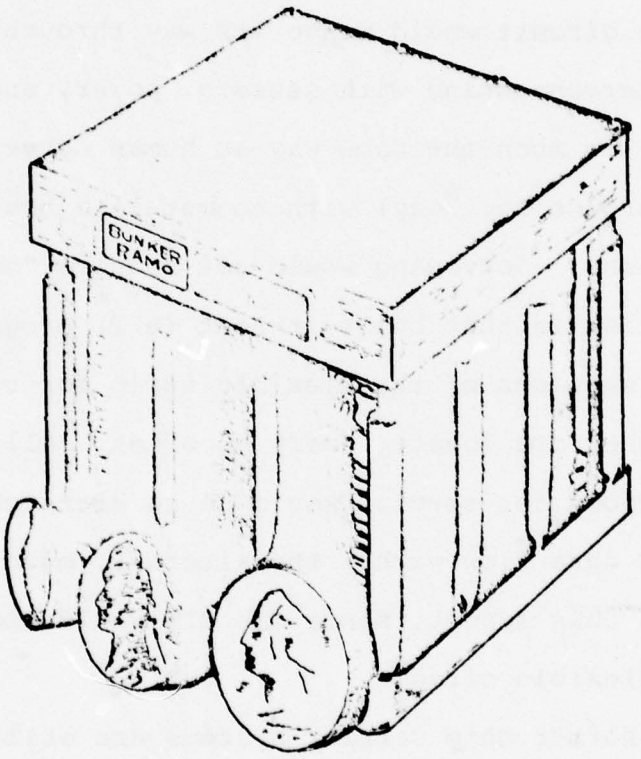
These are two of the more advanced chip carrier concepts used to create a product out of a chip set. The cost per connection for interconnecting the chip carriers as described above is projected at slightly more than 10¢ per connection in volume.

Another, albeit somewhat more exotic, technique which has particular meaning with regards to optical fiber data transmission technology and avionics, is the flexible circuit. Up until now, each of the interconnect arrangements described was still somewhat conventional in that chips were connected to boards, boards were interconnected, and these were placed into a card cage with a metallic ATR or SEM2A sized box and a control panel. The flexible circuit idea dispenses with the idea of boxing chips on boards. It involves taking several



SEM 2 A SIZED REFLUX SOLDER BOARD

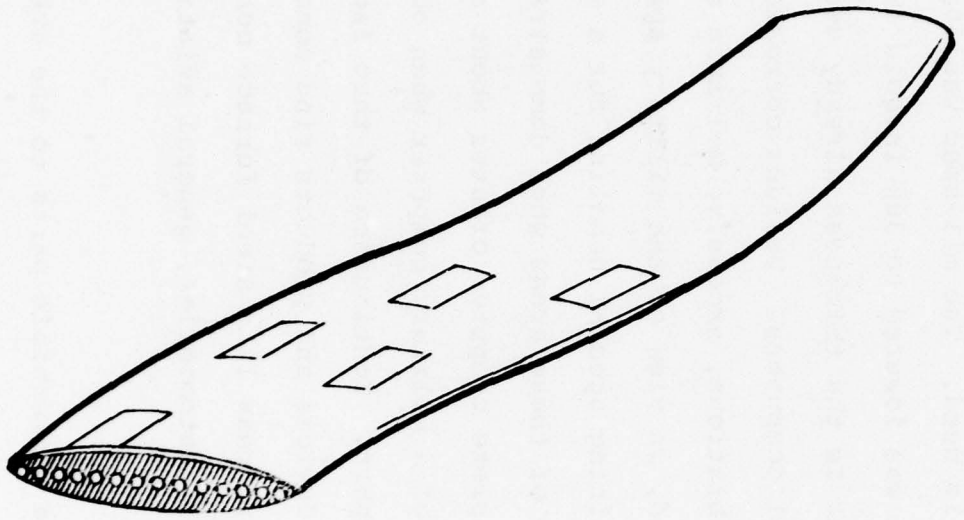
Figure A - 4



STACKED CHIPS CARRIERS
Figure A - 5

chip packages, or as is more likely by 1985, several chip carriers, each with its own convection cooling vanes, and interconnecting them to a flat, flexible bundle of wire, or a flat fiber optic bundle (Figure A-6). (The arrangement would look very much like the sugar drop candies affixed to paper tape, popular with children a generation or two ago.) The flexible circuit would weave its way through the aircraft airframe interconnecting with sensors, power, and ground as appropriate (in much the same way as human nerve fibers weave their way through the body) with no metallic housing (ATR or SEM2A) required. Servicing would not require "neurosurgery" level expertise in that built-in-test (BIT) programs could isolate out sections of the flexible strip for replacement via snap connectors located every so often. All that would have to be known for service would be an hierarchical level model of the data flow within the aircraft, much like the one described in this report, since the BIT would isolate to a section of flexible circuit.

The two former chip carrier systems are still slightly above the 10¢ per connection figure. The latter flexible circuit idea is only in the conceptual stage and no cost information is available. The key to the ability of all three of these advanced techniques to be able to compete cost-wise with existing interconnect techniques is in whether or not demand for them will be of such a volume that their automated manufacturing technology investment is justified. Based on the military computer system numbers generated in this report, it



CHIP CARRIERS ON FLEXIBLE CIRCUIT

Figure A - 6

is doubtful whether the military alone could foster the necessary volume. Rather, as postulated throughout this report, the military will have to get in on the commercial market.

A good example of how the military spec is already adjusting to commercial market forces is the coating used on connections to combat corrosion on interconnect surfaces. Gold plate over nickel is used. The mil-spec was 50μ in gold over 100μ in nickel but was lowered to 30μ in gold over 50μ in nickel. The latter is the thickness already used in certain types of commercial computers. Besides corrosion, temperature extremes, large vibrations, excessive moisture and radiation are also considered, in view of the military specification structure, as requiring special handling for a military computer. The totality of these areas when demonstrated in a mil spec device or complete computer creates about a factor of 2-3 more in the price of a military computer when compared to its commercial counterpart. Continuance of that factor is questionable as basic devices and products find commercial use in high reliability areas like steel furnace control, nuclear power plant control, automobiles, general aviation, etc.

B. SOFTWARE

There are three discernible parts to the avionics software cost issue, and the software cost-estimating problem. The first part is the cost associated with the applications software creation. In avionics this is the Operational Flight Program (OFP). The second part is the software development

and support tools used for creation of the applications software. These are the compilers, debuggers, editors, etc. The third part is the aircraft systems simulators, the mock ups which are used to simulate an aircraft in flight, to test out the OFP. To place the software cost issue in perspective, interviews with industry indicate that as much as 95% of software cost is in the development and support tools and systems simulators. Literature search indicates that cost estimating for the estimated remaining 5% is more structured with regards to parametric, or top down, techniques and established data bases. The cost estimating of development and support tools and simulator systems is left to the engineering, or bottom up method, and is dependent on the decision at hand, the actual computer system under review, and the systems already in place.

The systems simulators are not considered as a differential cost element in this report as they are created on parametric characteristics of the aircraft and the actual computer architecture of the avionics system is transparent to them. The differential cost elements with respect to the avionics computer system architecture are the software development and support systems and the applications software creation. These two cost elements are important to the economic analysis of this report because the worth of a standard computer family is most often argued for in light of the savings generated by software commonality across several applications. The software costing in this report will be framed around exploring the

validity of that argument in light of possible rates of growth of use of a standard computer family, possible rates of growth of technology embodied in computing devices in the commercial world, and the effects on aircraft airframe cost of the avionics computing system.

Of the two differential software cost areas, OFP creation, and development and support tools, generalization of the cost issues and methods of the former will be addressed first (a costing method used in contracting and a parametric technique will be described from the literature). The latter will then be addressed based on trends discerned by this author from the literature.

Aircraft contracts with hardware and software procurement bundled together cost the applications software development as a percentage of the airframe engineering full scale development man-hours. The percentage is a negotiated figure based on the corporate history of both the buyer and seller and may or may not include the cost of the development and support tools, and systems simulators. Because corporate costing history is proprietary and because this report is meant to generate some independent estimates, a parametric technique was used, best described by reference (A2). This technique will be paraphrased for the reader. In the process of doing so it will be highlighted with some additional current thoughts from the literature on the applications software life cycle and management. An annotated bibliography is also included at the end of this report for the reader specifically interested in this area.

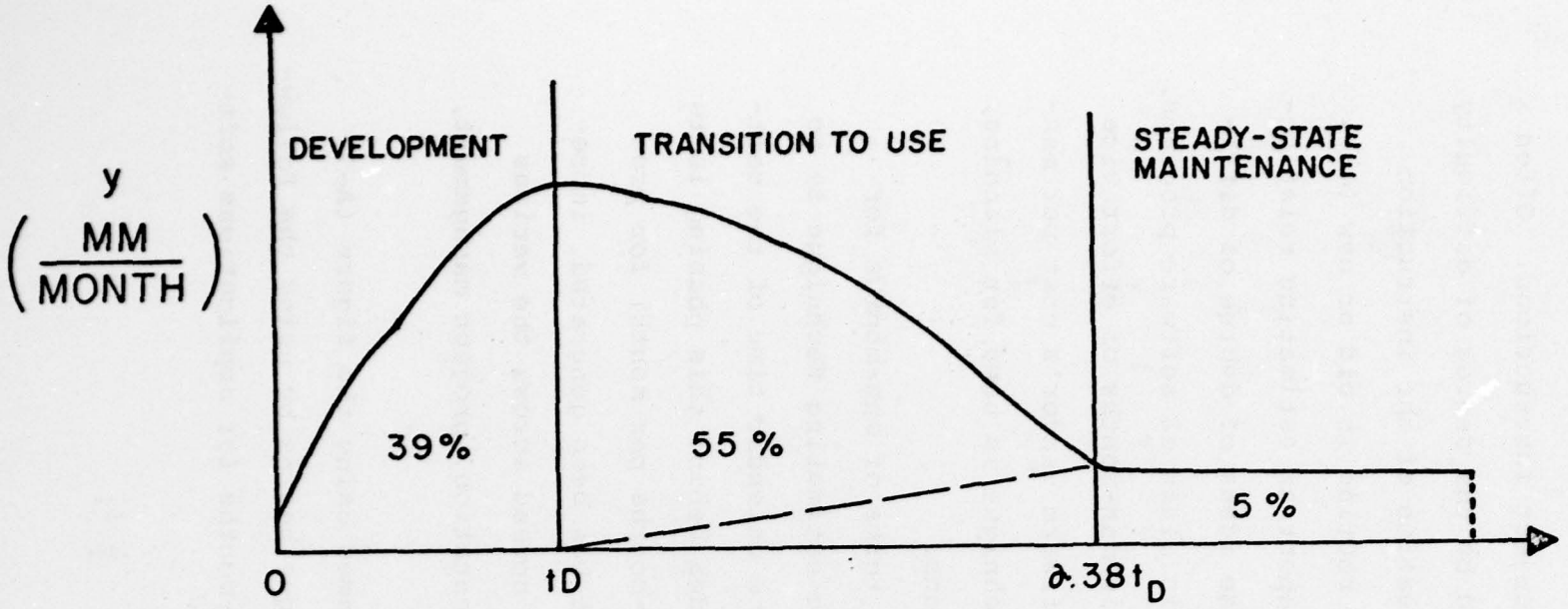
The parametric technique of reference (A2) is based, as most are, on the parameter of software program size as measured by number of source or object instructions. Often the instruction count is weighted by the degree of difficulty or complexity inherent in the creation of the instruction type and whether the programming routine is old or new (A3). In the technique used for this report an estimating relationship coefficient will embody these ideas of degree of difficulty and old or new for a certain class of software programs.

Cost is generally expressed in man-months of effort vice dollars so that an individual software vendor's cost per man-month can be applied when the technique is used for planning, budgeting, or contract negotiations.

After sizing the software in terms of man-months for development, the next step in the estimating technique is to time phase the man-months over the calendar time of the software development project. The idea behind this phasing is to generate a man-loading curve (man-months per month) for project management. Once this curve has been generated, increments of man-loading resource are spread across the various project activities such as documentation, project management, programming, etc.

The entire process can be framed using the figure (A-7) from reference (A2). The technique begins by using the following relationship to estimate man-months for applications software development effort.

$$MM = C(I_{o,s})^{P} \cdot \prod_{i} f_i$$



$$y = \left(\frac{K}{t_D^2}\right) t\, e^{-t/2t_D^2}$$

$$K = MM/0.39$$

$$MM = C\,(I_{o,s})^{P}\,\pi_i\, f_i$$

$$t_D = I_o/(99.25 + 2.33\, I_o^{0.667})$$

SOFTWARE LIFE CYCLE MANLOADING
Figure A-7

where

MM - man-months of development effort

C - a coefficient estimated from a data base of like projects

$I_{o,s}$ - the count of source or object instructions in the applications software

P - a power estimated from the data base of like projects

f_i - multiplicative factors that come from a stratification of the data into groups such as; developed on host computer vs. target computer, developed on a time share system vs. a batch system, etc.

As seen from figure(A-7), development is only part of the applications software life cycle. It has come to be sub-divided itself into three phases; analysis and design, code and debug, and module and systems test and integration. Literature search and interviews support the conventional wisdom that the split of effort between these three phases of development, i.e., the man-loading split, is 40%, 20%, 40% respectively (A4).

Once the number MM has been estimated, the technique takes the following quotient to determine the total man-months of effort in the applications software life cycle, i.e., the total area under the curve of figure (A-7).

$$K = \frac{MM}{0.39}$$

K - total man-months of effort for life cycle of applications program

The technique then describes the curve by

$$Y = \left(\frac{K}{t_D^2}\right) t\, e^{-t^2/2t_D^2}$$

where

$$t_D = I_o / (99.25+2.33Io^{0.667})$$

Y - man-months/month

t_D - natural development time for an applications program of I_o object instructions

The major point of the technique is that (K/t_D^2) is a constant based on instruction count, and that it is not possible to compress the natural development time by adding man-months of effort. Rather it is hypothesized, supported by independent sources (A5,A6,A7), that there exists a natural time and man-loading curve for a particular sized program. In the words of Brooks in his classic essay "The Mythical Man-Month", adding man-months of effort to an applications software development project which is off schedule makes it later, particularly if as pointed out in (A8), it is added late and tentatively. Preliminary work by others shows that the distribution of K, the total man-months of effort, by a 39%K to development, 55%K to transition to the user, and 5%K in steady state maintenance is reasonable within a few percentage points (A9).

Reasons for the shape of the man-loading curve in figure (A-7)can be found in another reference (A10) which reports that the error profile of an applications program can be characterized by the saw-tooth of figure (A-8).

The cause of this saw-tooth effect is intuitively pleasing and plausible. When a user begins use of the applications software during transition, it is put through a wider range of

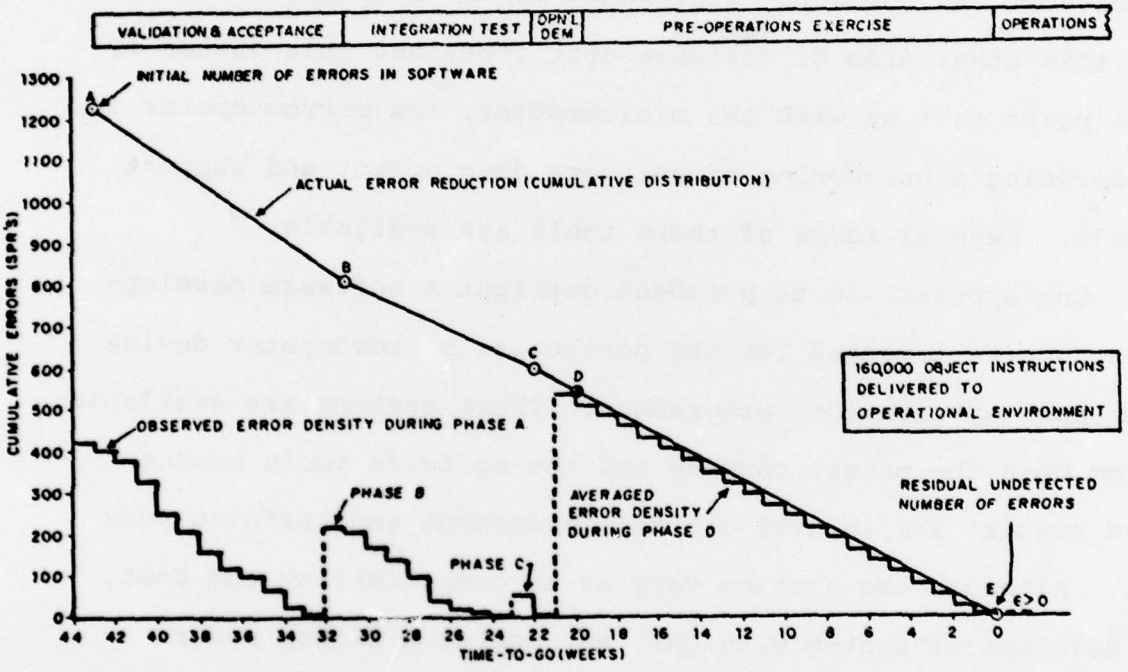


Figure A-8. Error Reduction Process Showing Discontinuities at Each New Phase.

data, on the target machine, and possibly a more taxing environment than that of the development lab. Bugs are discovered and worked out that weren't discovered in the more benign environment of the development lab.

This technique of reference (A2), paraphrased above, will be used to cost the navigation and ballistics portion of the VSTOL OFP analyzed in the first part of this report. It will also be used to address the issue of development and support tool cost and software maintenance cost. A literature search in this other area of software cost impressed this author of the point that as with the minicomputer, the microcomputer is undergoing a burgeoning of software development and support tools. Several forms of these tools are available.

One approach is to purchase outright a software development system targeted for the particular microcomputer device for each applications programmer. These systems are available from both the parent company and the software tools houses and require the initial capital investment and training work up. Although the systems vary as to compatibility and cost, a development system with CRT, key board, a higher order language (HOL) like FORTRAN, PLM, or PASCAL, a set of programming aids, and a read only memory (ROM) programmer can be had in the $25,000 range.

Another approach is to lease time from the increasing number of microcomputing development commercial time share systems capable of handling all of the various device types. These firms make the capital investment in the development

and support system tools and then lease the time, a process modeled by the same $S \cdot N \geq CNR + CR \cdot N$ inequality of the first part of this report. A current quoted figure is $1,000 per month for lease of one terminal that provides any one of several HOL's for any one of several microprocessor/microcomputer devices. The terminal would have a CRT, and a basic set of programming aids.

A third approach is to buy outright a development system capable of handling several different microprocessor/microcomputer devices. Although no quotes were available, these systems would start at $100,000.

A final approach considered is to create the development and support tools for the target microcomputing devices and the input/output structure for each application to run on a main frame host machine. This would be in the form of compilers, code generators, assemblers, debuggers, linkers, loaders, etc. The data generated by the Computer Family Architecture report will be used in this report in an order of magnitude way to address the costing issues of this method of supporting software with respect to the others. As an example of the magnitudes involved in this method, the CFA report estimated about $4.9M for a CMS2 compiler for the CFA candidate machine.

XI. REFERENCES

1. ALLEN, F.E.; and COCKE, J. A Program Flow Analysis Procedure. COMM. ACM 19, 3 (March 1976), 137-147.

2. BREUER, M.A., Editor. DESIGN AUTOMATION OF DIGITAL SYSTEMS: THEORY AND TECHNIQUES, Prentice Hall, 1972.

3. BURR, W.E.; COLEMAN, A.H.; SMITH, W.R. COMPUTER FAMILY ARCHITECTURE SELECTION COMMITTEE FINAL REPORT;T.R. ECOM-4525 SUMMARY. U.S. Army Electronics Command, Fort Monmouth, N.J. 07703. Sept 1977.

4. CONSOLVER, G. et al. DISTRIBUTED PROCESSING/MEMORY ARCHITECTURES DESIGN PROGRAM. AFAL-TR-75-80, AIR FORCE AVIONICS LABORATORY,Wright-Patterson Air Force Base, Ohio 45433, Feb. 1975.

5. DORN, W.S.; McCracken, D.D. NUMERICAL METHODS AND FORTRAN PROGRAMMING, Wiley and Sons, 1964.

6. ERTELSCHWEIGER, J.T. Verification and Feasibility Study of a Microcomputer Based Ballistics Algorithm. Masters Thesis, Naval Postgraduate School, Dec. 1976.

7. FORD, L.R. Jr; and FULKERSON, D.R. FLOWS IN NETWORKS. Princeton, N.J., Princeton University Press, 1962.

8. FREEMAN, H.A.; CHRISTIANSEN B.P. AVIONICS INFORMATION PROCESSING SYSTEM DESIGN METHODOLOGY, NADC-76026-20 Naval Air Development Center (code 207), Warminster, Pa. 18974, March 1977.

9. GRIFFITH, V.V. et al. AIRCRAFT AVIONICS TRADE-OFF STUDY. ASD/XRHA, Wright-Patterson AFB, Ohio 45433, Nov. 1973.

10. HANTLER, S.L.; and KING, J.C. An Introduction to Proving Correctness of Programs. COMP. SURV. 8, 3 (Sept. 1976), 331-353.

11. HAYNES, J. INTEL SBC 80/10 Single Board Computer Reliability Report. RR-17, INTEL 3065 Bowers Ave, Santa Clara Ca 95051, June 1977.

12. JOHNSON, D.E.; JOHNSON, J.R. GRAPH THEORY WITH ENGINEERING APPLICATIONS. New York, N.Y.: The Ronald Press Co. 1972.

13. JOHNSON, C.E.; KILPATRICK, P.S. et al. ALL SEMICONDUCTOR DISTRIBUTED AEROSPACE PROCESSOR/MEMORY STUDY. AFAL TR-73-226, AIR FORCE AVIONICS LABORATORY, Wright-Patterson AFB, Ohio 45433, Aug 1973.

14. JUPIN, H.A.; The Ballistics Processor of a Multiple Processor Airborne Tactical System, MS Thesis, Naval Postgraduate School, June 1975.

15. KODRES, U.R. Discrete Systems and Flowcharts. To appear in IEEE Transactions on Software Engineering Nov. 1978.

16. KOENIG, H.E.; TOKAD, Y.; KESAVAN, H.K. ANALYSIS OF DISCRETE SYSTEMS. New York, N.Y.: Mcgraw-Hill Book Co. 1967.

17. MCCABE, T.J. A Complexity Measure. IEEE Transactions on Software Engineering, vol. SE-2, No. 4, pp. 308-320, Dec. 1976.

18. NAVAIR, INTEGRATED WEAPONS SYSTEMS THEORY, MAINTENANCE INSTRUCTIONS, NAVAIR01-85ADF-2-10.11, Sept 1971.

19. NAVAIR, INTEGRATED WEAPONS SYSTEM THEORY, ORGANISATIONAL MAINTENANCE INSTRUCTIONS. NAVAIR 01-85ADF-2-10.1. Sept. 1971.

21. SHNEIDERMAN, B.; MAYER, R.; MCKAY, J. and HELLER, P. Experimental Investigation of the Utility of Detailed Flowcharts in Programming. COMM. ACM 20, 6 (June 1977) 373-381.

22. SUTHERLAND, I. E.; MEAD, C.A. Microelectronics and Computer Science. Scientific American, vol 237, No 3, Sept. 1977.

REFERENCES

9-1 Cooper, J. D., "Computers in Tactical Systems," Computers in the Navy, J. Prokop, Ed., Naval Institute Press, Annapolis, MD, 1976, pp 122-123.

9-2 McCarter, A., "Electronic Systems Viewpoint of the Automobile Environment," Proceedings of the Eleventh Electrical Insulation Conference, 30 September-4 October 1975, paper #21E.

9-3 Jones, C. R., et al, "Life Cycle Costing of an Emerging Technology: The Fiber Optics Case," Naval Postgraduate School, Monterey, CA, March 1976. (NPS-55Js76031).

9-4 "Teaming up to build military minicomputers," Business Week, December 20, 1976, pp 40-41.

9-5 Preston, G. W., "Large Scale Integrated Circuits for Military Applications," Institute for Defense Analysis, Arlington, VA, May 1977. (IDA paper P-1244).

9-6 "New Leaders in Semiconductors," Business Week, March 1, 1976, 40-46.

9-7 Nunn, M. E., "Navy RDT&E Program in Advanced Testing Technology," Tri-Service Briefing on Automatic Testing, 15-16 June 1977, p 228.

9-8 "Booming Micro Markets," Mini-Micro Systems, October 1976, pp 18-19.

A1 Plourde, R. A., "The PCB Connection," Electronic Packaging and Production," March 1977, pp 28-33.

A2 Bourden, G. A., "A Computerized Model for Estimating Software Life Cycle Costs," HQ Electronics Systems Division USAF, Bedford, MA, 6 July 1977. (Unpublished paper.)

A3 Wolverton, R. W., "The Cost of Developing Large-Scale Software," IEEE Transactions on Computers, June 1974, pp 615-636.

A4 Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp 48-50.

A5 Putnam, L. H., "A General Solution to the Software Sizing and Estimation Problem," 1976. (Unpublished paper.)

A6 Norden, P. V., "Useful Tools for Project Management," Management of Production, M. K. Starr, Ed., Penguin Books, Inc., Baltimore, MD, 1970, pp 71-101.

A7 Brooks, F. P., Jr., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley Publishing Company, Reading, MA, 1975.

A8 Gordon, R. L. and Lamb, J. C., "A Close Look at Brooks' Law," Datamation, June 1977, pp 81-86.

A9 Wolverton, R. W., TRW Systems Group, Redondo Beach, CA, interview.

A10 Shick, G. J. and Wolverton, R. W., "An Analysis of Competing Software Reliability Models," TRW Systems Group, Redondo Beach, CA, December 1976.

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Dr. Margaret M. Rogers Code 31 Naval Weapons Center China Lake, California 93555	1
Dr. R. G. Freeman, III, Rear Admiral, USN Commandant, Defense Systems Management College Ft. Belvoir, VA 22060	1
Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
Office of Research Administration Code 012 Naval Postgraduate School Monterey, California 93940	1
Associate Professor Uno R. Kodres Computer Science Department Naval Postgraduate School Monterey, California 93940	15