

AD-A081 607

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/6 19/5

COMPUTER ARCHITECTURE PERFORMANCE PREDICTION FOR NAVAL FIRE CON--ETC(U)

DEC 79 D M STOWERS

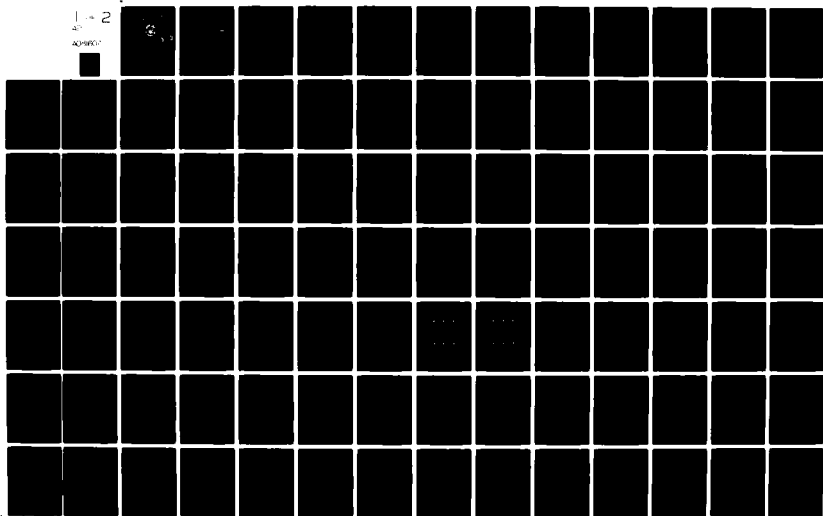
NP552-79-006

NL

UNCLASSIFIED

1 - 2

AD-A081 607



NPS52-79-006

LEVEL #

2
P. 3.

ADA081607

NAVAL POSTGRADUATE SCHOOL

Monterey, California



DTIC
ELECTE
MAR 1 1 1980
S A D

THESIS

COMPUTER ARCHITECTURE PERFORMANCE
PREDICTION FOR
NAVAL FIRE CONTROL SYSTEMS

by

Douglas Monroe Stowers

December 1979

Thesis Advisor:

Lyle A. Cox

Approved for public release; distribution unlimited

DDG FILE COPY

80 3 10 137

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) NPS52-79-006	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Computer Architecture Performance Prediction For Naval Fire Control Systems	(9) 9	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1979
7. AUTHOR(s) (10) Douglas Monroe Stowers	6. PERFORMING ORG. REPORT NUMBER	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940	8. CONTRACT OR GRANT NUMBER(s)	
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 186	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	12. REPORT DATE (11) Dec 79	
	13. NUMBER OF PAGES 105	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	16a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
18. 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)		
19. SUPPLEMENTARY NOTES		
20. KEY WORDS (Continue on reverse side if necessary and identify by block number) Simulation Petri-Net Evaluation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The United States Navy lacks the proper and efficient tools to evaluate/predict the performance of computer systems during the early design phases of system development. This thesis applies state of the art techniques to provide a methodology that can assist in the evaluation/prediction process for Naval fire control systems. The computer system evaluated is a part of a modular addition to existing shipboard gun fire control systems. A contract for the Engineering Development (ED) phase of the program has		

DD FORM 1473
1 JAN 73
(Page 1)EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601UNCLASSIFIED 252450
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

recently been awarded to industry. The computer system architecture is evaluated utilizing a Petri-Net simulation which is best suited to the purpose of concurrent computer system performance prediction. The prediction model, described herein, accomplishes the evaluation with the results being utilized to recommend possible performance improvements in the hardware and software to the U. S. Navy Program Office.

Accession For	
NTIS	☒
DTIC	☐
AD	☐
DA	☐
By	
Date	
Approved	
A	

DD Form 1473
1 Jan 73
S/N 0102-014-6601

UNCLASSIFIED

Approved for public release; distribution unlimited.

COMPUTER ARCHITECTURE PERFORMANCE
PREDICTION FOR
NAVAL FIRE CONTROL SYSTEMS

by

Douglas Monroe Stowers
GS 13, Naval Sea Systems Command
B.E.E., M.E., University of Louisville, 1968
M.S.S.M., University of Southern California, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1979

Author

Douglas M. Stowers

Approved by:

Lytle A. Cook Jr.

Thesis Advisor

Robert S. McLennan

Second Reader

John H. [illegible]

Chairman, Department of Computer Science

A. Strady

Dean of Information and Policy Sciences

ABSTRACT

The United States Navy lacks the proper and efficient tools to evaluate/predict the performance of computer systems during the early design phases of system development. This thesis applies state of the art techniques to provide a methodology that can assist in the evaluation/prediction process for Naval fire control systems. The computer system evaluated is a part of a modular addition to existing shipboard gun fire control systems. A contract for the Engineering Development (ED) phase of the program has recently been awarded to industry. The computer system architecture is evaluated utilizing a Petri-Net simulation which is best suited to the purpose of concurrent computer system performance prediction. The prediction model, described herein, accomplishes the evaluation with the results being utilized to recommend possible performance improvements in the hardware and software to the U.S.Navy Program Office.

TABLE OF CONTENTS

I. INTRODUCTION.....	8
A. BACKGROUND.....	8
B. APPROACH.....	9
II. THE COMPUTER SYSTEM ARCHITECT.....	11
A. INTRODUCTION.....	11
B. COMPUTER SYSTEM ORGANIZATIONAL LEVELS.....	11
1. Levels of Hardware Design.....	13
C. COMPUTER HARDWARE DESCRIPTION LANGUAGES.....	16
1. Design Automation.....	16
2. Digital Hardware Languages Under Development.....	18
a. Conlan.....	19
b. Ddltrn.....	20
D. INTEGRATED CIRCUIT DESIGN.....	22
1. The Problem and the Proposed solution.....	23
2. Can the Architect Exploit This Advance?.....	24
E. SUMMARY.....	26
III. COMPUTER PERFORMANCE EVALUATION.....	27
A. INTRODUCTION.....	27
B. PURPOSES OF PERFORMANCE EVALUATION.....	27
1. Selection Evaluation.....	28
2. Performance Projection.....	29
3. Performance Monitoring.....	30
a. At the Chip Level.....	31
b. At the CP/IO Level.....	31
c. At the System Level.....	32

d.	At the Network Level.....	32
C.	SUMMARY.....	33
IV.	SEAFIRE (ELECTRO-OPTICAL FIRE CONTROL SUBSYSTEM).....	34
A.	INTRODUCTION.....	34
B.	SEAFIRE DESCRIPTION.....	34
1.	Subsystem Definition.....	34
2.	SEAFIRE General Description.....	37
3.	Operational and Organizational Concepts.....	38
C.	REQUEST FOR PROPOSAL.....	40
1.	Microprocessors/Firmware Requirements.....	41
D.	PROPOSED CONTRACTOR SYSTEM DESIGN.....	43
1.	System Description.....	43
2.	General Description.....	45
3.	AN/UYK-20 Functional Description.....	47
4.	Target Motion Analysis (TMA).....	53
E.	SUMMARY.....	56
V.	USE OF AN EXISTING COMPUTER SYSTEM PERFORMANCE TOOL..	57
A.	INTRODUCTION.....	57
B.	DEVELOPMENT OF THE DESIGN EVALUATION SYSTEM.....	58
C.	THE PETRI PERFORMANCE PREDICTIVE PACKAGE (P4)....	60
D.	LIMITATIONS OF THIS APPROACH.....	71
E.	SUMMARY.....	74
VI.	IMPLEMENTATION/EXPERIMENTAL PROCEDURES.....	75
A.	INTRODUCTION.....	75
B.	A DESCRIPTION OF THE PROGRAM.....	75
C.	PRESENTATION OF RESULTS.....	80
VII.	CONCLUSIONS AND RECOMMENDATIONS.....	82

APPENDIX A - PROGRAM LISTING.....	84
LIST OF REFERENCES.....	103
INITIAL DISTRIBUTION LIST.....	105

.. INTRODUCTION

A. BACKGROUND

The fire control system evaluated, the SEAFIRE program, is presently in the Engineering Development (ED) phase of the weapon system development process with system deployment expected some time during the mid 1980's. A Request For Proposal (RFP) for the design of the system was released to industry and was responded to by a number of contractor teams. A thorough evaluation of these proposals, which included technical, management, cost and schedule factors, was performed over a ten month period resulting in contract award to industry during 1979. The contractor, although provided an AN/UYK-20 computer as a baseline processor, was not prohibited from using additional imbedded processors to support/enhance his design. The AN/UYK-20, being a standard Navy computer, was required for use in the SEAFIRE system at this time. The Program Office, however, has plans to replace the AN/UYK-20 with a modern computer prior to the SEAFIRE fleet introduction. These plans though will only be implemented if the AN/UYK-20 is replaced as one of the Navy standard computers and if the Naval Material Command allows it to occur.

The computer architecture, as designed by the contractor, represents an untested real time combat subsystem. This thesis attempts to evaluate the SEAFIRE

computer architecture and provide a meaningful input to the U. S. Navy SEAFIRE program office (Naval Sea Systems Command) as to its predicted performance.

B. APPROACH

The first step to accomplishing this goal was to become familiar with the present design status of the computer architecture and to develop liason with its designer. The present level of the design drove the evaluation to a modular configuration, as would be expected at this point in the design process. A determination was then made to establish the performance measures on which to measure or estimate the performance of the system.

The next step was to research a number of available methodologies for computer architecture performance prediction and to select the one methodology determined best suited for this project. The model selected was designed by L.A. Cox, based on work led by J. Dennis at M.I.T. and other researchers. This model was developed to execute on a non standard CDC-7600 computer system and thus required considerable effort in program modification to enable it to run on the PDP-11/50 minicomputer at NPS.

The remaining work consisted of developing program representations of the SEAFIRE software and hardware for use in the simulator, and finally in the analysis of the results. A number of assumptions, which were required due to

a lack of information, are denoted throughout this thesis. A listing of the final version of the Petri-Net simulator is contained in Appendix A.

II. THE COMPUTER SYSTEM ARCHITECT

A. INTRODUCTION

How does the computer system architect cope with the rapid pace of computer technology? His capability to describe the hardware at specified levels in an efficient, interactive manner that provides a dynamic atmosphere during the life of computer design may be the key. This chapter deals with a spectrum of design techniques that assist the architect.

B. COMPUTER SYSTEM ORGANIZATIONAL LEVELS

According to Doty and Liposki (11) Von Neumann's 1946 paper, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument" is fundamentally one of the most significant papers in computer architecture written; principally because it was written 15 years before the term was coined (Von Neumann claimed no ideas but was merely a focal point for them). This paper outlined the four principle units required of a general computing system; the control unit, the data operator, the memory, and the input/output unit. These units form the conceptual basis of almost all current computers.

What is computer architecture? By "computer architecture" we mean the abstract, functional description

of a computer as seen by a machine-level programmer, that is, everything the programmer needs to know to write programs that run effectively on the computer (i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation). As a result of the changing technologies of processors and memories, deficiencies in earlier designs, as well as innovations in networks and distributed processing, computer architecture is evolving rapidly.

In addition to technology, there are several other key factors that contribute to architectural innovation; most significant are increasingly inexpensive hardware and the rising cost of software (human labor). All future systems should be designed with consideration of these and other factors.

A good system can be defined as a well-organized collection of components chosen to meet the system goal. A modular system is a collection of these component modules. The systems are the largest design units, and subsystems are convenient intermediate-level complexes (18).

One system's components may be another's systems, in different situations. Therefore, a complex system design should be described at a number of different levels which may change dynamically as the design proceeds from concept to implementation.

1. Levels of Hardware Design

Bell and Newell (2) define the levels in the hierarchy of digital computer structures largely on the basis of considering the different activities of different technical practitioners. The 'institutional positions' of logic designers and circuit designers are used as evidence for the existence of distinct levels. Their highest level (the PMS-Processor Memory System level) has computers as structures and processors, memories (storage) etc., as components. The next, or programming level, sees programs as made of component instructions, operators, etc. The three logic design levels are the:

1) Register Transfer Level - arithmetic units made from registers, controls, data operators;

2) Sequential Switching Circuit Level - counters made from flipflops, latches;

3) Combinatorial Switching Circuit Level - encoders, selectors, iterative nets made from logic gates.

The lowest level they consider is the circuit level where example systems (circuits) are amplifiers, clocks and gates and where the components include relays, transistors, resistors, diodes and delays. The essential constraints for the notations to satisfy are ones of completeness, flexibility and brevity (high informational density) (3).

An appropriate criterion might be to identify a level of design with a design description (specification) then:

"A system level . . . is characterized by a distinct specification for representing the system (that is, the components modes of combination and laws of behavior). These distinct languages reflect special properties of the types of components and of the way they combine . . . The fact that the languages are highly distinct makes it possible to be confident about the existence of different levels" (2)

This method of identifying the various levels of system design allows one to identify the most recently emerged levels, but it leads to a significant difficulty. Whenever it is difficult to decide whether two languages are highly distinct, it is also difficult to decide whether they define different levels. Thus it seems as though there are no effective procedures, even in principle, for counting the number of distinct levels of system design. The number of levels, and thus the extent or depth of the levels, are difficult to precisely determine.

This view introduces a new notion: the span (depth) of a level is commensurate with the short term comprehension of a human being. That is, one historical reason for designing a large system in successive stages has been that the human designer has a certain limit to the range of detailed consideration which he can instantaneously handle effectively (although Cray/Amhdahl developed computers individually). If the design process is to be automated, it might be initially done in smaller steps than humans currently handle (for the machines are notoriously inept at handling the intuitive associations which a designer employs). The number and span of the design steps has always been difficult to precisely determine and we should expect

them to continue to change in the future (18). It is important to remember that all our experience to date shows that design automation cannot rely too much on artificial methods; the human has to stay involved.

The design specification is the key to the definition of a level. The language defines the level; is the tool for designing at that level; expresses the components and systems of the level; and provides the documentation for design at that level. The lowest-level, irreducible units of a design are the primitives (words) of the language; the system structures designed at that level are the sentences of the language. Preparing a design at a given level means writing a statement in the language of the level. The process of designing an entire system becomes a process of carefully translating statements in one higher-level language to successively lower levels.

C. COMPUTER HARDWARE DESCRIPTION LANGUAGES

Computer hardware description languages (CHDL's) can be defined as languages for describing, documenting, simulating, and synthesizing digital systems with the aid of a computer (23). A CHDL can be used to describe the logic gates, the sequential machines, and the functional modules, along with their interconnection and their control, in a variation of a programming language tuned to the overall needs of describing hardware.

1. Design Automation

Just as software designers use high-level languages to express algorithms in terms of language statements, so digital hardware designers are beginning to use hardware description languages to describe the digital systems they want to design (24).

The task of designing a digital hardware system can be considered as consisting of the following steps:

- 1) The generation of a system diagram from the specifications of the system to be designed.
- 2) The production of detailed logic diagrams for each subsystem.
- 3) The partitioning of the logic diagram into general units.

4) The assignment of integrated circuit chips for implementing each unit.

5) The placing of chips on logic cards and of cards on boards, and

6) The interconnecting of the chips.

7) The testing of the integrated circuit boards.

Computers have been widely used for aiding steps 4 to 7. A total design automation system requires that steps 1 to 6 be automated. CHDL's can be used for aiding system and logic design as well as partitioning a digital system. A designer can use a CHDL to express his design and leave the exacting, tedious, uninteresting details to a computer (23).

The process of automated logic design may consist of the following steps:

1) A designer expresses his design in a CHDL by writing a program.

2) A hardware compiler (translator) checks the syntax, consistency, etc. of the language statements and reports the errors to the designer for correction. After the errors are corrected, the translator produces a data base to be used by the system simulator and the logic synthesizer.

3) The system simulator models the design at the system level. This will save the large amount of computing time used for simulating everything at the detailed gate level. If the system performance is unsatisfactory, the design language statements are modified. If the performance is satisfactory, the next step is taken.

4) The logic synthesizer (a program) uses the data base produced by the translator, accepts the types and constraints of logic components, and produces a logic diagram.

Since a CHDL constitutes the input to the design automation process, it plays an important role in the task of achieving automated logic and system design.

2. Digital Hardware Languages Under Development

A number of digital hardware languages exist today and are in use by industry as well government sources. One of the most recent uses in the military was for the selection of the Computer Family Architecture (CFA) (1,20). This was a joint DOD effort aimed at providing defense systems developers with a software compatible family of military computers at varied levels of performance that have extensive systems/support hardware. One facet of the selection process was that the measurements and tests of hardware candidates were made, not on the various computers as physical objects, but on their formal descriptions expressed in ISPL (Instruction Set Processor Language) (2).

This was the first time that the architectures of commercially viable computers were described in a formal language, the description compiled, and then used to drive a simulator, executing benchmarks and diagnostic machine language programs. A valid sign for future users is that it

was generally accepted by industry, military and government.

What other methods have been developed since the above? The remainder of this section covers several of the efforts presently being developed as a result of the Working Group of the Conference on Computer Hardware Description Languages.

a. Consensus Language (CONLAN)

CONLAN is a consensus hardware description language capable of representing hardware at several distinct levels of detail (15). The range of language levels suggests a family of languages that share a common basic syntax and are rooted in a common semantic base.

Guidelines laid down for the language follow:

A. CONLAN must support design, description, and simulation of at least the following classes of systems: gate networks, register networks, processors, memories, processor systems. Each class has been fully defined.

B. Any system may be displayed via either (a) a network structure description or (b) a behavior description.

C. CONLAN is to service:

1. Computer architects and logic designers for purposes of trade-off exploration and optimization, design verification, and design documentation.
2. Systems, micro, and applications program-

mers.

3. Electronics production engineers.

4. Maintenance engineers.

D. CONLAN syntax and semantics must support:

1. Well-defined descriptions

2. Machine parsing, interpretation and simulation with error detection (strong typing has been adopted)

3. Comprehension of complex system structure and function

4. Division of design efforts

5. Control over the level of abstraction at which subsystems are described.

6. Simulation control

E. CONLAN will be evaluated in terms of benchmarks such as: standard function declarations, time operator declarations, integrated circuit descriptions (long list, including microprocessors), design descriptions (another long list including a multiprocessor system).

The details of CONLAN (i.e. BNF grammar, etc.) are contained in (15). Since CONLAN is still under development, additional information is available only from the working committee which is developing the language.

b. Digital Design Language Translator (DDLTRN)

Today, the greater complexity of systems, the desire for

short design cycles and error-free designs, and the use of array logic all suggest the need for machine assistance in the logic design activity (8). DDL is a block oriented, statement register transfer language for the description of digital hardware. DDLTRN is a program that translates a DDL description of a digital system to Boolean equations and register transfer statements suitable for driving a companion simulator program DDLSIM (6,9). DDLTRN is written in the IFTRAN (a structured FORTRAN) language (16). Translation consists of replacing the syntax of more abstract language constructs with more explicit syntax, yielding Boolean equations and IF-THEN conditioned register transfer statements.

As mentioned earlier, DDLSIM is a program for simulating digital systems described using DDL (7). DDLSIM does very extensive error checking of described systems, simulation control cards, (same system with different data sets and/or parameters), and the simulation process itself. DDLSIM permits multiple simulation runs within one job in order to either verify the system design or study its behaviors under different conditions.

DDLTRN/SIM and CONLAN are two examples of the growing number of design/automation aids available today. These CHDLs put the the architecture community in the position to explore and develop needed design automation tools. Since Dietmeyer is a member of the above committee, it would be expected that many of his ideas will be incorporated into,

or provide the basic groundwork for future efforts.

D. INTEGRATED CIRCUIT (IC) DESIGN

Ever since integrated circuit designers began to put thousands of transistors onto a single chip, the cost, in terms of human labor, required to lay out the circuit has been extremely high. Although hardware has reached a point of being considerably cheaper than software, the Department of Defense (DOD) requirements for special purpose, limited market chips has seen its time. The need for good design automation in the area of integrated circuit layout is severe. What is needed, and what is evolving, are design techniques which free the designer from the tedious aspects of IC design and allow him to concentrate on the more creative and necessarily human side of the design process.

Using traditional methods, large scale integrated circuit lay out is a tedious, time consuming and error-prone process. For commercial use, where literally millions of identical chips are sold each year, the cost to do this has not been a problem. But for the DOD it is becoming an increasingly significant problem; especially since the DOD market for ICs comprises only about 7% of the total IC market and because environmental and other constraints are becoming more severe (16). The overall goal of an IC design is to pack as much circuitry as possible into the smallest possible amount of "chip real-estate" (IC density).

therefore, higher production yields may be obtained.

1. The Problem and the Proposed Solution

At present, when a company designs large scale systems, there are often delays of months or even years in the development of a prototype IC and the price for a single chip ranges from one quarter to half a million dollars, the DOD is forced to revert to using older technology. This, coupled with the typical eight to fourteen year system development cycle of large computer systems (examples include AEGIS, TACFIRE and CDC STAR 100), has created quite a military dilemma. Add to these problems the stringent requirements for MIL-SPEC qualification, fault-tolerance, built-in test, high clock rates, and the use of advanced design concepts for affordability, causes the required chips not be ready for several years and when available are extremely high in cost to the user.

A new DOD (Tri-Service) program known as Very High Speed Integrated Circuits (VHSICs) began at the start of FY 79 and is a six year effort initially budgeted for in excess of \$200 million dollars. Program goals require a processing throughput capability for computers of between 100 to 1000 times greater than presently exists.

The overall purpose of the DOD program is to:

. Advance introduction of VHSIC into military systems by at least five years ahead of present projections

. Focus industry attention on DOD requirements through the establishment of distinct goals and funding infusion

. Make the latest state-of-the-art devices available for military use in advance of commercial exploitation, thereby reversing the present two to five year lag between commercial development and military availability

. Advance IC technology beyond the limits of optical litho- graphy to submicron dimensions

. Replace over fifty or more present ICs with one IC, thereby providing at least a ten-fold reduction in the size, weight, power consumption and failure rate with accompanying savings in both initial and life cycle costs of present military computer processing systems.

. Provide ICs with 100 times the processing throughput capability of present ICs (16).

By meeting the above stated goals, the DOD expects to achieve affordable chips, reduce potential supply and logistics problems and maximize system reliability. The improved architectural and design concepts should result in a limited chip set with broad applicability to military systems.

2. Can the Computer Architect Exploit this Advance?

Despite these advances that semiconductor technology has created, the question arises as to whether the computer architect can exploit these with proven design methods of

his own. A number of approaches have been actively pursued over the last few years (see previous section). However, there are not currently the languages, operating systems and design methods needed to effectively employ the new LSI devices which can now be produced. We would like to be limited only by economic factors, not technical or theoretical factors. A hope is that a new dimension for the architecture of computer systems will emerge from these design methods so that LSI design methodology can be used effectively. There is a need to proceed slowly and rather cautiously and to introduce somewhat more general purpose description languages selectively.

The military system cannot afford these time delays and much effort is being pursued to shorten this cycle and to obtain industry input earlier. A major directive, Office of Management and Budget Directive OMB A:109 (17), has as a major goal, industry involvement in system development earlier in the conceptual development phase. This thrust, combined with the availability of the tools discussed in this section on design automation and those to be covered on architecture evaluation could be implemented as Concept Development Phase evaluation techniques. The impact would be to provide state-of-the-art computer designs at lower costs with the added effect of shortening the entire development/procurement cycle.

E. SUMMARY

This section has provided the basis to understand computer architecture as viewed by different practitioners and how methods are being developed to assist in early design phases. These techniques can assist the DOD in realizing better structured hardware and to accomplish the tasks required.

The next section further defines the methodology phases of architecture evaluation used to enhance the automated design techniques covered.

III. COMPUTER PERFORMANCE EVALUATION

A. INTRODUCTION

Computer performance evaluation attempts to provide a methodology for examining the adequacy of a computer system as it serves or will serve the needs of its users. In this context, performance may be interpreted as the technical equivalent of the notion of value to the user. In other words, the performance evaluation activities can be regarded as those technical activities whose purpose is the assessment of performance (how well the system works) (12). This chapter discusses different levels of performance evaluation.

B. PURPOSES OF PERFORMANCE EVALUATION

In general, there are three major motivations for performance evaluation: selection evaluation, performance prediction and performance monitoring. These purposes can be classified along several dimensions according to their specific objectives. As with many other system evaluation techniques, these classifications are only convenient ways of organizing a repertoire of knowledge in to a framework which can be more easily understood. The dividing lines between categories are somewhat unclear, but are utilized for lack of a better method.

1. Selection Evaluation

One of the most frequent reasons for initiating an evaluation is to include performance as a "decision criteria" in computer system or digital electronics system decisions for a specified operational requirement. The section on SEAFIRE provides a description of such a system along with an overview of the original evaluation guidelines for procurement of the system. It should be recognized that the computer system was only a subsystem in the context of the SEAFIRE hardware, which in turn was but a single factor in the total weapon system procurement (cost, management, etc. were also weighted as portions of system value). Each competitive contractor teams proposal may have contained one or more superior subsystems, but were judged to have fallen short in many other areas. For example, one of the losing contractors may have had a better computer subsystem, but poorer subsystems in the other areas. Additionally, his management approach or cost proposal may not have been as good as the winners'. Therefore, the weapon system design selected may not necessarily provide the U.S Navy with the "best" computer architecture available, but the overall system approach is probably the soundest and the most cost effective for the Navy.

In general, selection problems may be classified into the following categories (12).

- a. Processing mode selection
- b. Vendor selection
- c. Installation Selection
- d. System Component Selection
- e. Application Program Selection

A definition of each category is not provided since the content is clear.

2. Performance Projection

This evaluation technique may be the least frequently used. The problem here is to estimate the performance of a system not yet in existence (in some state of design). Thus, the evaluation is oriented toward a new system design, both hardware and software. The evaluation technique pursued in this research is encompassed within this category. The performance evaluation of algorithms run on a particular computer architecture is mostly concerned with performance prediction and is restricted, in general, to some form of computer modeling or simulation. In section V a method of conceptually representing computer systems by use of a concurrent control system model is explained. This method forms the basis for the performance prediction system developed by Cox (4) and modified for use here.

3. Performance Monitoring

Once a system is operational, monitoring provides data on the actual performance of the system. The performance statistics that may be obtained while executing test programs aid in future equipment procurement decisions and are employed by the system user for system tuning; in forecasting the impact of changes in the system (either in reconfiguring the hardware or in improving executed software modules). The impact of future technology and computer architecture will greatly affect performance monitoring at all levels of the computer system. Internal and external instrumentation will provide data accessible to the performance evaluator. A distinction should be understood here between performance monitoring (continuous) and an evaluation study. Continuous monitoring is usually performed for a substantial portion of the lifetime of the existing, running system. Its objective is to keep the system's performance under observation in order to detect performance problems as soon as they arise. An evaluation study is generally much more limited in time and is usually triggered by the identification of a performance problem or the suspicion of its presence. The following sections delineate the evaluation aspects at various hardware levels.

a. At the Chip Level

With the trend to large scale circuit integration, performance evaluation through hardware instrumentation is becoming less flexible. The number of leads remain constant or decrease while the number of functions increase with the consequence of fewer test points per function being available. Since the cost per gate has reduced by a factor of 100 over the last ten years, it is now economically feasible to devote some of the circuitry in these chips to auxiliary functions such as performance monitoring. This will provide built-in data analysis without the addition of any hardware.

b. At the CP-I/O Level

At this level the large - scale integrated circuit chips will be interconnected in various ways to implement the hardware instrumentation. Chips such as microprocessors will be used to do the actual work in this area. As with the previous level, lack of test points is a major problem; microprogramming causes an elimination of some probe points. Also, more test points are lost due to the trend towards eliminating peripheral channels. Costs can be reduced by integrating device control units into the processor and transferring information as serial bit streams.

c. At the System Level

At this level, built-in hardware monitors may provide additional assistance. The performance statistics collected by the associated hardware/software can be time correlated through the use of other microprocessors. The result is two fold: first to reduce the overhead of whatever software instrumentation is still required, and second to eliminate the need for external monitoring devices. The important advance at this level is that performance data will be stored under the system's database management system which will allow for on-line display of performance monitoring data. The data is therefore available for on-line input to various scheduling algorithms used to "fine tune" the system dynamically. A major draw-back to this method may be that the evaluation schemes will have difficulty in dealing with the virtual environment of present and future systems. An additional way would be the tendency to less secure systems because of the required critical parameters associated with the performance evaluation schemes.

d. At the Network Level

Distributed processing is the functional distribution and cooperative processing of user applications among multiple, separately located computer systems of the

same and/or different size and characteristics. The decreasing cost of hardware coupled with the increasing performance of distributed systems offers some advantages to performance evaluation at this level. Performance data can be collected locally at each site and transmitted to a central cite for evaluation and will provide a baseline for network tuning. From a global viewpoint though, more factors must be taken into account to assure that suboptimization does not occur.

C. SUMMARY

This chapter has provided a partial outline of performance evaluation techniques used for computer evaluation. Other specific techniques such as benchmark, kernel, analytical model, synthetic programs, etc. are available but not discussed here. A thorough discussion is provided in reference 4. These areas are selection evaluation, performance prediction and performance monitoring. The DOD requires a more defined approach to all these areas but is most lacking in performance prediction techniques.

The next section describes a predictive method which will be applied in this thesis.

IV. SEAFIRE (ELECTRO-OPTICAL FIRE CONTROL SUBSYSTEM)

A. INTRODUCTION

This section provides an overall description of the SEAFIRE Program and outlines its intended capabilities. SEAFIRE is presently in the Engineering Development (ED) phase of the weapon system development process with system deployment expected sometime during the mid 1980's. A Request for Proposal was released to industry and was responded to by a number of contractor teams. A thorough evaluation of these proposals, which included technical, management, cost and schedule factors, was performed over a ten month period resulting in contract award to industry during 1979. The computer subsystem section of the RFP is more formally described and the computer architecture response to this section is described at the component level.

B. SEAFIRE DESCRIPTION

1. Subsystem Definition

SEAFIRE is an electro-optical fire control subsystem modular addition to shipboard gun fire control systems (GFCS). This addition will allow control of the guns and gunfire by the GFCS when ship's sensors can designate

certain targets to SEAFIRE for which an electro-optical sensor is effective. SEAFIRE will also allow uninterrupted operation of GFCSs when the GFCS sensors are ineffective because of performance degradation or are incapacitated by equipment failure, casualty or tactical limitations. SEAFIRE shall consist of an optical director (above deck) and control, test and display units. As a subsystem integrated with a GFCS, SEAFIRE will perform the following functions against Surface Major Combatants, shore vehicles/installations, surface coastal defense craft and river patrol craft(22):

a. Target Detection-SEAFIRE will provide the GFCS with day and night, passive electro-optical imaging for detection, manual and automatic angle tracking, and active laser rangefinding. SEAFIRE will be capable of performing these operations against sea, surface and shore-based targets which can be engaged by the GFCS during electronic countermeasures (ECM), electro-optical countermeasures (EOCM) and Emission Control (EMCON) conditions.

b. Target illumination - Once SEAFIRE has established track of a target, SEAFIRE will be capable of providing laser target illumination for laser-guided ordinance.

c. Other fire control functions - SEAFIRE will be capable of tracking reference points (landmarks, buoys, etc.) to provide navigation data to the GFCS for indirect or offset firing. SEAFIRE shall be capable of sharing its line

of sight (LOS) to the LOS of the GFCS radars for target identification and check sighting.

d. Ancillary Functions - When not employed as a target tracking sensor for fire control, the inherent capabilities of SEAFIRE will provide ancillary functions including, but not limited to: detection of chemical agent clouds, and aiding in navigation, station keeping, friendly operations, surveillance, intelligence collection, swimmer detection and underway replenishment.

In addition, SEAFIRE will be capable of being configured as a SEAFIRE independent GFCS for applications aboard ships on which no other GFCS exists. As a SEAFIRE independent GFCS, SEAFIRE should be capable of performing, in addition to a, b, c, and d above, all functions necessary to engage and direct gunfire against all trackable targets. These functions will include, but not be limited to: direct acceptance of tactical information, interface with ship's stable reference, generation of gun orders and interface with gun mounts.

2. SEAFIRE General Description

SEAFIRE will be comprised of a director, passive imaging sensors, laser transmitter and receiver, as well as support, display, and control devices. SEAFIRE controls and display will be integrated into the consoles in the MARK 86 and 92 GFCS applications. The controls and display in the MARK 68 GFCS application will be configured as a drawer of the AN/SPG-53 radar console. SEAFIRE will have an independent console in the SEAFIRE independent GFCS. The following major component list represents the SEAFIRE baseline(21):

Director

Laser Rangefinder/Illuminator (LR/I)

Thermal Imaging Sensor (TIS)

Television Sensor (TVS)

Computer, Computer Program, and Related Equipment

Maintenance Panel

Interconnecting Cables

Remote Video Displays

Support and Test Equipment

Console

Automatic Video Tracker (AVT)

Interface Module

Video Character Generator

Video Processor

Real Time Clock

SEAFIRE is depicted in the functional block diagram of Figure 1.

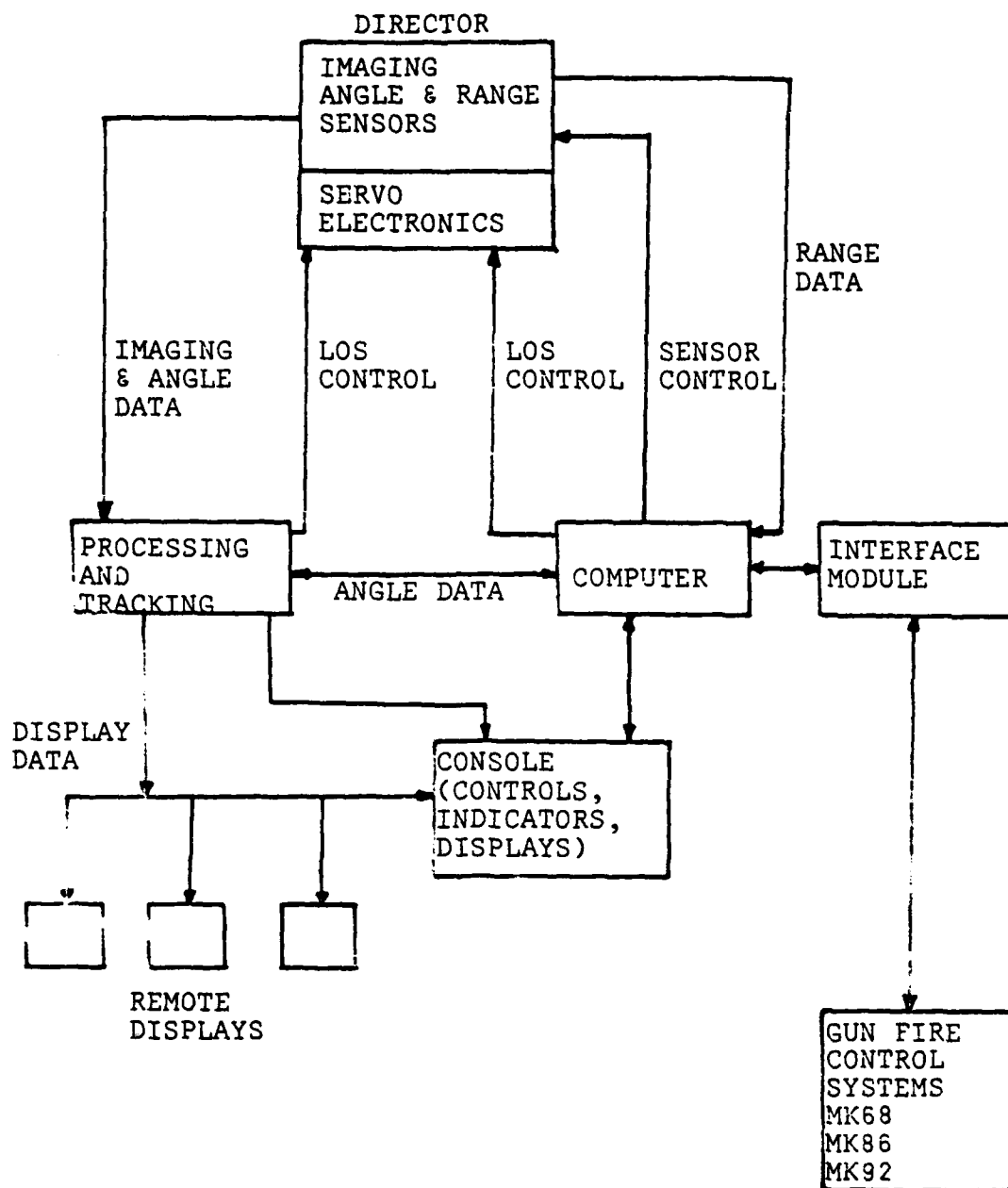
3. SEAFIRE Operational and Organizational Concepts

SEAFIRE will be used in conjunction with the MARK 86, 68 (digital), 92 and SEAFIRE independent GFCSs with ship's interface being provided through the GFCS, except in the SEAFIRE independent GFCS. In all applications, SEAFIRE mode structure and controls should be designed to minimize operator work load. The following list represents some SEAFIRE operational concepts.

a. For engagements in which the fire control radars can provide adequate track data, SEAFIRE may be used predominantly in DESIGNATION/SLAVE for check-sighting, threat evaluation, spotting corrections for fall of shot, and kill/damage assessment.

b. For engagements in which the fire control radars have degraded performance due to ECM or clutter, SEAFIRE will provide independent target tracking data. The fire control operator can then select the sensor which is providing the best track data.

c. In the event of a detection/track function or equipment failure of the GFCS sensor(s), SEAFIRE will provide a total casualty capability for the GFCS,



SEAFIRE FUNCTIONAL BLOCK DIAGRAM

Figure 1

allowing continued gun and gunfire control by providing target tracking data. This will be accomplished by using the GFCS displays and controls, where practical, and the GFCS computer to perform the gun control functions such as ballistics, ammo select, fuze function and code set, signal data transmission and mount status.

d. Under EMCON, the SEAFIRE passive imaging sensors may be used in horizon search, or to evaluate contacts detected by the ship's other passive sensors. If tactics permit limited emissions, the passive imaging sensors may be used with the Laser Rangefinder/Illuminator (LR/I) transmitting single shot to generate fire control solutions while remaining covert.

e. For Laser Guided Ordnance (LGO) engagements SEAFIRE will, as a minimum, provide laser target illumination during the actual guidance time of the LGO. To minimize operator workload during this critical period of an engagement, SEAFIRE should be optimized for automatic target tracking.

C. REQUEST FOR PROPOSAL

As previously mentioned, a Request for Proposal (RFP) was released to industry for design and support of SEAFIRE. The contractor's response required not only a firm system design but also data substantiating his awareness of and implementation experience in production and life cycle

support of major weapons systems. The following is a list of volumes included in the contractor's proposal:

1. Prime Item Development Specification
2. Interface Definitions
3. Master Test Plan
4. Substantiating Technical Data
5. System Project Management
6. Training
7. Support and Test Equipment Plan
8. Contractor Furnished Spares and Repair Parts
9. Producibility Engineering and Planning
10. Technical Manual Organizational Plan
11. LAMPS Electro-Optical POD Engineering Considerations
12. Cost Data

The above list depicts the depth of design/support detail required of the contractor and are only mentioned to provide a top level view of the information used by the U.S. Navy evaluation team.

1. Microprocessors/Firmware Requirements

The SEAFIRE computer (see Figure 1) is an integral subsystem which provides for processing of all data necessary for the functioning of the system. The word computer is a misleading term because it connotes a single item. Although the SEAFIRE contractor was provided an

AN/UYK-20 computer set with peripheral equipment for use during system development and check-out, in actuality he was not prohibited from using additional imbedded processors to support/enhance the AN/UYK-20 processing capabilities. Specifically the use of microprocessors was encouraged.

The RFP stated that microprocessors introduced in SEAFIRE would be selected based on performance, logistic/maintenance support, ease of programming and cost. Additionally, microprocessor architecture would have to be designed to emulate a subset of the AN/UYK-20 computer instruction repertoire such that presently available Navy development software (e.g. CMS-2 compiler, assemble debug tools, data retrieval, data reduction, etc.) could be used to minimize development/life cycle support risk and cost. At least a 20 percent memory reserve and a 35 percent processing time reserve applies to each processor. In addition, the firmware development/documentation/testing and review would be treated the same as the software development documentation/testing phases. Firmware is defined as all software that is not resident in the AN/UYK-20 and is necessary for the operation of SEAFIRE. This includes all programs developed for microprocessors, microcomputers, and microcontrollers. The microprocessors were also to be designed such that effort required to change the program for an inservice SEAFIRE would be minimized.

Based on the above description in the RFP, each contractor team responded with a distinctly different

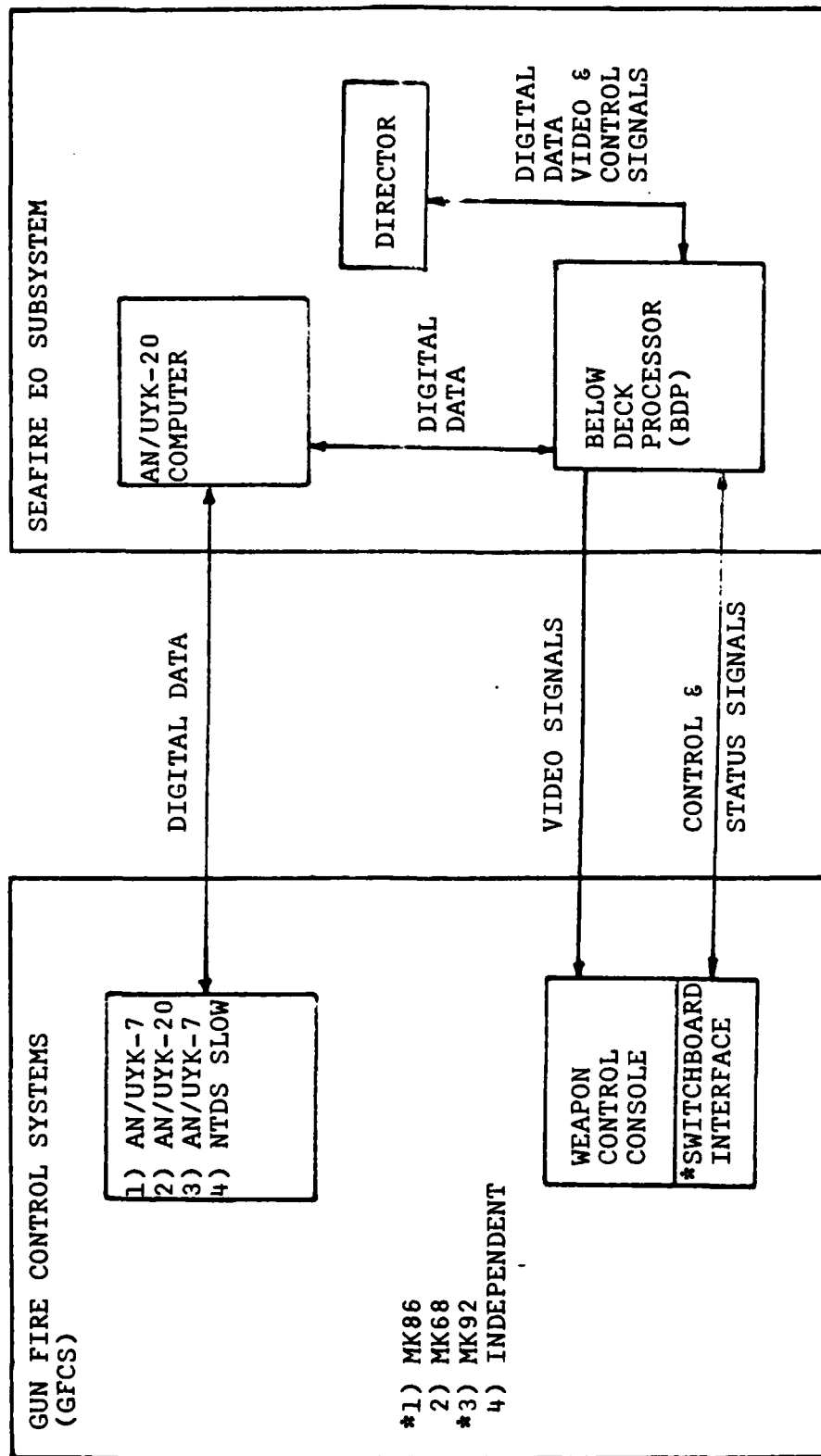
computer architecture for SEAFIRE. Due to this fact and others as stated before, the evaluation of varying computer architectures on the same strict performance factors presented a difficult problem and did not necessarily result in the "best" computer architecture selection. Be that as it may, the design presented in the next section is the evaluation object for this thesis and it is hoped that as a result of the performance evaluation, specific proposals can be suggested which may provide possible system enhancements.

D. PROPOSED CONTRACTOR SYSTEM DESIGN

1. System description

SEAFIRE, as described by the system contractor (21), is an Electro - Optical Fire Control Subsystem (EOFCS) modular addition to existing shipboard Gun Fire Control Systems (GFCS) Mk 86, Mk 68, and Mk 92. This addition allows those functions previously defined.

The modular design of SEAFIRE permits it to be configured as an independent GFCS for application onboard ships on which there is no other GFCS. (See Figure 2) As an independent GFCS, SEAFIRE can perform the functions listed above and all functions necessary to engage and direct gunfire against all trackable surface targets, including direct acceptance of tactical information, interface with own ship sensors, generation of gun laying orders, and



SEAFIRE INTEGRATION BLOCK DIAGRAM

Figure 2

interface with gun mounts.

2. General Description

SEAFIRE comprises two primary equipment groups, which are implemented in accordance with the Standard Electronic Module (SEM) program:

a) The above deck equipment, consisting of the EO director. The EO director includes an enclosed turret, which is mounted on the outer gimbals of the SEAFIRE pedestal. The turret enclosure is designed to house the Television Sensor (TVS), Thermal Imaging Sensor (TIS) and Laser Rangefinder/Illuminator (LR/I). The turret is temperature-controlled to optimize sensor performance.

b) The below deck equipment, consisting of the Below Deck Processor (BDP), Pedestal Electronic Cabinet (PEC), Environmental Control System (ECS), Power Converter Unit (PCU), three remote video displays, and a console.

A common SEAFIRE interface allows integration with host or independent GFCS without hardware or software modifications. The console for the independent GFCS includes the processing for gun order generation and interface with own ship systems. This impacts only the external interface to the applicable ship and not the basic SEAFIRE interface design.

System processing is performed in the AN/UYK-20 computer programmed in the CMS-2 language. Computer program

components are required to implement the following functions: Executive, Input/Output, Control, Displays, Director Control, Target Motion Analysis, Fault Isolation/Detection, and Data Extraction. The program is constructed in modules, with each module structured to perform one of the processing functions. The multitude of functions that must be performed within the system are interfaced and monitored for correctness by the BDP Interface Controller, which also performs the core activities associated with fault detection and location.

As previously mentioned, the contractors' use of microprocessors was encouraged by the U.S. Navy. The contractor has chosen to implement microprocessor technology in the BDP unit. Specifically, microprocessors or microcontrollers are implemented in the following units of the BDP:

- a) Interface Controller (IFC)
- b) Automatic Video Tracker (AVT)
- c) Data Director (DD)

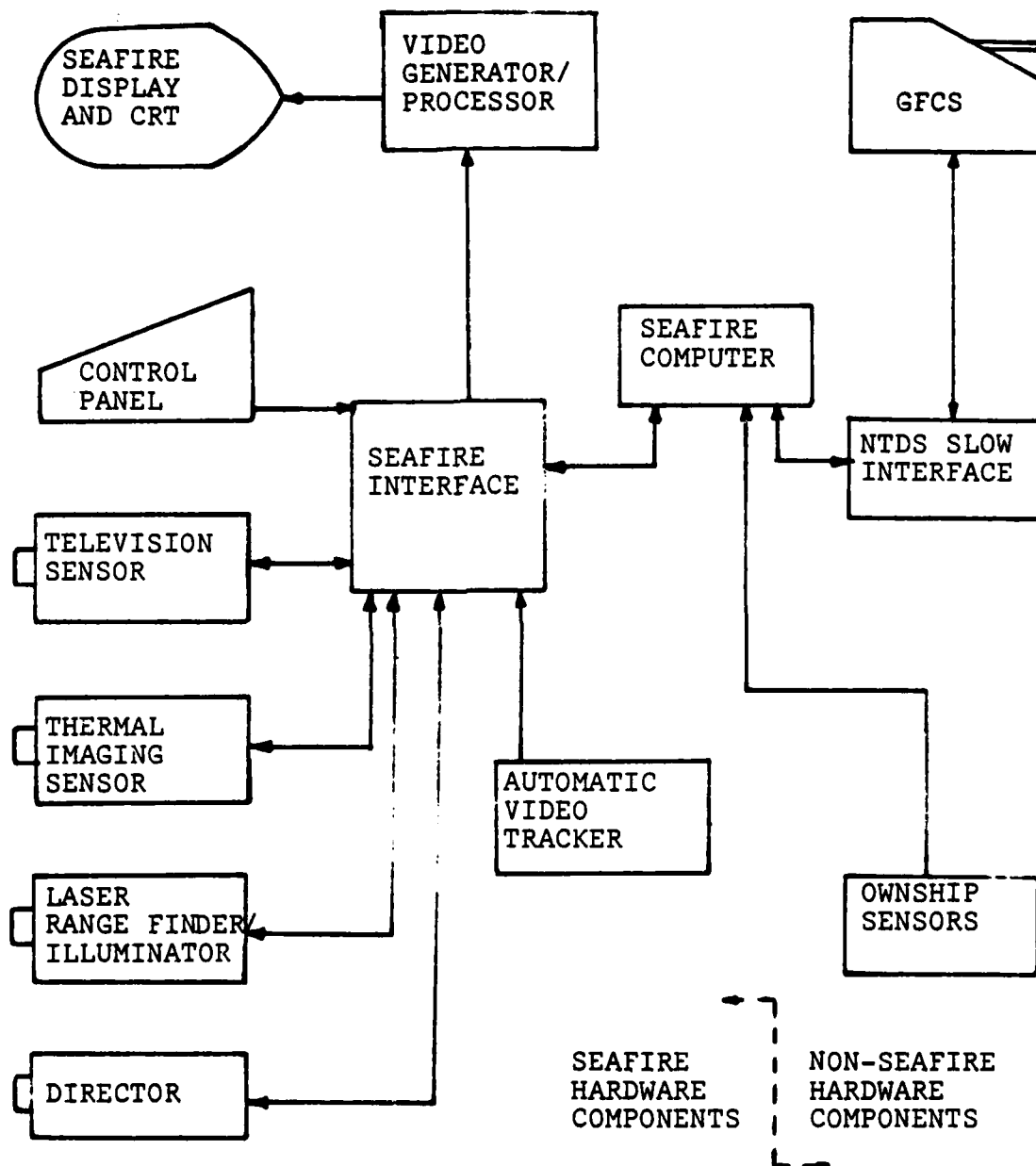
It was originally intended to perform the analysis in this thesis on algorithms running on the microprocessor architecture. But since much of the architectures' software and hardware is still in the process of design and the fact that several areas may currently be proprietary to a contractor or subcontractor, these architectures were not evaluated. The particular facet of the system evaluated (AN/UYK-20 Computer Program Components) will be explained in

a later section.

3. AN/UYK-20 Functional Description

This section provides an overview of the software functional Computer Program Configuration Item (CPCI) and its included Computer Program Components (CPCs). The software architecture and interface are also described. The SEAFIRE computer serves as the controlling center of the SEAFIRE system, receiving data from its separate components, and routing information to those components requiring data from other sources (see figure 3).

The SEAFIRE Interface will provide the SEAFIRE Computer with the means to communicate with all of the SEAFIRE hardware components, collecting data from each component and transferring these data to the SEAFIRE Computer in a single block. Similarly, the SEAFIRE Interface will receive outputs from the SEAFIRE Computer and distribute these data among the SEAFIRE hardware components. To the SEAFIRE Computer, all of the SEAFIRE hardware components appear to be a single device, because a single block transfer is performed for both input and output. Furthermore, a single input interrupt and single output interrupt is involved. Due to the appearance of a single input/output device relative to the SEAFIRE Computer, the software is discussed in terms of the SEAFIRE Interface (ie, same as Interface Controller or Below Deck Processor).



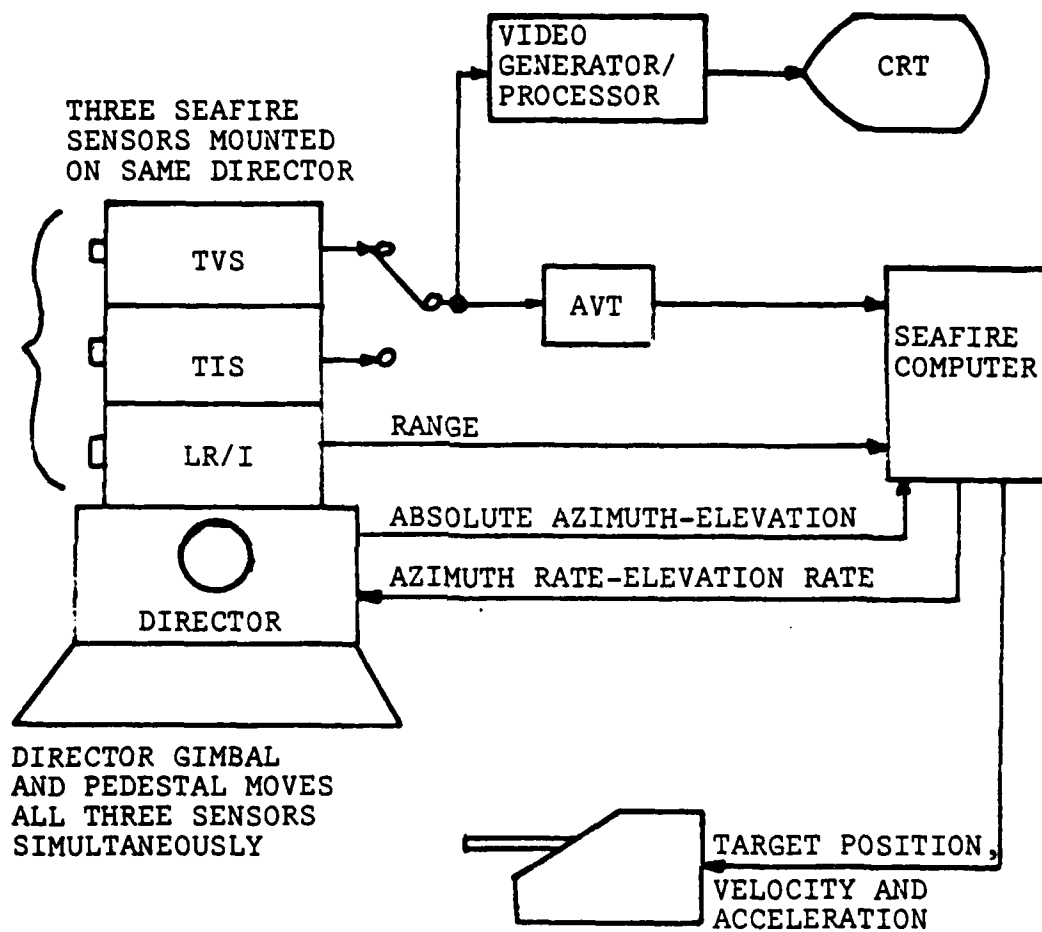
SEAFIRE HARDWARE COMPONENT BLOCK DIAGRAM

Figure 3

The host GFCS Operator will be able to control and monitor the SEAFIRE System at the Weapon Control Console (WCC). The WCC is upgraded to include a SEAFIRE Control Panel for control, and a shared Video Display for monitoring. The Television Sensor (TVS) or Thermal Imaging Sensor (TIS), used with the AVT, and director position readouts will provide the SEAFIRE Computer information necessary to determine target azimuth and elevation. The Laser Rangefinder/Illuminator (LR/I) will provide the range to the target. Using information from these sources, the SEAFIRE Computer will be capable of outputting target position, velocity, and acceleration to the GFCS for engaging the target. The optically aligned TVS, TIS, and LR/I common optical pointing will be controlled by a single azimuth and elevation rate command from the SEAFIRE Computer (see figure 4).

The target image data received by the TVS and TIS will be sent to the AVT, where the target position is calculated. The AVT will determine target position relative to the upper left corner of the video raster and send the target relative position data to the SEAFIRE Computer at a 60 Hz rate.

AVT data may come from either the TVS or TIS, but not simultaneously. The data source is specified by the operator at the SEAFIRE Control Panel. Additional options are available at the SEAFIRE Control Panel that affect the data flow from the TVS/TIS to the AVT and actual processing



SEAFIRE HARDWARE COMPONENT DATA FLOW DIAGRAM

Figure 4

within the AVT, embedded microprocessor. The operator may select one of up to six filters to modify the video input at the TVS/TIS. For the TIS only, he may control the Gain, Bias, and select either Black or White track. For the TVS/TIS he may control video Enhancement, Focus, and select either Wide Field Of View (WFOV) or Narrow Field Of View (NFOV). At the AVT, he may select either Scene or Point digital tracking.

The SEAFIRE Computer will pass the target position through a Kalman Filter; (1) to smooth the target position to a steady state, (2) to calculate target position, velocity, and acceleration and (3) to predict where the target will be in the next update cycle. The SEAFIRE Computer will then output target position, velocity and acceleration via NTDS Slow Interface, to the GFCS, so that the GFCS can compute a ballistic solution. The data input and output over the NTDS Slow Interface will be identical for the four configurations of the SEAFIRE System (Mk 86, Mk 68, Mk 92 and Standalone); therefore, only one version of the computer program need be maintained. The development and maintenance of only one computer program reduces costs and accents software commonality. The SEAFIRE Computer also will output commands to move the Director so that the target will remain in the TVS/TIS FOV.

The SEAFIRE Computer contains one Computer Program Configuration Item (CPCI); the Operational CPCI. The Operational CPCI is used as a GFCS to provide target

tracking and engagement and to maintain the SEAFIRE system in a state of operational readiness. The Operational CPCI performs eight major functions:

- a) Executive
- b) Input/Output
- c) Control
- d) Display
- e) Tracking
- f) Director
- g) Fault Isolation/Detection
- h) Data Extraction

The CPCs listed below perform the eight major functions of the Operational CPCI:

- a) Executive
- b) SEAFIRE Input Interrupt
- c) SEAFIRE Output Interrupt
- d) NTDS Slow Input Interrupt
- e) NTDS Slow Output Interrupt
- f) NTDS Fast Input Interrupt
- g) NTDS Fast Output Interrupt
- h) Control Panel Input
- i) Control Panel Processor
- j) Director
- k) Designation
- l) Target Motion Analysis
- m) Alphanumeric Display
- n) Symbology Display

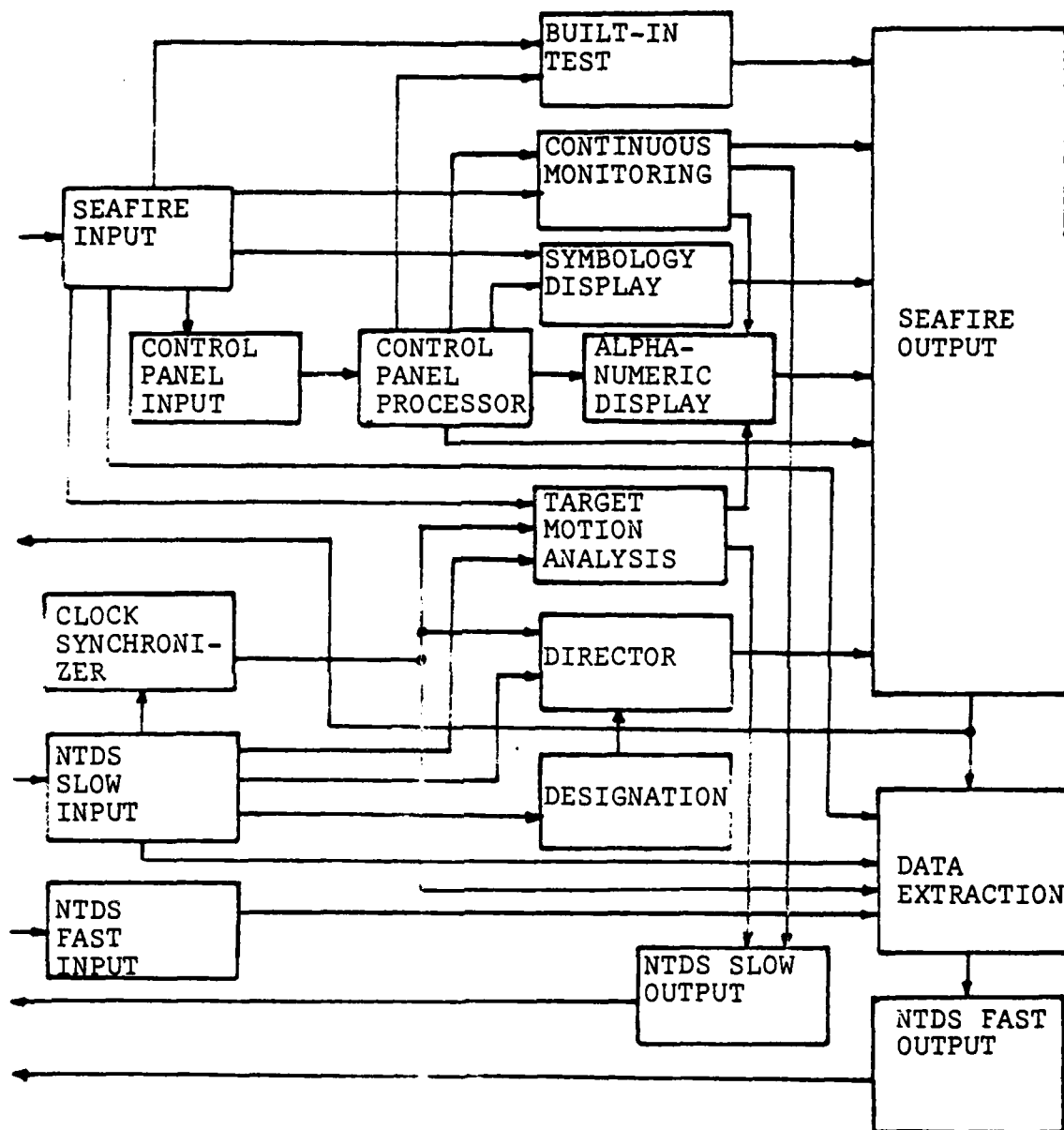
- o) Built In Test
- p) Performance Monitoring
- g) Data Extraction
- r) Clock Synchronization

The functions allocated to each will not be described in detail. The data flow between each of the above CPC functions is shown in Figure 5. As can be seen Target Motion Analysis (TMA) it is a major central function to the system as it includes I/O to several other key functions. A description of the TMA is delineated in the next section.

4. Target Motion Analysis (TMA)

The TMA CPC is called by the Executive CPC at a 4 Hz rate to compute target position, speed, and acceleration for output to the GFCS and to the Display CPC. Executive rate will be 4 Hz since GFCS outputs are required at this rate. The TMA CPC will use the AVT reported target position relative to the raster upper left corner position, Boresight Offset, Sensor Type, and Director Azimuth and Elevation to determine the target position, velocity, and acceleration.

The Director will provide inputs necessary to determine Sensor Line Of Sight (LOS) in terms of azimuth and elevation, and the AVT will provide inputs such that target azimuth and elevation relative to the LOS can be obtained. In manual track, only the Director angles are used. The LR/I will provide target range as the input.



SEAFIRE COMPUTER SOFTWARE DATA FLOW

Figure 5

The AVT will report the target azimuth error and elevation error. This position must be transformed to a position relative to the LOS, and must be adjusted further for the effects caused by Control Panel selections. Finally, the position in elements must be converted to units in degrees. Control Panel selections have the effects listed as follows:

- a) The number of degrees/element is different depending on wide or narrow FOV selection.
- b) The Boresight offset varies as a function of TVS or TIS sensor selection.
- c) The algorithm can be changed, and therefore, the target position.

The TMA CPC will include the necessary processing to:

- a) Correct for angle bias, convert target data to the appropriate reference frame, and correct for parallax.
- b) Prefilter the data to correct for timing delays.
- c) Perform TMA computations required to derive smooth target state variables (position, velocity, acceleration) in both the stabilized Spherical and Cartesian coordinate frames.
- d) Perform necessary computations during coast conditions.
- e) Output track quality data.

At the end of the Kalman filter, maneuver detection

is performed. The maneuver detection subroutine is part of the TMA CPC but is not discussed due to its classification.

E. SUMMARY

Now that the system has been described and methodologies have been discussed in general for performance evaluation of computer systems, the next logical step is a specific application of one of these techniques. The next section provides a description of the performance tool that is to be applied in the evaluation of the SEAFIRE system.

V. USE OF AN EXISTING COMPUTER SYSTEM PERFORMANCE TOOL

A. INTRODUCTION

The methodology to be used for the computer performance evaluation is the one designed by L. A. Cox, Jr. (4). This section provides a summary of his approach.

In his dissertation, Cox described the development of a methodology for efficiently predicting concurrent computer system performance. This methodology allows the estimation of performance of an existing (or conceptual) computer organization operating on a linear mathematical algorithm. An existing program is taken and the control structure of all or some representative kernel of the code is expressed in a fashion which makes the potential parallelism exploitable. For a given computer system, the control structure dictated by the software can then be mapped onto the hardware structure, and the performance predicted.

The key to this process is the representation of a kernel program or one of the basic cyclic events as a special kind of Petri Net similar to a marked, directed graph. In the directed graph, each arc can be regarded as having some propagation delay which is dependent upon the performance of the computer system executing the program. If these delays are fixed and known, then the question of performance reduces to a question about the minimum period for the cyclic behavior of the marked graph which represents

the program.

A requester/server interface provides for construction of a two graph structure which allows the representation of algorithms and hardware organizations by separate graph structures. This permits each graph to be constructed in such a manner as to both express the control structure and to maintain a direct and meaningful representation of the important concepts being modeled.

B. DEVELOPMENT OF THE DESIGN EVALUATION SYSTEM

An effective concurrent computer system design tool must consider the characteristics of both systems and software on a more conceptual level. Hopefully, the same descriptive system could be employed to describe both the hardware organization and the software requirements. The design evaluation system should provide for the inclusion of varying levels of detail in some hierarchical manner and should provide quantitative results of concurrent systems in some cost effective manner.

Why use Petri-Nets for the predictive system? A Petri-Net may be thought of as an abstract, formal model of information flow. As such, it is possible to describe not only the information flow, but the controls and constraints of such flow. The Petri-Net graph models the static structure of a system in much the same manner as a flowchart models the structure of a computer program. In order to

represent the dynamic properties of the system to be modeled, a Petri-Net can be "executed" to respond to the flow of information (or the occurrence of events) in the system.

The static graph of a Petri-Net is composed of two types of nodes, circles which are traditionally called places, and bars which are called transitions. These nodes are connected by directed arcs which run from either places to transitions or from transitions to places. The source of a directed arc is referred to as the input, while the terminal node is referred to as the output.

The dynamic execution of a Petri-Net is controlled by the position and movement of information, as represented by markers which are called tokens. Movement of the tokens proceeds according to certain rules. A token or tokens move when a transition fires. In order to fire, a transition must be enabled, that is all of the places which are inputs to the transition may fire. When a transition fires, the tokens are removed from the input places, and tokens are placed on all output places of the transition.

Petri-Nets can model actual parallel processes by attaching some significance to token movement. For example, multiple outputs from a transition create multiple tokens upon firing, which could be interpreted as a "fork" operation activating multiple parallel processes. Similarly, the multiple inputs to a transition (which must all be marked for the transition to fire) could be interpreted as a

'join' operation terminating or merging independent parallel sequences.

In each case, the status of the execution at a given time can be described by defining the status of the tokens. This distribution of tokens in a marked Petri-Net is called the marking, and defines the state of the net for a given instant. Figures 6 through 9 show the different stages of a marked Petri Net progressively at incremental time units in the system.

As first formally defined by Petri, Petri-Nets were not always deterministic. For the purposes of performance evaluation, a small restriction was made to eliminate non-determinism, something not generally sought after in either hardware or software.

Petri-Net concurrent control system models have many characteristics which are desirable in a concurrent computer system performance prediction system. This model is capable of representing both hardware and software systems and is hierarchical in nature. These characteristics are important in the predictive system.

C. THE PETRI PERFORMANCE PREDICTIVE PACKAGE (P4)

The requirement for an architectural design aid existed. Cox created and implemented on an experimental basis, a performance prediction system based on Petri-Net models. The system, named P4, standing for Petri Performance Predictive

A MARKED PETRI NET (TIME=0)

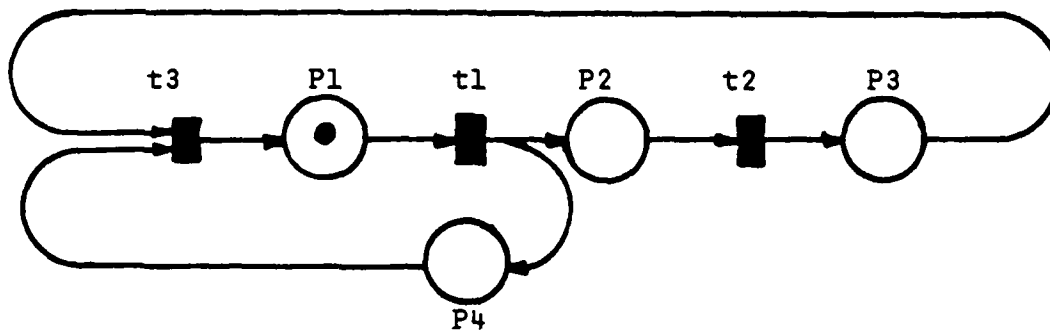


Figure 6

A MARKED PETRI NET (TIME=1)

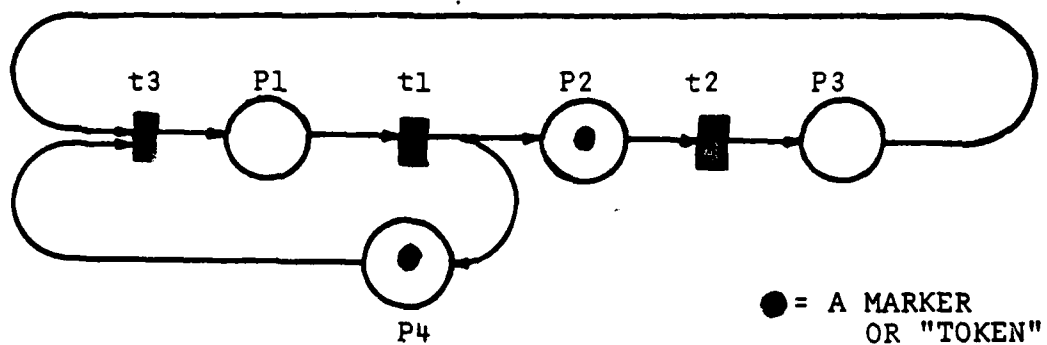


Figure 7

A MARKED PETRI NET (TIME=2)

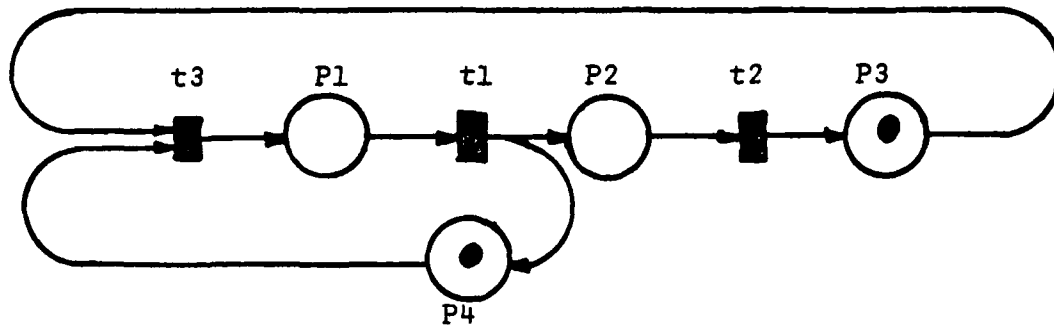


Figure 6

A MARKED PETRI NET (TIME=3)

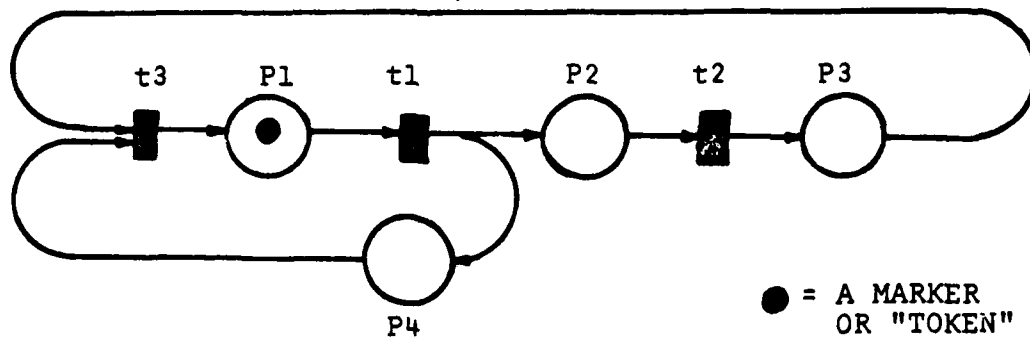
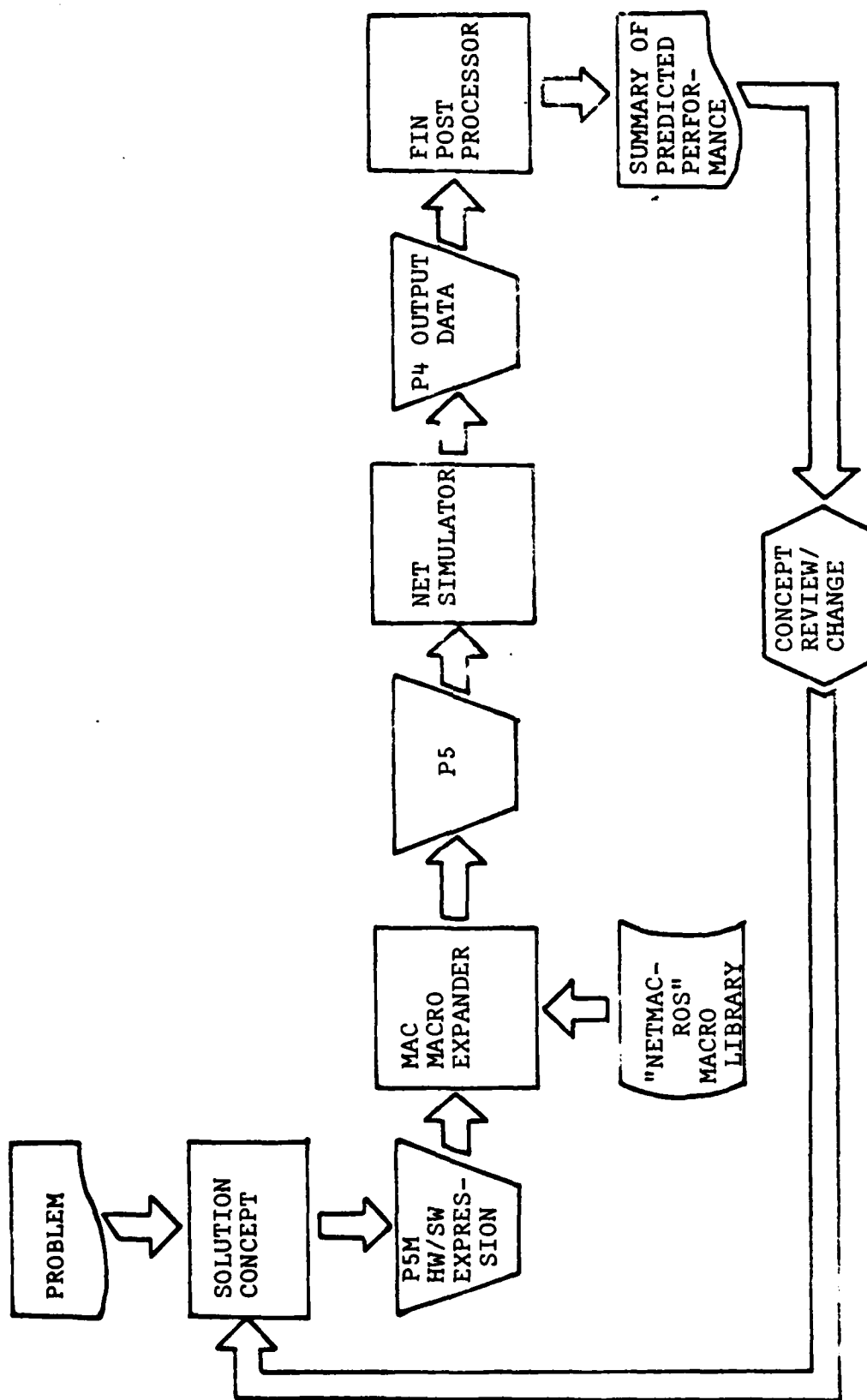


Figure 9

Package, is described below. Major components of the P4 system, and the system's intended employment are shown graphically in figure 10. The model implemented at NPS does not utilize the MAC macro expander or the macro library.

The design of a new computer system (or the modification of an existing system) is usually initiated with the realization that a problem exists whose solution is both important, and not economically feasible in some sense. The P4 system is intended to be used in cases where a problem has been defined and a system architecture is to be developed. In response to this problem, the designer develops a solution concept. This concept includes the algorithmic portion of the problem, and some computer organization which hopefully will solve the problem within the various constraints.

At this point the designer describes this solution concept in terms of the P4 system. A P4 program (P5) consists of a discription of the computer system organization and capabilities. As we will see later, these descriptions are Petri-Nets, and in order to make use of the heirarchical nature of these nets, and to express system organizations in a more concise and convenient manner, a macroprocessor was included in the system; although one is not used in this thesis. A P4 program can be either a "pure" P5 description, or can make use of the macro facility, in which case it is referred to as a P5M description. This description of the solution concept is then



THE P4 SYSTEM OVERVIEW

Figure 10

evaluated in a dynamic sense and produces an analysis of the system's predicted performance.

The performance predictions are made on the basis of the execution of a Petri-Net simulator. This simulator operates on the P5 description of the proposed system. A complete P4 program which is to be evaluated by the Petri-Net simulator consists of three sections: a hardware section, a software section and a dynamic section. The hardware section consists of a description of the basic subsystems of the computer system and some degree of subsystem interconnections. The network which represents hardware is quantified in terms of its operation in time. The software section consists of a description of a Petri-Net which represents the algorithm to be executed on the system. This net is quantified in terms of the basic functions which are to be required of the hardware. The dynamic section contains certain output instructions and specifications of the Petri-Net's initial conditions. Formally, both the software section and the hardware section are merely descriptions of static Petri-Net structures. Performance prediction comes from the attachment of certain significance to the structures and certain restrictions on the movement of tokens or markers within these networks.

The dynamic nature of Petri-nets is used to approximate the activity of the proposed computer system as it executes the algorithm of interest. Accordingly, the two Petri-Nets, software and hardware, can be viewed in a

requester/server context. The software or algorithm makes a series of requests for the services of the computer system. The computer system fulfills these requests according to the constraints of its design.

In the hardware net, events roughly represent operations in time. A collection of one or more events are used to represent a functional unit and its temporal response to the hardware control constraints. Token movement through the hardware net represents the data and control flow of the hardware system. A simple example of a P5 hardware description is shown in figure 11.

The software net's events represent basic requests for service. For example, an event might represent a request for an integer addition. The flow of tokens, which is initiated by a single marker on the event "BEGIN", represents the logical flow of the algorithm. An example of the hardware and software net is shown in figure 12.

Once these two Petri-Net structures have been defined, they can be executed together in a manner which will simulate the operation of the computer system. The interaction of the two nets is controlled by the 'requester/server token arbiter.' The network simulation begins with the marking of the "BEGIN" node of the software net. This net is then executed according to standard Petri rules. The arrival of a token at a place in the net is interpreted as a request for service, the type of service depending on the type of the place. Upon arrival, the

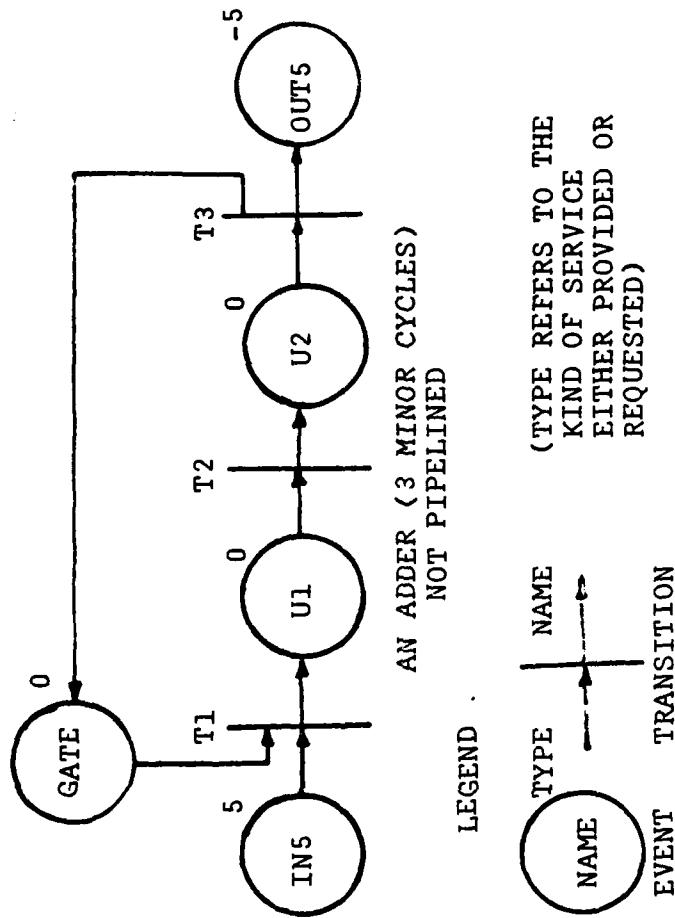
P5

```

BEGIN MACHINE NET;
DECLARE IN5 EVENT TYPE 5;
DECLARE GATE EVENT TYPE 0;
DECLARE U1 EVENT TYPE 0;
DECLARE U2 EVENT TYPE 0;
DECLARE OUT5 EVENT TYPE -5;
DECLARE T1 TRANSITION;
INPUT IN5;
INPUT GATE;
OUTPUT U1;
END T1;
DECLARE T2 TRANSITION;
INPUT U1;
OUTPUT U2;
END T2;
DECLARE T3 TRANSITION;
INPUT U2;
OUTPUT GATE;
OUTPUT OUT5;
END T3;
END MACHINE NET;

```

PETRI-NETWORK



A P4 EXAMPLE (HARDWARE FUNCTIONAL UNIT)

Figure 11

P5

```

BEGIN PROGRAM EXAMPLE;
DECLARE BEGIN EVENT TYPE 0;
DECLARE J+K EVENT TYPE 5;
DECLARE M+J EVENT TYPE 5;
DECLARE END EVENT TYPE 0;
DECLARE ST1 TRANSITION;
INPUT BEGIN;
OUTPUT J+K;
OUTPUT M+J;
END ST1;
DECLARE ST2 TRANSITION;
INPUT J+K;
INPUT M+J;
OUTPUT END;
END ST2;
END PROGRAM EXAMPLE;

```

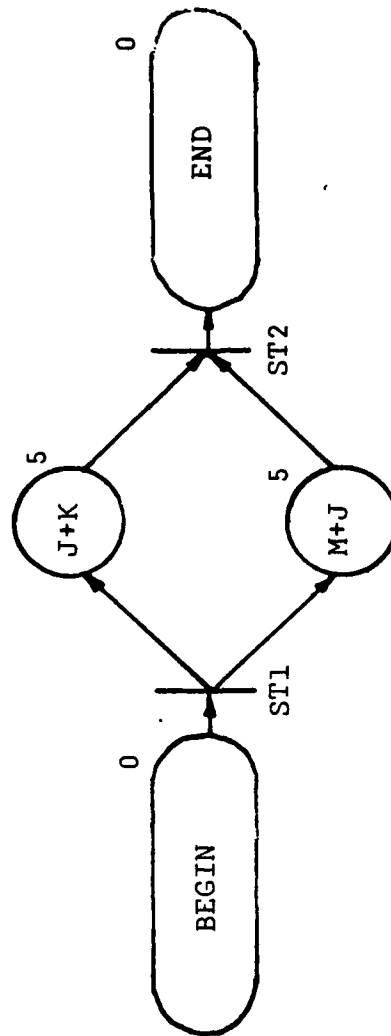
FORTRAN PROGRAM

```

C BEGIN (EVERYTHING IN REGISTERS)
I=J + K
L=M + J
C END

```

THE PETRI-NET REPRESENTATION:



A P4 EXAMPLE (SOFTWARE PROGRAM)

Figure 12

arbiter is notified of the request for service.

The arbiter removes the token from the net, and allows the software net execution to continue until such time as no further moves are possible. At this time, the arbiter initializes the appropriate hardware units which correspond to the requests by marking them with tokens.

The hardware net is then executed one step. If any tokens reach events which correspond to completion of requested service, the arbiter is notified. Here again, the token is removed, and the token of the software net whose movement caused the original request for service is replaced by the arbiter. This cycle is repeated, with the execution of the software network, followed by execution of the hardware network. A P5 dynamic section and the P4 results of the examples shown in figures 11 and 12 are shown in figure 13.

Examples were tried by Cox and predicted results agreed well with actual measurements in most cases. Some cases with wide discrepancies pointed out a significant characteristic of the P4 methodology. When maximally parallel representations of the hardware and the software are provided to P4, the resulting prediction in most circumstances represents the "best case" execution time. This means that in cases where a system has been either implemented or simulated at the bit level, P4 predictions can be compared with bit level timings and used as an indication of the efficiency of the assembly code generated

P5	P4 OUTPUT
BEGIN DYNAMIC NET;	S5: EXECUTE 10;
MARK GATE WITH 1;	*PROGRAM EVENT J+K REQUESTS HW SVCS(1)
COMMENT: GATE ENABLED, TO ALLOW ONLY 1 OPERATION IN PROGRESS AT ANY TIME	*PROGRAM EVENT M+J REQUESTS HW SVCS(1)
EXECUTE 10;	TIME = 1:
COMMENT: EXECUTE 10HW CYCLES ON UNTIL PROGRAM IS COMPLETE. (WHICHEVER OCCURS FIRST.)	TIME = 2:
END DYNAMIC NET;	TIME = 3: *PROGRAM EVENT J+K COMPLETES (3)
	TIME = 4:
	TIME = 5:
	TIME = 6: *PROGRAM EVENT M+J COMPLETES (6)
	S6: END DYNAMIC NET;

A P4 EXAMPLE (DYNAMIC SECTION)

Figure 13

by either manual or automated means.

The results Cox received indicated that the P4 methodology provides not only a simple and accurate method for predicting computer system response but is economical of modeling resources as well.

D. LIMITATIONS OF THIS APPROACH

The Petri-Net is a concurrent control system model of demonstrated power; however, Cox indicated that it does have some limitations, perhaps the most significant of which is its inability to represent conditional events. Petri-Nets are not able to handle these conditions as they are traditionally designed. Some work has been done on developing extensions to Petri representations which consider this situation though a model which basically represents data as tokens is difficult to extend to data value dependent situations.

Cox indicated that these extensive modifications do not appear to be justified in view of the intended operation of the performance model. In general, the linear mathematical models which drove his research can be characterized by a single or at most a few main computational loops. The performance of the loop calculation drives the overall performance of the program. These loops can be represented as linear code, and their performance evaluated. Using this methodology, the conditional path problem is avoided.

Another limitation of the P4 approach stems not so much from the concept, but from the realization. Both software and hardware must be described in terms of descriptions of Petri-Nets. These descriptions are "programs" which are subject to all of the problems of any human generated program.

Experience has shown that the representation of existing computer programs and algorithms as Petri-Nets is usually straightforward. Few errors have occurred at this stage. The automatic generation of these descriptions from a FORTRAN or other algorithmic language source may be possible.

The representation of hardware structures has proven a bit more complex. The hardware Petri-Net program must carefully include all explicit and implicit limitations to concurrency which the system will impose. This requires careful consideration of each design, and careful programming, sometimes by persons without significant programming experience. In hardware systems which make use of variable time intervals for execution (such as the data dependent nature of completion signaling devices), some average propagation delay must be substituted. This complicates somewhat the programming problem by demanding a detailed analysis of some sub-systems, and by including "average performance" figures.

There is one other property which should be mentioned. Currently, there is considerable discussion of "data flow" computer architectures. These are machines which would be

based on the principle of executing instructions in response to the arrival of operands rather than in response to some sequential or explicit control flow. These machines are conceptually important because programs expressed in data flow form are free from sequencing constraints other than those required by the algorithm, and a processor using data flow representation can achieve highly parallel operation. In the Petri performance model, all programs are expressed in essentially a data flow notation. A Petri performance prediction as previously described makes use of all the possible parallelism of both the hardware and software, and is thus "best case" in some sense.

This "best case" prediction property stems from the fact that when properly represented in Petri-Net structures the hardware and software descriptions describe potential parallelism on a global basis. The mapping of requests for service into actual hardware operations makes use of this global parallelism, and the limits are only those explicit in either the hardware or software. It is this property of the Petri performance model that makes it useful in the evaluation of the efficiency of generated code, and makes it a valuable tool in investigations of compiler and language development for highly parallel machines.

Cox's initial experience using the P4 methodology has shown that performance predictions based on dual Petri-Net representations of hardware and software structures are accurate and efficient in terms of resources required to

make the predictions. Additionally, the system is easy and sufficiently general so as to permit detailed investigations of alternative computer system organizations such as would be expected in the design and development of a new system such as SEAFIRE.

E. SUMMARY

The Petri performance model has some limitations which must be understood before it can be properly applied; however, when intelligently used, it comes very close to fulfilling all the goals of an ideal design tool intended for use in the conceptual development of concurrent computer system organizations. The next section deals with the actual implementation of the technique described in this chapter.

VI. IMPLEMENTATION/EXPERIMENTAL PROCEDURES

A. INTRODUCTION

This section provides a description of the hardware and software model for SEAFIRE and how this model was executed by the Petri-Net simulator. Some of the detail that was required concerning the actual functioning of the SEAFIRE software was not available and therefore certain assumptions had to be made in order to develop these networks. The results of the analysis is covered as a function of the number of target loops (TMA) generated.

B. A DESCRIPTION OF THE PROGRAM

A discussion of Computer Software Data Flow at the CPC level was provided in Chapter IV along with a diagram of how the CPCs interface (Figure 5). Since it was decided to perform the analysis at the CPC module level, a representation of the hardware function for each module is best represented as a time interval delay as predicted by the contractor. Table 1 depicts the contractor's timing estimates for each module in the Automatic Track Mode. These figures have been rounded off for ease of implementation.

Figure 14 represents the SEAFIRE hardware (Machine Net). Each execution cycle (D1,D3,...D260) is utilized for one or more of the CPCs of Table 1. The interrupt cycle represents

the first seven CPCs listed. These interrupts occur at a rate of 645 per second; and since one cycle equates to 100 usec, one interrupt would occur approximately every 15.5 cycles. The other calculations are linear representations of the execution time for each CPC.

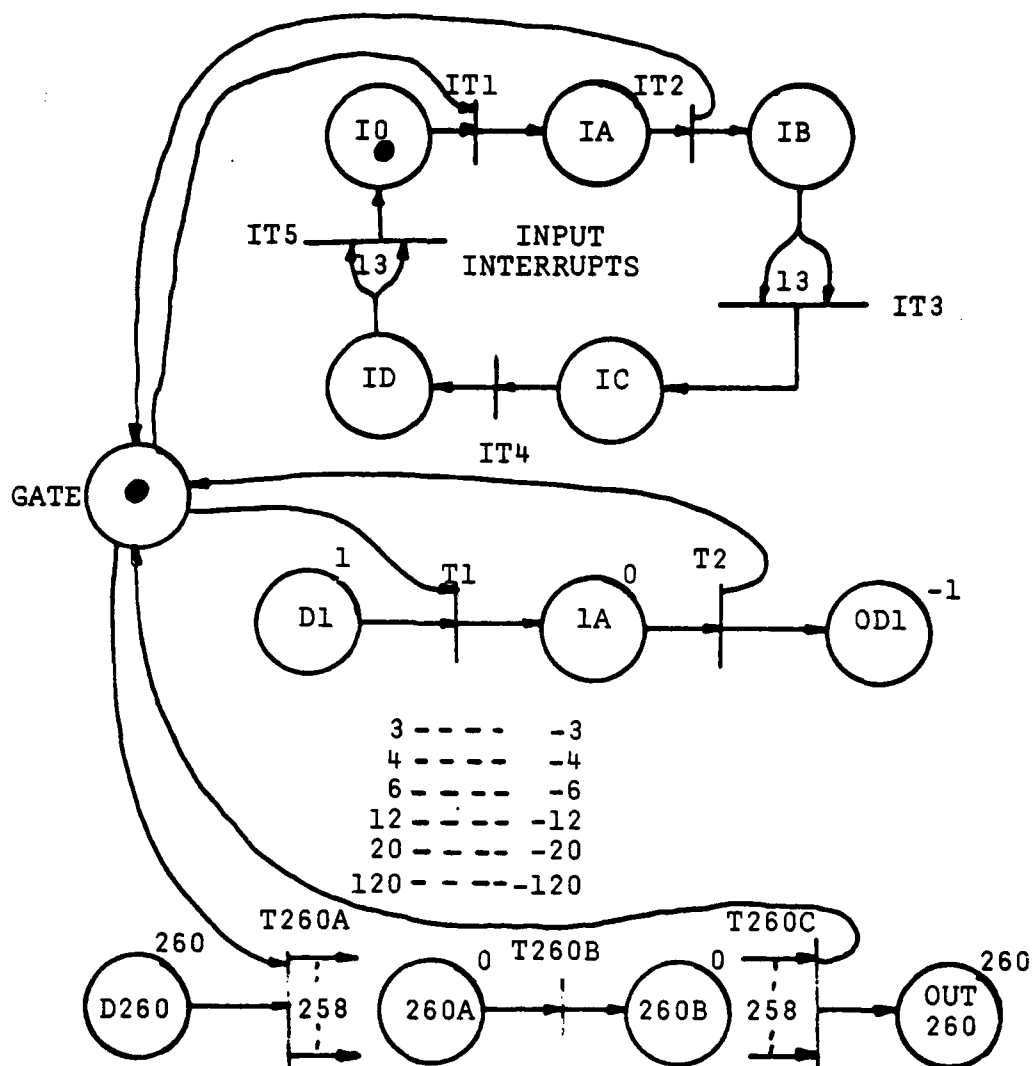
Figure 15 depicts the best estimate of how the software functions for SEAFIRE in the Automatic Track Mode. Steady state was assumed so that the designation function could be ignored. TMA was first executed for a total of two target loops, then was varied on additional runs. The intention was to determine the loading capacity for the SEAFIRE computer at these varying stages of number of target loops. The other routines are interrupt driven from a clock and are depicted in the overhead loop.

As previously mentioned, the basic simulator was available in a form which ran on a CDC-6700 computer. A large amount of effort to modify this simulator resulted in the program of APPENDIX A that now runs on a PDP-11/50 minicomputer at NPS. Computer printouts of the resultant output is not provided as it was felt that it would not have been of significant benefit to the reader. The results of the analysis are discussed in the next section.

Computer Program Components	Time Per Execution(100us)	Rate(Hz)
Executive	1	490
SEAFIRE Input Interrupt	1	60
SEAFIRE Output Interrupt	1	60
NTDS Slow Input Interrupt	1	16
NTDS Slow Output Interrupt	1	16
NTDS Fast Input Interrupt	1	1
NTDS Fast Output Interrupt	1	1
Control Panel Input	4	60
Control Panel Processor	6	10
Director	6	60
Designation	70	16
Target Motion Analysis	260	4
Alphanumeric Display	12	60
Symbology Display	20	20
Built-in Test	3	60
Continuous Monitoring	120	1
Data Extraction	3	60
Clock Synchronizer	1	1

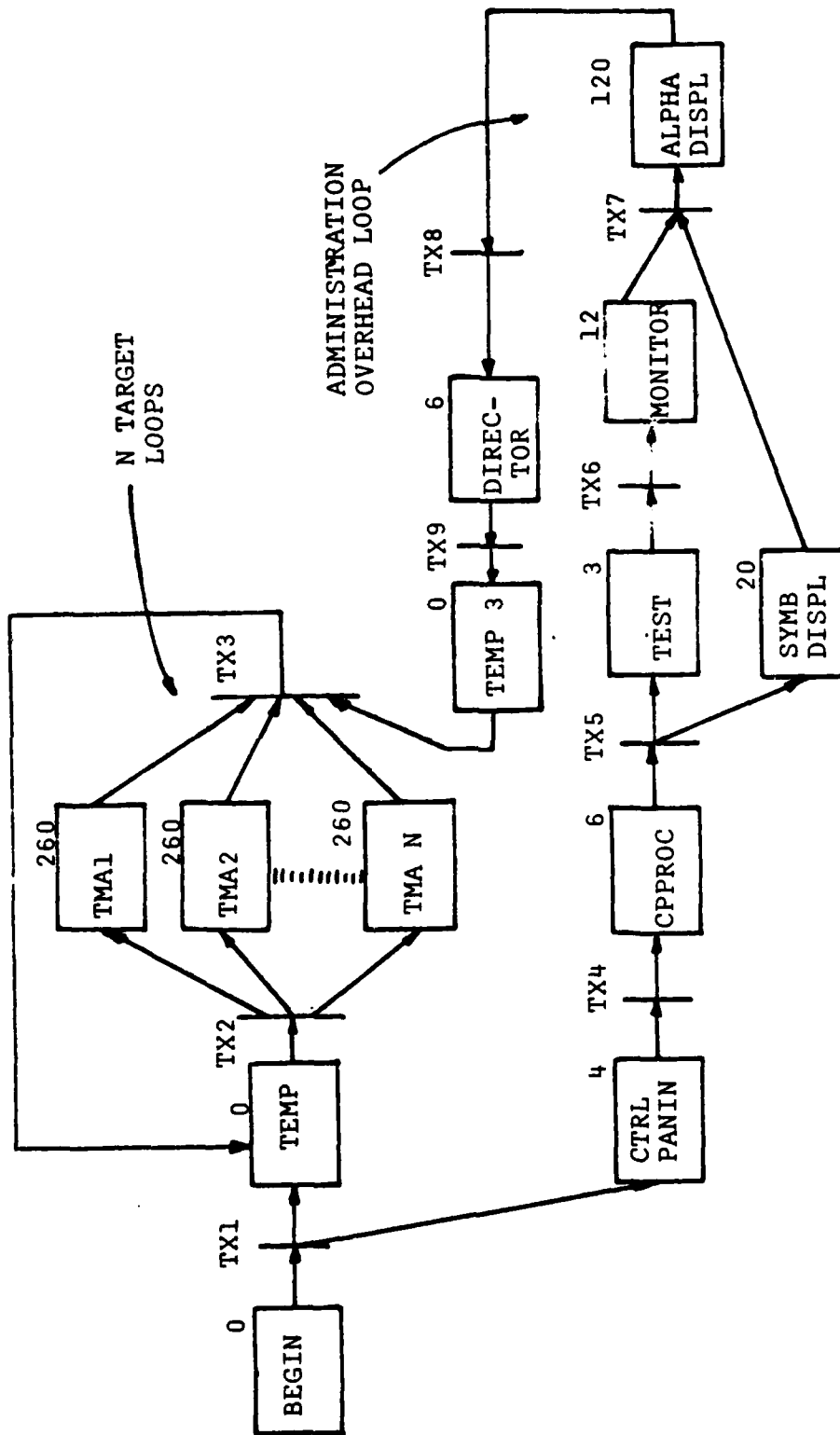
SEAFIRE COMPUTER TIMING ESTIMATE
FOR AUTOMATIC TRACK MODE

TABLE 1



OUTLINE OF SEAFIRE HARDWARE REPRESENTATION

Figure 14



SEAFIRE SOFTWARE (AUTOMATIC TRACK MODE)

Figure 15

C. PRESENTATION OF RESULTS

Initial results showed that the performance of the SEAFIRE system under development was approximately 30% below design goals. Detailed analysis of the performance prediction showed significant problems in the methodology used to predict the performance. The multiple cyclic loop structures that are present in the SEAFIRE hardware/software representation present deadlock like competition for the hardware resources. Several times during processing it was evident that one cyclic loop would gain "control" of the hardware to the exclusion of all other processes; this loop consuming all hardware resources available. In a real time system, an Executive routine would drive the interrupts based on a clock. This reflects a problem of using the P4 system as it currently stands to model real-time (interrupt driven) systems.

Subsequent experiments indicated that the computer program flow could be manipulated in a cyclic (synchronous) manner to approximate an interrupt driven environment. Although the results closely replicate the contractor's predictions concerning the timing estimates required for program execution, a true representation of the real time fire control program was not created.

It would also have been preferred if the processing of the embedded microprocessors could have been included; although it would have been rather simple to implement at

this level, the results would not have been significantly altered. A lower level of detail (ie, software running on the actual hardware) would provide the expected output of a faster, more efficient fire control solution which is less dependent on the centralized processor concept.

The final timing estimates indicate that the proposed software design will meet the Navy's processing time requirements and have the capacity of expansion to include additional functions as system development proceeds.

After further analysis, the structure of the programs were modified so that a maximum number of target loops could be accomplished without consideration for the administrative functions.

VII. CONCLUSIONS AND RECOMMENDATIONS

The U.S. Navy and DOD are not doing an adequate job of specifying and developing the criteria to be used as standards for computer system evaluation and the prediction of their performance. The tools are available, but yet past methods are implemented without considering innovative industrial ideas. Only token amounts of funding are expended where the payoff is the greatest; in early conceptual development phases.

Despite the advances in these areas, the question also arises as to whether the DOD can exploit these ideas with the support of industry. A number of approaches have been actively pursued over the last few years, however, there is not currently a firm direction in employing these new techniques in industry or DOD.

A new dimension for the analysis of computer architectures has emerged. These methods can enhance the performance of computer systems and create an iterative atmosphere between industry and DOD which is required for future systems development.

The methodology presented in this thesis should be considered as a partial effort in this direction. The approach is theoretically sound but its implementation requires a more thorough analysis with appropriate tailoring for its implementation. The rapid development of computer technology dictates that the DOD be able to better cope with

this pace. Further research and development into the causes and the nature of the problem of simulating an interrupt driven real-time combat system is highly recommended. Section V mentioned that the P4 system is directly analogous to data flow computing models. If the problem is inherent in the P4 system, it may very well be inherent in data flow computing models, which will inhibit their use in this type of analysis. For this reason, it makes further research in this area imperative prior to other implementations. It is recommended that this and other methodologies be explored further and hopefully utilized in the near future.

APPENDIX A

PROGRAM LISTING

petrinet.ftn Page 1 Tue Nov 27 06:18:55 1979

```

1 C
2 C
3 C      PROGRAM PETRI-NET REQUESTOR/SERVER SIMULATOR
4 C
5 C
6 C
7 C      THIS PROGRAM IS THE REQUESTOR/SERVER INTERFACE
8 C      MODEL FOR IMPLEMENTATION IN THE P5 NETWORK.
9 C
10 C
11 C      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
12 C      INXTF,KTIME,NEV,NTR
13 C      COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
14 C
15 C      DO 0005 I=1,255
16 C      NET(I,1)=0
17 C      NTRNS(I,1)=0
18 C      JNET(I,1)=0
19 C      JTRNS(I,1)=0
20 C      0005 CONTINUE
21 C      CALL INIT
22 C
23 C      0010 CONTINUE
24 C      CALL SCANR(5)
25 C      IF(MATCHS(1,5HBEGIN,5).EQ.1) GO TO 0020
26 C      IF(MATCHS(1,3HEND,3).EQ.1) GO TO 0040
27 C      CALL ERRRRR(1,7H MAIN,0,0)
28 C      GO TO 0040
29 C
30 C      0020 CONTINUE
31 C      IF(MATCHS(2,7HMACHINE,7).EQ.1) CALL STATIC(NET,
32 C      INTRNS,NEV,NTR)
33 C      IF(MATCHS(2,7HDYNAMIC,7).EQ.1) CALL DYNAMO
34 C      IF(MATCHS(2,7HPROGRAM,7).EQ.1) CALL PGMNET
35 C      IF(MATCHS(1,3HEND,3).EQ.1) GO TO 0010
36 C      IF(MATCHS(1,7HMACHINE,7).NE.1 .AND. MATCHS(1,
37 C      17HDYNAMIC,7).NE.1
38 C      1 .AND. MATCHS(1,7HPROGRAM,7).NE.1) GO TO 0030
39 C      GO TO 0010
40 C
41 C      0030 CONTINUE
42 C      CALL ERRRRR(1,7H MAIN,0,0)
43 C
44 C      0040 CUNTINUE
45 C      CALL DUMP(NET,NTRNS,NFRE,NXTF,KTIME,NEV,NTR)
46 C      CALL DUMP(JNET,JTRNS,NFRE,NXTF,KTIME,JEV,JTR)
47 C      CALL IDUMP
48 C      CALL EXIT
49 C      END
50 C
51 C
52 C
53 C      SUBROUTINE INIT
54 C      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
55 C      INXTF,KTIME,NEV,NTR
56 C      COMMON/CTRLR/ IMODE,ICQ(250)
57 C      COMMON/RAND/ RANDP
58 C
59 C      START-UP..... DEFINE DEVICES FOR OUTPUT,
60 C      GET START-UP CTRLR MODE AND RANDOM MODE PROBABILITY
        FROM TTY. CREATE A FICTICIOUS NODE 'RANDOM'.

```

```

61 C
62 C
63 OPEN(UNIT=7,NAME='NETOUT.LST',TYPE='NEW')
64 OPEN(UNIT=5,NAME='DPO:NETINP.INP;1',TYPE='OLD',
65 1 READONLY)
66 0100 FORMAT(/,' NET-SIM VER. 24',//)
67 WRITE(7,0100)
68 0120 FORMAT(5X,'*PROGRAM START-UP MODE= ',I5,
69 1 ' RAND. PRD.= ',F6.3,//)
70 CALL SCANR(5)
71 CALL XINTGR(1,IMODE)
72 CALL XFLOAT(2,RANDP)
73 WRITE(7,0120) IMODE,RANDP
74 C CREATE THE PHONY NODE...AS NUMBER 255
75 CALL NAMEIT(NET(255,1),6HRANDOM,6)
76 RETURN
77 END
78 C
79 C
80 SUBROUTINE STATIC(KNET,KTRNS,KEV,KTR)
81 COMMON/DMP/ NFREE1
82 COMMON/SCAN/ NUMB,IWORD(15,10)
83 COMMON/VEI/ NET(255,4),NTRNS(255,3),NFRE(999,2),
84 1 NXTF,KTIME,NEV,NTR
85 DIMENSION KNET(255,4),KTRNS(255,3)
86 BYTE TEMP
87 DIMENSION TEMP(10)
88 C
89 C C C C C N E T D E F I N I T I O N S
90 C
91 C C C C C NET(N,1) = POINTER TO NAMES ARRAY WHICH HOLDS NAME.
92 C C C C C NET(N,2) = MARKER (0 --) UNMARKED)
93 C C C C C NET(N,3) = RESOURCE REQUIREMENTS (TYPE)
94 C C C C C NET(N,4) = OUTPUT FLAG FOR EXECUTION PRINT.
95 C C C C C NEV IS NUMBER OF EVENTS
96 C
97 C C C C C N T R N S D E F I N I T I O N S
98 C
99 C C C C C NTRNS(N,1) = NAME OF TRANSITION
100 C C C C C NTRNS(N,2) = POINTER TO TRANSITION INPUTS IN FREE
101 C C C C C SPACE
102 C C C C C NTRNS(N,3) = POINTER TO TRANSITION OUTPUTS
103 C C C C C NTR IS NUMBER OF TRANSITIONS
104 C
105 C C C C C N F R E E D E F I N I T I O N S
106 C
107 C C C C C NFRE(N,1) POINTER TO AN EVENT IN NET
108 C C C C C NFRE(N,2) POINTER TO NEXT ENTRY IN NFRE OR NIL (0).
109 C C C C C NXTF IS POINTER TO NEXT FREE SPACE.
110 C
111 C C C C C
112 C
113 C BEGIN STATIC LOOP TABLE BUILDING
114 0200 CONTINUE
115 CALL SCANR(5)
116 IF(MATCHS(1,3HEND,3).EQ.1) GO TO 0291
117 IF(MATCHS(1,7HDECLARE,7).NE.1) GO TO 0290
118 IF(MATCHS(3,5HEVENT,5).EQ.1) GO TO 0210
119 IF(MATCHS(3,10HTRANSITION,10).EQ.1) GO TO 0240
120 GO TO 0290

```

```

121 C
122 0210 CONTINUE
123     CALL JWORD(2,TEMP)
124     NEXT=IFINDN(TEMP,KNET)
125     IF(NEXT.NE.0) GO TO 0220
126     KEV=KEV+1
127     IF(KEV.GT.255) GO TO 0230
128     CALL NAMINP(KNET(KEV,1),2)
129     CALL XINTGR(NUMB,NEXT)
130     KNET(KEV,3)=NEXT
131     GO TO 0200
132 0220 CONTINUE
133     CALL ERRRRR(3,8H STATIC,0,0)
134     GO TO 0200
135 0230 CONTINUE
136     CALL ERRRRR(2,8H STATIC,0,0)
137 C
138 0240 CONTINUE
139     CALL JWORD(2,TEMP)
140     N1=IFINDT(TEMP,KTRNS,KTR)
141     IF(N1.EQ.KTR) GO TO 0250
142     CALL ERRRRR(3,8H STATIC,0,0)
143     GO TO 0200
144 0250 CONTINUE
145     CALL SCANR(5)
146     IF(MATCHS(1,3HEND,3).EQ.1) GO TO 0200
147     IF(MATCHS(1,5HINPUT,5).EQ.1) GO TO 0260
148     IF(MATCHS(1,6HOUTPUT,6).EQ.1) GO TO 0280
149     CALL ERRRRR(5,8H STATIC,0,0)
150     GO TO 0250
151 0260 CONTINUE
152     CALL JWORD(NUMB,TEMP)
153     N2=IFINDN(TEMP,KNET)
154     IF(N2.NE.0) GO TO 0270
155     CALL ERRRRR(4,8H STATIC,1WORD(NUMB,1),0)
156     GO TO 0250
157 0270 CONTINUE
158     CALL SETFRE(KTRNS(N1,2),N2)
159     GO TO 0250
160 0280 CONTINUE
161     CALL JWORD(NUMB,TEMP)
162     N2=IFINDN(TEMP,KNET)
163     IF(N2.EQ.0) GO TO 0260
164     CALL SETFRE(KTRNS(N1,3),N2)
165     GO TO 0250
166 C
167 0290 CONTINUE
168     CALL ERRRRR(5,8H STATIC,0,0)
169     GO TO 0200
170 C
171 0291 CONTINUE
172     IF(IMFRST.NE.0) GO TO 0292
173     IMFRST=1
174     NFREE1=NXTF
175 0292 CONTINUE
176     CALL LISTX(KNET,KTRNS,KFRE,NXTF,KTIME,KEV,KTR)
177     RETURN
178     END
179 C
180 C

```

```

181 SUBROUTINE JWORD(NUMBER,STRING)
182 BYTE IWORD
183 COMMON/SCAN/ NUMB,IWORD(15,10)
184 BYTE STRING
185 DIMENSION STRING(10)
186 DO 0295 I=1,10
187 0295 STRING(I)=IWORD(NUMBER,I)
188 D9000 FORMAT(' JWORD:NUMBER,STRING:',I4,2X,10A1)
189 D WRITE(7,9000) NUMBER,(STRING(I),I=1,10)
190 RETURN
191 END
192 C
193 C
194 SUPROUTINE SFTFRE(IPOINT,IVALUE)
195 COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
196 INXTF,KTIME,NEV,NTR
197 C
198 C FOR TRANSITION INPUTS OR OUTPUTS, SET UP AND
199 C ENTER VALUE IN THE CHAIN POINTED TO BY IPOINT.
200 C
201 IF(IPOINT.EQ.0) GO TO 0310
202 NEXT=IPOINT
203 C
204 0300 CONTINUE
205 NEXOLD=NEXT
206 NEXT=NFRE(NEXT,2)
207 IF(NEXT.NE.0) GO TO 0300
208 C END OF CHAIN
209 NXTF=NXTF+1
210 IF(NXTF.GT.999) CALL ERRRRR(2,8H SETFRE,0,0)
211 NFRE(NEXOLD,2)=NXTF
212 NFRE(NXTF,1)=IVALUE
213 RETURN
214 0310 CONTINUE
215 NXTF=NXTF+1
216 IPOINT=NXTF
217 NFRE(NXTF,1)=IVALUE
218 RETURN
219 END
220 C
221 C
222 FUNCTION IFINDT(NAME,NTRNS,NTR)
223 BYTE NAME,TEMP
224 DIMENSION NAME(10),TEMP(10)
225 DIMENSION NTRNS(255,3)
226 C
227 C FIND THE TRANSITION 'NAME' IN THE TABLE !
228 C RETURN NUMBER
229 C
230 D9000 FORMAT(' IFINDT: NAME,NTR',2X,10A1,2X,I4)
231 D WRITE(7,9000) (NAME(I),I=1,10),NTR
232 DO 0400 I=1,255
233 IFINDI=I
234 IF(NTRNS(I,1).NE.0) CALL GETNAM(NTRNS(I,1),TEMP)
235 IF(MATCHC(NAME,TEMP,10).EQ.1) RETURN
236 0400 CONTINUE
237 C DIDN'T FIND IT, SO CREATE IT.
238 C
239 NTR=NTR+1
240 CALL NAMEIT(NTRNS(NTR,1),NAME,10)

```

```

241      IFINDT=NTR
242      IF(NTR.LE.255) RETURN
243      CALL EPRRRR(2,8H IFINDT,0,0)
244      RETURN
245      END
246 C
247 C
248      FUNCTION IFINDN(NAME,NET)
249      BYTE NAME,TEMP
250      DIMENSION NAME(10),TEMP(10)
251      DIMENSION NET(255,4)
252 C
253 C      FIND THE NAME IN THE TABLE
254 C      RETURN 0 IF NOT THERE
255 C
256 D9000 FORMAT(' IFINDN:NAME ',10A1)
257 D      WRITE(7,9000) (NAME(I),I=1,10)
258      IFINDN=0
259      DO 0500 I=1,255
260      IF(NET(I,1).NE.0) CALL GETNAM(NET(I,1),TEMP)
261      IF(MATCHC(NAME,TEMP,10).EQ.1) GO TO 0510
262      0500 CONTINUE
263      RETURN
264      0510 CONTINUE
265      IFINDN=I
266      RETURN
267      END
268 C
269 C
270      FUNCTION MATCHC(STRNG1,STRNG2,KOUNT)
271      BYTE STRNG1,STRNG2
272      DIMENSION STRNG1(10),STRNG2(10)
273      MATCHC=0
274 D9000 FORMAT(' MATCHC: ',2(10A1,2X))
275 D      WRITE(7,9000) (STRNG1(I),I=1,10),(STRNG2(I),I=1,10)
276      DO 0550 I=1,KOUNT
277      IF(STRNG1(I).NE.STRNG2(I)) RETURN
278      0550 CONTINUE
279      MATCHC=1
280      RETURN
281      END
282 C
283 C
284      SUBROUTINE LISTX(NET,NTRNS,NFRE,NXTF,KTIME,NEV,NTR)
285      BYTE TEMP
286      DIMENSION TEMP(10)
287      DIMENSION NET(255,4),NTRNS(255,3),NFRE(999,2)
288 C
289 C      AFTER STATIC HARDWARE NET IS IN, DO AN ANALYSIS,
290 C      FIRST PRINT SYMBOL TABLE DUMPS, THEN DO STATIC
291 C      CONFLICT ANALYSIS OF THE NETWORK
292 C
293      0600 FORMAT('/',20X,'-----STATIC-STRUCTURE-ANALYSIS-----',
294      1//)
295      WRITE(7,0600)
296      CALL DUMP(NET,NTRNS,NFRE,NXTF,KTIME,NEV,NTR)
297      WRITE(7,0600)
298      RETURN
299      END
300 C

```

```

301 C
302 C
303 SUBROUTINE DUMP(NET,NTRNS,NFRE,NXTF,KTIME,NEV,NTR)
304 BYTE TEMP
305 DIMENSION TEMP(10)
306 DIMENSION NET(255,4),NTRNS(255,3),NFRE(999,2)
307 C
308 0700 FORMAT(/,20X,'NETWORK ARRAY DUMP TIME = ',IS,
309 1' EVENTS ',/,3X,' --NAME-- --MARKER-- --TYPE',
310 2' --OUTPUT- ',//)
311 C
312 0710 FORMAT(X,10A1,3I10)
313 WRITE(7,0700) KTIME
314 DO 0720 I=1,NEV
315 CALL GETNAM(NET(I,1),TEMP)
316 WRITE(7,0710) (TEMP(IJK),IJK=1,10),(NET(I,J),J=2,4)
317 0720 CONTINUE
318 0730 FORMAT(/,20X,' TRANSITION TABLE AND FREE SPACE DU'
319 1'P ',/,3X,' --NAME-- INPUT PTR. OUTPUT PTR ',//)
320 0740 FORMAT(X,10A1,2I10)
321 WRITE(7,0730)
322 DO 0750 I=1,NTR
323 CALL GETNAM(NTRNS(I,1),TEMP)
324 WRITE(7,0740) (TEMP(IJK),IJK=1,10),(NTRNS(I,J),
325 1J=2,3)
326 RETURN
327 END
328 C
329 C
330 SUBROUTINE TDUMP
331 COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
332 1NXTF,KTIME,NEV,NTR
333 COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
334 COMMON/DMP/ NFREE1
335 COMMON/NAME1/NAMES(203,10),NXTNAM
336 BYTE NAMES
337 BYTE TEMP1,TEMP2
338 DIMENSION TEMP1(10),TEMP2(10)
339 0800 FORMAT(X,'-----')
340 0810 FORMAT(/,20X,'FREESPACE ',/,3X,
341 1' -NUMBER- -EVENT- --NEXT--',//)
342 0820 FORMAT(X,1I10,2X,10A1,1I10)
343 WRITE(7,0810)
344 DO 0830 I=1,NXTF
345 CALL GETNAM(NET(NFRE(I,1),1),TEMP1)
346 CALL GETNAM(JNET(NFRE(I,1),1),TEMP2)
347 IF(I.LE.NFREE1) WRITE(7,0820) I,(TEMP1(J),J=1,10),
348 1NFRE(I,2)
349 IF(I.EQ.NFREE1) WRITE(7,0800)
350 IF(I.GT.NFREE1) WRITE(7,0820) I,(TEMP2(J),J=1,10),
351 1NFRE(I,2)
352 0830 CONTINUE
353 0840 FORMAT(5X,'NAMES NXTNAM=',I4)
354 0850 FORMAT(5X,10A1)
355 WRITE(7,0840) NXTNAM
356 DO 0860 I=1,NXTNAM-1
357 0860 WRITE(7,0850) (NAMES(I,J),J=1,10)
358 RETURN
359 END
360 C

```

```

361      SUBROUTINE DYNAMO
362      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
363      1NXTF,KTIME,NEV,NTR
364      COMMON/DYN/ LOOK1(255),LOOK2(255)
365      COMMON/SCAN/ NUMB,IWORD(15,10)
366      C
367      C      INTERPRET DYNAMIC COMMANDS AND RUN SIMULATION
368      C
369      C
370      IF(KTIME.EQ.0) KTIME=1
371      C
372      0900 CONTINUE
373      CALL SCANR(5)
374      IF(MATCHS(1,3HEND,3).EQ.1) RETURN
375      IF(MATCHS(1,4HMARK,4).EQ.1) GO TO 0920
376      IF(MATCHS(1,6HOUTPUT,6).EQ.1) GO TO 0930
377      IF(MATCHS(1,7HEXECUTE,7).EQ.1) GO TO 0940
378      0910 CONTINUE
379      CALL ERRRRR(5,8H DYNAMO,0,0)
380      GO TO 0900
381      C
382      0920 CALL MARKET
383      GO TO 0900
384      0930 CALL SETOUT
385      GO TO 0900
386      0940 CALL EXEC
387      GO TO 0900
388      END
389      C
390      C
391      SUBROUTINE MARKET
392      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
393      1NXTF,KTIME,NEV,NTR
394      COMMON/SCAN/ NUMR,IWORD(15,10)
395      C
396      BYTE TEMP
397      DIMENSION TEMP(10)
398      C
399      C      MARK AN EVENT WITH THE DESIRED VALUE
400      C
401      CALL JWORD(2,TEMP)
402      N1=IFINDN(TEMP,NF1)
403      IF(N1.NE.0) GO TO 1000
404      CALL ERRRRR(4,8H MARKET,IWORD(2,1),0)
405      RETURN
406      1000 CONTINUE
407      CALL XINTGR(NUMB,IVALUE)
408      NET(N1,2)=IVALUE
409      RETURN
410      END
411      C
412      C
413      SUBROUTINE SETOUT
414      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
415      1NXTF,KTIME,NEV,NTR
416      BYTE IWORD
417      COMMON/SCAN/ NUMB,IWORD(15,10)
418      BYTE TEMP
419      DIMENSION TEMP(10)
420      C

```

```

421 C      SET OUTPUT FLAG ON DESIGNATED EVENT
422 C
423      CALL JWORD(NUMB,TEMP)
424      N1=IFINDN(TEMP,NET)
425      IF(N1.NE.0) GO TO 1100
426      CALL ERRRRR(4,8H  SETOUT,IWORD(NUMB,1),0)
427      RETURN
428 1100 CONTINUE
429      NET(N1,4)=1
430      RETURN
431      END
432 C
433 C
434      SUBROUTINE EXEQ
435      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
436 1NXTF,KTIME,NEV,NTR
437      COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
438 C
439 C      EXECUTE THE REQUESTOR/SERVER NETWORK
440 C
441      CALL XINTGR(2,ITIME)
442      KLIMIT=KTIME+ITIME
443 C
444 1200 CONTINUE
445      IF(KTIME.GE.KLIMIT) RETURN
446      CALL EXEQ1(JNET,JTRNS,NFRE,NXTF,JEV,JTR,1,IGO,
447 1KTIME)
448      IF(IGO.EQ.1) GO TO 1200
449      IF(KSOFT(IGO).EQ.0) RETURN
450      CALL HWGO
451      CALL EXEQ1(NET,NTRNS,NFRE,NXTF,NEV,NTR,0,IGO,
452 1KTIME)
453      GO TO 1200
454      END
455 C
456 C
457      SUBROUTINE EXEQ1(KNET,ITRN,NFRE,KTF,IEV,ITR,IFUNC,
458 1IFIRE,KTIME)
459      BYTE TEMP
460      DIMENSION TEMP(10)
461      COMMON/DYN/ LOOK1(255),LOOK2(255)
462      DIMENSION KNET(255,4),ITRN(255,3),NFRE(999,2)
463      DIMENSION KPRINT(255)
464 C
465 C      EXECUTE THE SPECIFIED NETWORK FOR ONE CLICK
466 C      IFUNC = 1 --) SOFTWARE NET,
467 C      IFUNC = 0 --) HARDWARE NET.....
468 C
469 C      IFIRE = 0 --) NOTHING FIRED THIS TIME
470 C      IFIRE = 1 --) ONE OR MORE TRANSITIONS FIRED.
471 C
472 C
473 C
474      IFIRE=0
475 1300 FORMAT(X,'TIME=',I4,'\\')
476      IF(IFUNC.EQ.1) GO TO 1310
477      WRITE(7,1300) KTIME
478      CALL HWRAND
479 C
480 1310 CONTINUE

```

```

481      CALL MARKER(LOOK1,KNET,ITRN,NFRE,KTF,IEV,ITR)
482      DO 1320 I=1,IEV
483      KPRINT(I)=0
484 1320 CONTINUE
485 C
486      DO 1390 I=1,ITR
487 C      CHECK WHICH TRANSITIONS ARE READY TO FIRE !
488 C      EXECUTE
489      CALL MARKER(LOOK2,KNET,ITRN,NFRE,KTF,IEV,ITR)
490      IF(LOOK1(I).EQ.0 .AND. LOOK2(I).EQ.1) GO TO 1390
491      IF(LOOK1(I)+LOOK2(I) .EQ. 0) GO TO 1390
492      IF(LOOK1(I).EQ.LOOK2(I)) GO TO 1330
493      CALL EPRRRR(6,7H EXEC,ITRN(I,1),0)
494      GO TO 1390
495 C
496 1330 CONTINUE
497 C      FIRING A TRANSITION - UNMARK INPUTS, MARK OUTPUTS
498 C
499      IFIRE=1
500      NEXT=ITRN(I,2)
501 1340 CONTINUE
502      IF(NEXT.EQ.0) GO TO 1350
503      NEXOLD=NEXT
504      NEXT=NFRE(NEXT,2)
505 C      CHECK IF NO TOKENS..MAYBE USED ON THIS TRANSITION
506      IF(KNET(NFRE(NEXOLD,1),2).EQ.0) GO TO 1370
507 C      IF NO MORE TOKENS, HAVE TO BACK OUT OF TRANSITION
508      KNET(NFRE(NEXOLD,1),2)=KNET(NFRE(NEXOLD,1),2)-1
509      GO TO 1340
510 C
511 1350 CONTINUE
512 C
513      NEXT=ITRN(I,3)
514 1360 CONTINUE
515      IF(NEXT.EQ.0) GO TO 1390
516      NEXOLD=NEXT
517      NEXT=NFRE(NEXT,2)
518      KNET(NFRE(NEXOLD,1),2)=KNET(NFRE(NEXOLD,1),2)+1
519      IF(IFUNC.EQ.1) CALL RSIN(KNET(NFRE(NEXOLD,1),1))
520      IF(IFUNC.EQ.0) CALL PSOUT(NFRE(NEXOLD,1))
521      KPRINT(NFRE(NEXOLD,1))=1
522      GO TO 1360
523 C
524 1370 CONTINUE
525 C      REPLACE SOME TOKENS.. THIS TRANSITION IS WIERD...
526      ISTOPD=NEXOLD
527      NEXT=ITRN(I,2)
528 1380 CONTINUE
529      IF(NEXT.EQ.ISTOPD) GO TO 1390
530      NEXOLD=NEXT
531      NEXT=NFRE(NEXT,2)
532      KNET(NFRE(NEXOLD,1),2)=KNET(NFRE(NEXOLD,1),2)+1
533      GO TO 1380
534 C      END OF BAK-UP PROCESS
535 C
536 1390 CONTINUE
537 C
538      DO 1392 J=1,IEV
539 1391 FORMAT(5X,'*****EVENT ',10A1,' MARKED WITH ',1I10,
540 1'*****')

```

```

541      CALL GETNAM(KNET(J,1),TEMP)
542      IF(KPRINT(J).EQ.1 .AND. KNET(J,4).NE.0)
543      1 WRITE(7,1391) (TEMP(K),K=1,10),KNET(J,2)
544 1392 CONTINUE
545 C
546      IF(IFUNC.EQ.0) KTIME=KTIME+1
547      RETURN
548      END
549 C
550 C
551      SUBROUTINE HWGO
552      COMMON/NET/NET(255,4),NTRNS(255,3),NFRE(999,2),
553 1 NXTF,KTIME,NEV,NTR
554      COMMON/CTRLR/ IMODE,ICQ(250),ICQPTR
555 C
556      MARK AS MANY HARDWARE UNITS AS DESIRED
557      (ACCORDING TO OUTSTANDING SW REQUESTS)
558      NOT TO EXCEED THE LIMIT OF 'IMODE'.
559 C
560      THEN SHIFT UP THE QUEUE, ICQ, AND RESET ICQPTR
561 C
562      IF ICQPTR = 0 NOTHING TO DO, SO RETURN...
563      IF(ICQPTR.EQ.0) RETURN
564 C
565      DO 1400 I=1,ICQPTR
566      NET(ICQ(I),2)=NET(ICQ(I),2)+1
567      J=I
568      IF(I.EQ.IMODE) GO TO 1410
569 1400 CONTINUE
570 C
571 1410 CONTINUE
572 C      REPACK QUEUE
573      DO 1420 I=1,ICQPTR-J
574      ICQ(I)=ICQ(J+I)
575 1420 CONTINUE
576      ICQPTR=ICQPTR-J
577 C
578      RETURN
579      END
580 C
581 C
582      SUBROUTINE HWRAND
583      COMMON/NFT/ NET(255,4)
584      COMMON/RAND/ RANDP
585 C
586      CHECK RANDOM EVENT AND MARK IT PROBABILISTICALLY.
587 C
588      PROB=LAN(I1,I2)
589      IF(PROB.LT.RANDP) NET(255,2)=NET(255,2)+1
590      RETURN
591      END
592 C
593 C
594      SUBROUTINE MARKER(KRAY,NET,NTRNS,NFRE,NXTF,NEV,NTR)
595      DIMENSION NET(255,4),NTRNS(255,3),NFRE(999,2)
596      DIMENSION KRAY(255)
597 C
598      DO 1500 I=1,NTR
599      KRAY(I)=0
600 1500 CONTINUE

```

```

601 C
602 DO 1530 I=1,NTR
603 K1=0
604 K2=0
605 C
606 NEXT=NTRNS(1,2)
607 1510 CONTINUE
608 IF(NEXT.EQ.0) GO TO 1520
609 NEXOLD=NEXT
610 NEXT=NFRE(NEXT,2)
611 K2=K2+1
612 IF(NET(NFRE(NEXOLD,1),2).NE.0) K1=K1+1
613 GO TO 1510
614 C
615 1520 CONTINUE
616 IF(K1.EQ.K2) KRAY(I)=1
617 C
618 1530 CONTINUE
619 C
620 RETURN
621 END
622 C
623 C
624 SURROUTINE PGMNET
625 COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
626 C
627 THE SOFTWARE NET BUILD ROUTINE...
628 FIRST RENAME
629 C
630 THEN CONTINUE REBUILDING THE NET
631 CALL STATIC(JNET,JTRNS,JEV,JTR)
632 C NOW LOOK FOR THE BEGIN STATEMENT...
633 I=TFJNDN(10HBEGIN,JNET)
634 IF(I.NE.0) GO TO 1600
635 CALL ERRRRR(R,6HPGMNET,0)
636 C
637 1600 CONTINUE
638 C MARK THE SOFTWARE BEGIN EVENT
639 JNET(I,2)=1
640 RETURN
641 END
642 C
643 C
644 SURROUTINE RSIN(NAME)
645 RYTE TEMP
646 DIMENSION TEMP(10)
647 COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
648 INXTF,KTIME,NEV,NTR
649 COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
650 COMMON/RSTABL/ IRS(90,3)
651 C
652 THE REQUESTOR/SERVER INTERFACE TABLE
653 C
654 IRS(M,1) --) POINTER TO NAME OF SOFTWARE EVENT
655 REQUESTING SERVICE
656 IRS(M,2) --) START TIME OF REQUEST.
657 IRS(M,3) --) HARDWARE UNIT NUMBER REQUIRED.
658 C
659 COMMON/CTRLR/ IMODE,ICQ(250),ICQPTR
660 C

```

```

661 C      COMMON BLOCK CTRLR CONTAINS INFO REGARDING
662 C      HARDWARE REQUESTS, AND EXISTS SO THAT THE # OF
663 C      REQUESTS PER MINOR CYCLE CAN LIMIT AS DESIRED.
664 C      REQUESTS STACKED IN ICQ, THE QUEUE, AND SERVICED
665 C      AS POSSIBLE, A MAX OF IMODE EVERY MINOR CYCLE.
666 C
667 C      ENTER NAME IN THE TABLE, PLACE HW IN QUEUE, THEN
668 C      REMOVE SOFT TOKEN. (DO NOTHING IF SOFTEVENT
669 C      IS TYPE 0 WHICH IS A NULL EVENT FOR PRECEDENCE)
670 C
671 C      CALL GETNAM(NAME,TEMP)
672 C      NUMBER=JNET(IFINDN(TEMP,JNET),3)
673 C      IF(NUMBER.EQ.0) RETURN
674 C
675 C      DO 1700 I=1,90
676 C      IF(IRS(I,1).EQ.0) GO TO 1710
677 1700 CONTINUE
678 C      CALL FPRRRR(11,4HRSIN,TEMP,0)
679 C
680 C      1710 CONTINUE
681 C      DO 1720 J=1,NEV
682 C      IF(NET(J,3).EQ.NUMBER) GO TO 1730
683 1720 CONTINUE
684 C      CALL ERRRRR(9,4HRSIN,NAME,0)
685 1730 CONTINUE
686 C      ICQPTR=ICQPTR+1
687 C      IF(ICQPTR.GT.250) CALL ERRRRR(11,10HRSIN (ICQ),0,0)
688 C      ICQ(ICQPTR)=J
689 C      J=IFINDN(TEMP,JNET)
690 C      JNET(J,2)=0
691 C      IRS(J,1)=NAME
692 C      IRS(J,2)=KTIME
693 C      IRS(J,3)=NUMBER
694 1740 FORMAT(5X,'*PROGRAM EVENT ',10A1,' REQUESTS HW',
695 C      1' SVCS ',1I5)
696 C      WRITE(7,1740) (TEMP(K),K=1,10),KTIME
697 C      RETURN
698 C      END
699 C
700 C
701 C      SUBROUTINE RSOUT(NUMBER)
702 C      RYTE TEMP
703 C      DIMENSION TEMP(10)
704 C      COMMON/NET/ NET(255,4),NTRNS(255,3),NFRE(999,2),
705 C      INXTF,KTIME,NEV,NTR
706 C      COMMON/SOFT/ JNET(255,4),JTRNS(255,3),JEV,JTR
707 C      COMMON/RSTABL/ IRS(90,3)
708 C
709 C      NET TRANSITION NUMBER (HARDWARE) HAS COMPLETED,
710 C      SEE IF ITS TYPE IS .LT. 0 (A UNIT FINISH EVENT),
711 C      AND IF SO, SEE IF A SOFTWARE EVENT WAS EXECUTING.
712 C      IF SO, ON A FIFO BASIS, COMPLETE THE SOFTEVENT,
713 C      PRINT A MESSAGE, REPLACE THE TOKEN IN THE SOFTNET
714 C      AND CONTINUE.
715 C
716 C      IF(NET(NUMBER,3).GE.0) RETURN
717 C
718 1800 FORMAT(5X,'*PROGRAM EVENT ',10A1,' COMPLETES ',1I5)
719 C      J=0
720 C      K=10000

```

AD-A081 607

NAVAL POSTGRADUATE SCHOOL MONTEREY CA
COMPUTER ARCHITECTURE PERFORMANCE PREDICTION FOR NAVAL FIRE CON--ETC(U)
DEC 79 D M STOWERS
NPS52-79-006

F/G 19/5

UNCLASSIFIED

NL

2 - 2

AD-A081 607



END

DATE

FILED

4 80

DTIC

```

721 C
722 DO 1810 I=1,90
723 IF(IRS(I,3).NE.(NET(NUMBER,3)*-1)) GO TO 1810
724 IF(IRS(I,2).GE.K) GO TO 1810
725 J=1
726 K=IRS(I,2)
727 1810 CONTINUE
728 C
729 IF(J.EQ.0) RETURN
730 FOUND IT
731 C MARK SOFTEVENT, UNMARK HARDEVENT
732 CALL GETNAM(IRS(J,1),TEMP)
733 K=IFINDN(TEMP,JNET)
734 CALL GETNAM(JNET(K,1),TEMP)
735 WRITE(7,1800) (TEMP(L),L=1,10),KTIME
736 JNET(K,2)=1
737 NET(NUMBER,2)=NET(NUMBER,2)-1
738 IRS(J,1)=0
739 IRS(J,2)=0
740 IRS(J,3)=0
741 RETURN
742 END
743 C
744 C
745 FUNCTION KSOFT(IDUMMY)
746 COMMON/RSTABL/ IRS(90,3)
747 C
748 COUNT NUMBER OF ACTIVE PROCESSES IN TABLE
749 C
750 J=0
751 DO 1900 I=1,90
752 IF(IRS(I,1).NE.0) J=J+1
753 1900 CONTINUE
754 KSOFT=J
755 RETURN
756 END
757 C
758 C
759 SUBROUTINE NAMEIT(IPOINT,STRING,KOUNT)
760 C
761 C
762 C -----
763 C ENTER NAME OF EVENT OR TRANSITION 'STRING'
764 C INTO THE GENERAL NAME TABLE. RETURN A
765 C POINTER TO ITS ENTRY 'IPOINT'.
766 C -----
767 C
768 BYTE NAMES,STRING,IBLANK
769 COMMON/NAME1/NAMES(203,10),NXTNAM
770 DIMENSION STRING(10)
771 DATA NXTNAM/1/
772 DATA IBLANK/1H /
773 C ERASE THE ENTRY
774 DO 1920 I=1,10
775 NAMES(NXTNAM,I)=IBLANK
776 COPY IN THE DATA
777 DO 1921 I=1,KOUNT
778 NAMES(NXTNAM,I)=STRING(I)
779 BUMP THE POINTER AND TEST FOR OVERFLOW.
780 IPOINT=NXTNAM
781 NXTNAM=NXTNAM+1

```

```

781 IF(NXTNAM.GT.203) GO TO 1922
782 D9000 FORMAT(' NAMEIT: IPOINT,STRING ',I4,2X,1A10)
783 D WRITE(7,9000) IPOINT,(STRING(I),I=1,10)
784 RETURN
785 C GOT A PROBLEM.....
786 1922 CONTINUE
787 1923 FORMAT(' **NAME TABLE OVERFLOW DETECTED BY',
788 1' NAMEIT. (FATAL)')
789 WRITE(6,1923)
790 CALL EXIT
791 END
792 C
793 C
794 SUBROUTINE GETNAM(IPOINT,STRING)
795 C
796 C GET THE NAME (10 BYTE STRING) POINTED TO BY
797 C "IPOINT" AND RETURN IT IN "STRING"
798 C
799 C
800 C BYTE NAMES,STRING
801 COMMON/NAMF1/NAMES(203,10),NXTNAM
802 DIMENSION STRING(10)
803 C
804 DO 1940 I=1,10
805 D9000 STRING(I)=NAMES(IPOINT,I)
806 D FORMAT(' GETNAM: IPOINT,STRING ',I4,2X,1A10)
807 D WRITE(7,9000) IPOINT,(STRING(I),I=1,10)
808 C
809 C WASN'T THAT SIMPLE?
810 C
811 RETURN
812 END
813 C
814 SUBROUTINE NAMINP(IPOINT,NUMB)
815 C
816 C NAMEIT FROM AN INPUT SCANNER WORD
817 C
818 C
819 C BYTE IWORD
820 COMMON/SCAN/NUMBER,IWORD(15,10)
821 C
822 C
823 C
824 C
825 1960 DO 1960 I=1,10
826 C TEMP(I)=IWORD(NUMB,I)
827 D9000 FORMAT(' NAMINP: NUMB,STRING ',I4,2X,1A10)
828 D WRITE(7,9000) NUMB,(TEMP(I),I=1,10)
829 CALL NAMEIT(IPOINT,TEMP,10)
830 C
831 RETURN
832 END
833 C
834 C
835 SUBROUTINE SCANR(LUNIT)
836 C
837 C
838 C
839 C
840 C

```

FREE FORMAT INPUT ROUTINE. READS AN 80 BYTE
RECORD FROM LOGICAL UNIT "LUNIT" AND STORES UP

```

841 C      TO 15 BLANK DELIMITED TOKENS (LEFT ADJUSTED)
842 CC     IN BYTE ARRAY "IWORD".
843 CC
844 CC     NUMERICAL VALUES CAN BE REFORMATTED FROM BYTE
845 CC     STRINGS INTO INTEGER AND FLOATING POINT VALUES
846 CC     THRU THE SUBROUTINES "XFLOAT" AND "XINTGR".
847 CC     -----
848 C
849 C      BYTE IWORD,ISC,IBLANK
850 C      COMMON/SCAN/NUMBER,IWORD(15,10)
851 C      BYTE NBUFFR
852 C      COMMON/SCAN1/NBUFFR(80)
853 C      DATA ISC/1H;/
854 C      DATA IBLANK/1H /
855 C      DATA NEOF/0/
856 C      DATA KLINE/0/
857 2001 KLINE=KLINE+1
858 2000 FORMAT(80A1)
859 C      BEGIN BY READING A LINE OF 80 BYTES...
860 C      READ(LUNIT,2000,END=2035,ERR=2035) (NBUFFR(I),
861 C      1 I=1,80)
862 2002 FORMAT(X,1I4,' - ',80A1)
863 C      WRITE(7,2002) KLINE,(NBUFFR(I),I=1,80)
864 C      SET POINTER TO FIRST CHARACTER IN THE BUFFER
865 C      IPOINT=1
866 C      NOW PROCESS THE FIRST 15 TOKENS DELIMTIED BY
867 C      EITHER A BLANK (OR MULTIPLE BLANKS) OR A
868 C      SEMICOLON.
869 C
870 C      DO 2025 NUMBER=1,15
871 C      IFLAG=0
872 C      SET IWORD(NUMBER,X)=IBLANK(SET WORD TO ALL BLANKS)
873 C      DO 2005 I=1,10
874 2005 IWORD(NUMBER,I)=IBLANK
875 C      START SCAN OF LINE FROM POINTER ON, FIND NON-BLANK
876 C      KOUNT=1
877 C      "KOUNT" KEEPS TRACK OF NO. OF CHAR. IN THE TOKEN
878 C      DO 2015 KPOINT=IPOINT,80
879 C      IF(NBUFFR(KPOINT).NE.IBLANK .AND. NBUFFR(KPOINT)
880 C      1.NE.ISC) GO TO 2010
881 C      IF(IFLAG.EQ.0) GO TO 2015
882 C      IF(IFLAG.EQ.1) GO TO 2020
883 C      GOT SOMETHING, SO PROCESS IT....
884 2010 CONTINUE
885 C      IFLAG=1
886 C      IWORD(NUMBER,KOUNT)=NBUFFR(KPOINT)
887 C      KOUNT=KOUNT+1
888 C      IF(KOUNT.GT.10) GO TO 2020
889 2015 CONTINUE
890 C
891 2020 CONTINUE
892 C      END OF TOKEN FOUND, RESET SOME POINTERS
893 C      IPOINT=KPOINT+1
894 C      IF(IPOINT.GT.80) GO TO 2030
895 C
896 2025 CONTINUE
897 C      END OF BASIC TOKEN GETTING LOOP
898 C
899 2030 NUMBER=NUMBER+1
900 C      IF(NUMBER.EQ.0) GO TO 2001

```

```

901      IF(MATCHS(1,'COMMENT:',8).EQ.1) GO TO 2001
902      RETURN
903 2035 CONTINUE
904 C      END OF FILE OR I/O ERROR DETECTED
905 2040 FORMAT(' EOF OR ERROR ON SCANNER INPUT FROM UNIT ',
906           1I3)
907      WRITE(7,2040) LUNIT
908      NEOF=NEOF+1
909      IF(NEOF.GE.3) CALL EXIT
910      NUMBER=0
911      RETURN
912      END
913 C
914 C
915      SUBROUTINE XFLOAT(NWORD,FWORD)
916 C
917 C      CONVERT THE ENTRY IN ARRAY IWORD (# NWORD) TO
918 C      STANDARD FLOATING POINT REPRESENTATION, RETURN
919 C      IT AS "FWORD".
920 C
921      BYTE IWORD
922      COMMON/SCAN/NUMBER,IWORD(15,10)
923 C
924      BYTE TSTRNG
925      DIMENSION TSTRNG(10)
926 C
927 C      COPY STRING (TO ALLOW COMPILER TO STORE THE ARRAY
928 C      HOWEVER IT WANTS TO)
929 C      DO 2045 I=1,10
930 2045 TSTRNG(I)=IWORD(NWORD,I)
931 C
932 C      DEFINE THE FORMATS:
933 2050 FORMAT(1F10.3)
934 C      DO IT!
935      DECODE(10,2050,TSTRNG) FWORD
936      RETURN
937      END
938 C
939 C
940      SUBROUTINE XINTGR(NWORD,IVALUE)
941 C
942 C      CONVERT THE ENTRY IN "IWORD" TO INTEGER
943 C      RETURN INTEGER "IVALUE"
944 C
945      BYTE IWORD
946      COMMON/SCAN/NUMBER,IWORD(15,10)
947      BYTE TSTRNG
948      DIMENSION TSTRNG(10)
949      BYTE IBLANK
950      DATA IBLANK/1H /
951 C
952 C      COPY THE STRING (SAME PROBLEM AS ABOVE)
953 C      DO 2055 I=1,10
954      KOUNT=I
955      TSTRNG(I)=IWORD(NWORD,I)
956      IF(IWORD(NWORD,I).EQ.IBLANK) GO TO 2060
957 C      WE'VE FOUND THE END OF THE LINE
958 2055 CONTINUE
959 2060 CONTINUE
960      KOUNT=KOUNT-1

```

```

961 C
962 2065 FORMAT(11110)
963 C
964 DECODE(KOUNT,2065,TSTRNG) IVALUE
965 RETURN
966 END
967 C
968 C
969 FUNCTION MATCHS(NUMB,STRING,NCHAR)
970 C
971 C THIS FUNCTION DETERMINES IF SCANNER TOKEN
972 C IWORD(NUMB) MATCHES THE CHARACTERS IN "STRING"
973 C AT LEAST FOR THE FIRST "NCHAR" CHARACTERS.
974 C
975 C IF THERE IS A MATCH, IT RETURNS THE INTEGER "1"
976 C NO MATCH RETURNS "0".
977 C
978 BYTE IWORD
979 COMMON/SCAN/NUMBER,IWORD(15,10)
980 BYTE STRING
981 DIMENSION STRING(10)
982 MATCHS=0
983 C
984 C TEST THE STRINGS...
985 DO 2070 I=1,NCHAR
986 IF(IWORD(NUMB,I).NE.STRING(I)) RETURN
987 2070 CONTINUE
988 C
989 C IF YOU GET HERE, THEY WERE THE SAME...
990 MATCHS=1
991 RETURN
992 END
993 C
994 C
995 SUBROUTINE ERRRRP(KIND,KALLER,NAME,MARK)
996 COMMON/RRR/MSG(11)
997 C MSG(N)=FATAL FLAG (1--FATAL)
998 BYTE KALLER,NAME
999 DIMENSION KALLER(10),NAME(10)
1000 MSG(1)=1
1001 MSG(2)=1
1002 MSG(3)=0
1003 MSG(4)=0
1004 MSG(5)=0
1005 MSG(6)=0
1006 MSG(7)=0
1007 MSG(8)=1
1008 MSG(9)=1
1009 MSG(10)=0
1010 MSG(11)=1
1011 C
1012 2101 FORMAT(5X,1A2,'ERROR ',1I2,' DETECTED BY ',10A1,
1013 1' MISSING SECT. BEGIN (FATAL) ',1A2)
1014 2102 FORMAT(5X,1A2,'ERROR ',1I2,' DETECTED BY ',10A1,
1015 1' SYMBOL TABLE OVERFLOW (FATAL) ',1A2)
1016 2103 FORMAT(5X,1A2,'ERROR ',1I2,' DETECTED BY ',10A1,
1017 1' NAME DUPLICATION (IGNORED) ',1A2)
1018 2104 FORMAT(5X,1A2,'ERROR ',1I2,' DETECTED BY ',10A1,
1019 1' ',10A1,' UNDEFINED (IGNORED) ',1A2)
1020 2105 FORMAT(5X,1A2,'ERROR ',1I2,' DETECTED BY ',10A1,

```

```
1021      1' --SYNTAX ERROR-- (IGNORED) ',1A2)
1022      2106 FORMAT(5X,1A2,'ERROR ',112,' DETECTED BY ',10A1,
1023      1' DYNAMIC CONFLICT ',10A1,1A2)
1024      2107 FORMAT(5X,1A2,' EVENT ',10A1,' MARKED ',1110,
1025      1A2)
1026      2108 FORMAT(5X,1A2,'ERROR ',112,' DETECTED BY ',10A1,
1027      1' NO REGIN EVENT FOUND (SOFTNET)',1A2)
1028      2109 FORMAT(5X,1A2,'ERROR ',112,' DETECTED BY ',10A1,
1029      1' NON-EXIST. HW UNIT REQUESTED.',1A2)
1030      2110 FORMAT(5X,1A2,'ERROR ',112,' DETECTED BY ',10A1,
1031      1' -----ERROR-10-----HAD-CALL-TO-ER',1A2)
1032      2111 FORMAT(5X,1A2,'ERROR ',112,' DETECTED BY ',10A1,
1033      1' R/S TABLE OVERFLOW (FATAL) ',1A2)
1034      C
1035      KSTAR=2H**
1036      C
1037      IF(KIND.LT.1 .OR. KIND.GT.11) KIND=10
1038      IF(KIND.EQ.1) GO TO 2121
1039      IF(KIND.EQ.2) GO TO 2122
1040      IF(KIND.EQ.3) GO TO 2123
1041      IF(KIND.EQ.4) GO TO 2124
1042      IF(KIND.EQ.5) GO TO 2125
1043      IF(KIND.EQ.6) GO TO 2126
1044      IF(KIND.EQ.7) GO TO 2127
1045      IF(KIND.EQ.8) GO TO 2128
1046      IF(KIND.EQ.9) GO TO 2129
1047      IF(KIND.EQ.10) GO TO 2130
1048      C      THEN KIND=11
1049      WRITE(7,2111) KSTAR,KIND,KALLER,KSTAR
1050      IF(MSG(KIND).EQ.0) RETURN
1051      CALL EXIT
1052      2121 CONTINUE
1053      WRITE(7,2101) KSTAR,KIND,KALLER,KSTAR
1054      IF(MSG(KIND).EQ.0) RETURN
1055      CALL EXIT
1056      2122 CONTINUE
1057      WRITE(7,2102) KSTAR,KIND,KALLER,KSTAR
1058      IF(MSG(KIND).EQ.0) RETURN
1059      CALL EXIT
1060      2123 CONTINUE
1061      WRITE(7,2103) KSTAR,KIND,KALLER,KSTAR
1062      IF(MSG(KIND).EQ.0) RETURN
1063      CALL EXIT
1064      2124 CONTINUE
1065      WRITE(7,2104) KSTAR,KIND,KALLER,NAME,KSTAR
1066      IF(MSG(KIND).EQ.0) RETURN
1067      CALL EXIT
1068      2125 CONTINUE
1069      WRITE(7,2105) KSTAR,KIND,KALLER,KSTAR
1070      IF(MSG(KIND).EQ.0) RETURN
1071      CALL EXIT
1072      2126 CONTINUE
1073      WRITE(7,2106) KSTAR,KIND,KALLER,NAME,KSTAR
1074      IF(MSG(KIND).EQ.0) RETURN
1075      CALL EXIT
1076      2127 CONTINUE
1077      WRITE(7,2107) KSTAR,KALLER,MARK,KSTAR
1078      IF(MSG(KIND).EQ.0) RETURN
1079      CALL EXIT
1080      2128 CONTINUE
```

petrinet.ftn Page 19 Tue Nov 27 06:18:55 1979

```
1081            WRITE(7,2108) KSTAR,KIND,KALLER,KSTAR
1082            IF(MSG(KIND).EQ.0) RETURN
1083            CALL EXIT
1084       2129    CONTINUE
1085            WRITE(7,2109) KSTAR,KIND,KALLER,KSTAR
1086            IF(MSG(KIND).EQ.0) RETURN
1087            CALL EXIT
1088       2130    CONTINUE
1089            WRITE(7,2110) KSTAR,KIND,KALLER,KSTAR
1090            IF(MSG(KIND).EQ.0) RETURN
1091            CALL EXIT
1092            END
```

LIST OF REFERENCES

1. Barbacci, M. R. and D. P. Siewiorek, "Evaluation of the CFA Test Programs via Formal Computer Descriptions", Computer Vol 10, No. 10 (October 1977).
2. Bell, C. G. and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, 1971.
3. Bell, C. G. and A. Newell, "The PMS and ISP Descriptive Systems for Computer Structures", Proc. AFIPS 1970 SJCC, Vol 36, pp 351-374.
4. Cox, L. A. Jr. "Performance Prediction of Computer Architectures Operating on Linear Mathematical Models", Ph.D. Thesis, Computer Science Dept., UC Davis Report UCRL-52582 (Sept 28, 1976).
5. Dietmeyer, D. L., "IFTRAN", Univ. of Wisconsin, Working Paper (November 1977).
6. Dietmeyer, D. L. "Digital Design Language Translator - DDLTRN", Univ. of Wisconsin, Working Paper, (November 1977).
7. Dietmeyer, D. L., "Digital Design Language Simulator - DDLSIM", Univ. of Wisconsin, Working Paper, (January, 1978).
8. Dietmeyer, D. L. and M. H. Doshi, "Automated PLA Synthesis of the Combinatorial Logic of a DDL Description", Univ. of Wisconsin, Report ECE-78-17, (November 1978).
9. Dietmeyer, D. L., "Translation of DDL Descriptions of Digital Systems", Univ. of Wisconsin Report ECE-77-13 (September 1977).
10. Dietmeyer, D. L., "Connection Arrays From Equations", Univ. of Wisconsin, Report ECE-78-18, (December 1978).
11. Doty, D. L. and G. J. Lipovski, "Developments and Directives in Computer Architecture", Computer, Vol 11, No. 8 (August 1978).
12. Ferrari, D., Computer Systems Performance Evaluation, Prentice-Hall, 1978.
13. Freeman, H. A., "Performance Evaluation Trends", IEEE Computer Society Conference Proceedings, COMPCON, (Fall 1978) pp. 396-398.

14. Lipovski, G. J., "Hardware Description Languages, Voices from the Tower of Babel", Computer Vol. 10, No. 6 (June 1977).
15. Loomis, H., "Memo 3.20 Progress Report of the Working Group of the Conference on Computer Hardware Description Languages", Working Paper (October 20, 1977).
16. MacMichael, A., "New DOD Effort In VHSICS", Military Electronics/Countermeasures (January 1979).
17. OMB Circular A-109 (August 1976).
18. Powers, V. M., "Functional Program Modules (FPMs) and Digital Systems Design", Report NPSA52PW72071A (20 July 72).
19. Reitmeyer, R. A. Jr., "Computer Aided Design, Design Automation and LSI; Keys to High-Performance Military Electronics", ERADCOM (June 1978).
20. Salisbury, A., LTC and Bruce Wald, "The Computer Architecture Project: Service Prospective and Overview", Computer Vol. 10 No. 10 (October 1977).
21. SEAFIRE Proposal Vol. 4C, "Substantiating Technical Data" (29 May 1979).
22. SEAFIRE Weapon System Specification XWS-17824.
23. Su, Stephen Y. H., "An Introduction to CHDL (Computer Hardware Description Languages)", Computer Architecture News, Vol. 4, No. 3 (September 1975), pp 22-23.
24. Su, Stephen Y. H., "Hardware Description Language Applications", Computer Vol. 10, No. 6 (June 1977).
25. Weiss, D. M., "Evaluating Software Development by Error Analysis: The Data From the Architecture Research Facility," Naval Research Laboratory Report 8268 (December 28, 1978).

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Curricular Officer, Code 33 Weapons Engineering Naval Postgraduate School Monterey, California 93940	1
4. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5. Associate Professor Lyle A. Cox, Code 52CL Department of Computer Science Naval Postgraduate School Monterey, California 93940	4
6. Naval Sea Systems Command Washington, D.C. 20362 Attn: SEA 62Y21D Douglas M. Stowers	3
7. Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1