AD-A083 842   NAVAL POSTGRADUATE SCHOOL MONTEREY CA   F/G 9/2
PERFORMANCE PREDICTION FROM A COMPUTER HARDWARE DESCRIPTION.(U)
DEC 79 L A COX
UNCLASSIFIED   NPS52-79-001   NL

1 OF 1
AD A
083842

END
DATE
FILMED
6-80
DTIC

② LEVEL II

NPS52-79-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

ADA083842

PERFORMANCE PRECISION FROM A COMPUTER HARDWARE DESCRIPTION

Lyle Ashton Cox, Jr.

December 1979

Approved for public release; distribution unlimited

Prepared for:
Chief of Naval Research
Arlington, VA 22217

80 5 6 046

9 Jan 1980

NAVAL POSTGRADUATE SCHOOL
Monterey, California

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

[illegible]

LYLE A. COX, JR.
Assistant Professor of Computer Science

Reviewed by:

[illegible]

G. H. BRADLEY, Chairman
Department of Computer Science

Released by:

William M. Tolles

WILLIAM M. TOLLES
Dean of Research

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
14 1. REPORT NUMBER NPS52-79-001 ✓	2. GOVT ACCESSION NO. AD-A083 842	9 3. RECIPIENT'S CATALOG NUMBER
6 4. TITLE (and Subtitle) Performance Prediction from a Computer Hardware Description.		5. TYPE OF REPORT & PERIOD COVERED Technical Report.
		6. PERFORMING ORG. REPORT NUMBER
10 7. AUTHOR(s) Lyle Ashton Cox, Jr		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School ✓ Monterey, CA 93940 16 R00001		12 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61152N; RR000-01-10 N0001480WR00054
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940 12 25		11 12. REPORT DATE Dec [redacted] 79
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

Performance Evaluation
Simulation
Petri-Nets

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

Today's computers are among the most complex man made systems in existence today. How can we make our designs not only more precise, but make these systems more accurately and cost effectively achieve their goals? We have begun to reply upon computer aided design techniques. The use of these techniques often commences not with the statement of the system goals, but rather with the specification of an architecture and a logic technology. In view of the high costs of computer development, we should be quite confi-

next page

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

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251450

dent that our architecture/technology combination is capable of meeting the system requirements before we proceed to more detailed design phases. The entire system design process must be integrated, and include performance prediction and verification techniques. This design process should be reflected in a single description language. A methodology for accomplishing this goal capable of employing many existing design practices is discussed here.

ABSTRACT

Today's computers are among the most complex man made systems in existence today. How can we make our designs not only more precise, but make these systems more accurately and cost effectively achieve their goals? We have begun to reply upon computer aided design techniques. The use of these techniques often commences not with the statement of the system goals, but rather with the specification of an architecture and a logic technology. In view of the high costs of computer development, we should be quite confident that our architecture/technology combination is capable of meeting the system requirements before we proceed to more detailed design phases. The entire system design process must be integrated, and include performance prediction and verification techniques. This design process should be reflected in a single description language. A methodology for accomplishing this goal capable of employing many existing design practices is discussed here.

[illegible]

A

PERFORMANCE PREDICTION FROM A COMPUTER HARDWARE DESCRIPTION

by

Lyle Ashton Cox, Jr.
Department of Computer Science
Naval Postgraduate School
Monterey, California

Introduction

We do not need to develop more computer system description languages; we need to develop a single, comprehensive language capable of unifying the system design and implementation process. Today's computers are among the most complex man made systems in existence today. The development of such systems represents a significant commitment of physical and mental resources. This cost can only be justified if these computing devices serve their intended purpose -- the efficient processing of data in response to specific needs. How can we make our designs not only more precise, but make these systems more accurately and cost effectively achieve this goal? What can we do in the design phases of computer system development to insure that our final product will meet our expectations? We have begun to reply upon computer aided design techniques. The use of these techniques often commences not with the statement of the system goals, but rather with the specification of an architecture and a logic technology. In view of the high costs of computer development, we should be quite confident that our architecture/technology combination is capable of meeting the system requirements before we proceed to more detailed design phases. The entire system design process must be integrated, and include performance prediction and verification techniques. This design process should be reflected in a single description language. A methodology for accomplishing this goal capable of employing many existing design practices is the subject of this paper.

MOTIVATION

Classically, new digital systems have evolved in response to specific problems or families of problems. In the case of large computer systems, this evolution has usually occured in the following sequence of events:

(1) A problem is recognized with no existing (economical) solution.

(2) The problem is analyzed.

(3) A concept is developed for the design and construction of a new hardware system.

(4) The system is designed and constructed.

(5) Software is generated for the applications.

(6) The system is tested and made operational.

(7) The system's performance is enhanced by software and hardware modifications as required.

In some cases the software may already exist, and will form part of the problem statement for the hardware designer. In other cases, step (5) may preceed steps (3) and (4). Essentially, these steps are not allowed to proceed in parallel.

Partially this reluctance to allow the hardware and software designs to proceed simultaneously appears to be due to the sequential nature of human thought. Much of the reason for our sequential design methods stems from limitations of our current design tools. In terms of overall system performance, the hardware and software are closely coupled. To design one of these two components without knowledge of the other's structure (or the knowledge that the other will be designed later for compatibility) is poor strategy. It allows too many independent variables for efficient design using conventional techniques.

Furthermore, the sequential nature of this design process limits the speed with which we can implement effective systems. This is one reason why today's systems use yesterday's technology. It also makes our design process one of trial and error. Until both the software and hardware are complete, we can not assess overall performance. (If we are willing to pay the high costs of simulation, we can perform some testing when the designs are complete. This is still essentially trial and error development, since we have paid the price for the detailed designs before we can begin the simulations.) If performance proves satisfactory, all is well and good. If not, we are condemned to a lifetime of expensive system modifications (step 7).

If demand for computing was static, perhaps we could continue to develop new systems in this manner. This is not the case. Our design load is increasing both in volume and complexity. Modern technology has lowered the hardware costs sufficiently so that unique, architecturally customized, limited purpose machines are becoming economically feasible. Since design costs are essentially independent of eventual production volume, the increasing introduction of limited purpose machines will add to the overall design demand. In a somewhat similar manner, the decreasing hardware costs coupled with requirements for high performance systems had lead to the increasing use of digital systems capable of executing multiple operations simultaneously. We have comparatively little experience with systems of this type, and there is no formal theory to aid our investigations. Our design efforts for these complex systems are therefore more costly. For these reasons and sheer increased demand, our design load is growing. We can not continue to design systems as we have.

The only viable solution to the increasing design load is to increase productivity. The digital engineering community recognized this problem relatively early, and computer aided design/design automation efforts have been actively explored for years.

INCREASING DESIGN "EFFICIENCY"

Much of the work in developing computer aided design tools has focused on the stages of the design process described by Su (1975):

"The task of designing a digital 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 several 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.

(6) The interconnecting of chips.

(7) The testing of the integrated circuit boards."

While digital design efforts which have followed this pattern have been both valuable and necessary, they have served only to enhance the existing (sequential) design process. Design automation, and the development of Computer Hardware Description Languages (CHDL's) have served to broaden the scope of optimization within this design process. These efforts have not significantly changed the logical structure of the digital design task.

Just as we are moving from sequential computation to parallel computation in order to increase our processing throughput, we must move from sequential system design techniques to parallel design techniques to increase our design throughput. To make this type of design possible we will have to create appropriate controls and methodologies which enforce coordination and assure that our designs, when complete, fulfill the original system requirements.

Hardware engineers are not alone in this problem. Much the same situation is being faced by software engineers. They are charged with the development of large software systems, and traditional design efforts are no longer adequate. Their moves to develop specification languages, and concurrent control descriptions are allowing them to introduce more local parallelism into the software design process while increasing reliability. Hardware design automation and CHDL analysis will allow us the same advantages. Neither of these localized efforts is sufficient.

Further major improvements to our digital system design process must stem from "global" rather than "local" optimizations. We must view the entire process as a whole, from the formulation of requirements to the system implementation, including the design and development of both the hardware and the problem software. Can we undertake such a massive task?

THE UNIFIED SYSTEM DESIGN PROCESS

In fact, we may not be very far away from just such a unified system design and implementation practice. Mateland (1976) demonstrated an unoptimized but comprehensive system capable of designing both hardware and software for specialized control systems. Work in design automation and CHDL analysis, as well as the analogous work in software engineering provides many of the basic tools we will need. Consider the entire digital system design and development process of large computing systems as outlined in Figure 1.

The initial system development begins in the "Problem Definition Phase" where, after the need for a new system has been recognized, the requirements are formalized, and specifications for system performance are written. These requirements and specifications should be independent of perceptions of current technological capabilities. They must also be either free from abiguity or contain an explicit ambiguity resolution methodology. Significant progress is being made in this area by software engineers, administrative scientists and operations analysis researchers.

Once the system's requirements have been specified (in machine processable form, so as to make this information available to the automated design tools used in the following stages) the system design enters a "Conceptual Design and Analysis Phase." In this phase the hardware and the software designs are started concurrently. The hardware designers, using what I term "Hardware Conceptual Design Descriptions" (HCDD's) develop the basic structure of the processing system, and select the implementation technologies. The software development proceeds, with the selection of algorithms and the overall control structure as expressed in enhanced (concurrent) specification languages.

Before investing more effort and resources on the designs, these design concepts are validated. Using performance prediction techniques (which will be discussed later) the requirements of the software structure are mapped onto the hardware, and the overall system performance evaluated in terms of cost, system throughput, response time, subsystem utilization, logical performance of algorithms etc. The predicted performance is compared with the specifications for the system, and both the hardware and software design concepts can be iteratively refined until the performance appears satisfactory.

At this point we can be confident that the hardware and the software designs will perform adequately together, and that the overall system performance will be satisfactory. The design concepts have been validated. We then proceed to the next stage: "Design Implementation."

Here, the hardware design is completed using automated techniques which follow Su's steps. The software design is also completed and some portions of code written. At this point the performance of the completed design can be more precisely verified using simulation techniques similar to those demonstrated during the selection of the Army-Navy Joint Computer Family Architecture (Barbacci and Siewiorek, 1977). Again, the results are compared with the system specifications, and the designs iteratively modified if necessary. When the designs are shown to be satisfactory, the system enters the final development stage: "Construction and Integration."

This is the "nuts and bolts" implementation of the designs which we have developed. Since performance prediction and monitoring have been a continuing part of the design process, there should be comparatively little doubt that the system will function not only as advertised, but as originally desired.

What is stopping us from having such a unified, optimized system implementation process today? There are several problems which must still be overcome. Even if all of the various functions were understood well, the creation of working interfaces between the various phases, the software and hardware aspects, the designers and users, is certainly not a trivial task. Many technical and not so technical problems exist, including developing the specifications and requirements interface to the non-engineer.

Perhaps more significant, it should be realized that the implementation of such an integrated system design system has certain unavoidable language design problems. We no longer can be content to just describe computer hardware. We must describe performance, specifications and software as well. Moreover, our language must live and grow as the design grows. We have almost no experience with "information added" languages, where a valid "program" is defined, and then is refined and expanded in iterative steps. It is precisely this adding of information at each stage which the system design process requires. Development of efficient, usable languages which support "information adding" will be a challenge.

ASSURING COMPATIBILITY OF THE DESIGNS

The most difficult problem, however, is achieving (guaranteeing) the coordination between the hardware and the software. This is controlled by the performance prediction and performance evaluation stages shown in Figure 1. Conceptually, we would like this process to proceed conttinuously throughout the designs evolution. However, the difference between the two stage method hypothesized and the continuous evaluation will not be significant in comparison to other effects.

The problem of performance evaluation, given the hardware and the software designs has been shown to be feasible. The ISPL based simulator used in the Army-Navy Joint Computer Family Architecture selection is one example.

The remaining key stage of our integrated system design process is then the performance prediction of systems in the conceptual design stages. If this problem can be resolved, and in a manner such that the integration of the components is facilitated, and "information adding" is supported, then the entire concept of a unified digital system design and implementation language system is possible. The remainder of this paper demonstrates such a performance prediction system exists.

CONCEPTUAL DESIGN PERFORMANCE PREDICTION

In the conceptual design and analysis phase shown in Figure 1, the inputs to the performance prediction package consist of three main sources of information. These are: first, the user's performance requirements in terms such as "turnaround time," "thruput," "system and "sub-system utilization," and of course, "cost;" second, in response to the user's processing requirements, a description of the algorithms and control structure of the software in terms of the requirements it will levy against the hardware; and third, a description, quantized in terms of its actual response in time, of the actual hardware resources and any limitations enforced by the nature of the organization.

This initial performance evaluation stage, as described utilizes

"Hardware Conceptual Design Descriptions" (HCDD) and software specifications to predict performance. In order to be compatible with other computer hardware description languages (i.e. CHDLs) the HCDD's must meet the criteria outlined by Su and Baray (1975). Such languages must facilitate multi-level modeling and support variable levels of detail. They must allow the specification of the structure and the control of the system they describe. They must allow analysis by decomposition. They must be able to describe synchronous, asychronous and mixed systems. And finally they must be conceptually similar to the subject matter, and report back results in usable terms.

A language which meets all of these requirements would be capable of providing more and more detailed performance projections as the design progresses (information adding). It would also allow the graceful transition into existing CHDL/DA systems. Such a system would also serve as a useful tool for evaluating the impact of mid-term design re-directions, changes in tasking, or technology changes.

A suitable test-bed language, "P5" has been developed and refined in accordance with these goals.

THE PETRI PERFORMANCE PREDICTIVE PACKAGE

In response to the requirement for an architectural design aid a performance prediction system based on Petri-Net models was created. The system, named P4, standing for Petri Performance Predictive Package, operates as follows. When developing systems, the designers describe their concepts in terms of the P4 system. A P4 program (P5) consists of a description of the computer system organization and capabilities, and a description of software control and functional requirements. These descriptions are Petri-Nets, and in order to make use of the hierarchical nature of these nets, and to express system organizations in a more concise and convenient manner, a macroprocessor is included in the system. P5 descriptions include a prototype HCDD and a software control specification language. This description of the solution concept is then evaluated in a dynamic sense and directly produces an analysis of the system's predicted performance.

Conceptually, this prediction methodology takes an algorithm and expresses the control structure of all or some representative kernel of the algorithm in a fashion which makes the potential parallelism exploitable. For a given computer system, the control sturcture dictated by the software is then mapped onto a similarly expressed hardware structure, and the performance evaluated.

The key to this process is the expression of a representative program kernel and hardware control structure as special kinds of concurrent control system models (in this case, a Petri-Nets) similar to the marked, directed graph discussed by Commoner, et al. (1971). This type of approach has been recently suggested by others, including Dennis, Misunas and Leung (1977) to predict the performance of computer systems including data flow machines. It has also been used by Patil (1975) to describe digital systems and their behavior in the context of CHDL's.

In a directed graph representing the logical flow of functions to be performed, 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 our program. This problem was solved by Karp and Miller in 1966.

The requester/server interface (Cox, 1978), allows the construction of a two graph structure which in a wide variety of interesting circumstances is equivalent to the single graph. The two graph nature of the requestor/ server interface allows the representation of user 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 in each domain.

A complete review of Petri-Nets will not be given here. For a more detailed treatment, Peterson's recent Computing Survey article (1978) provides excellent background. Briefly, however, 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" with "tokens" to respond to the flow of information (or the occurrence of events) in the system. Petri-Nets can model actual parallel processes by attaching some significance to token movement.

Petri-Net concurrent control system models have many characteristics which are desirable in a performance prediction system. This model is capable of representing both hardware and software systems and is hierarchical in nature. These characteristics are intrinsically important, and important in interfacing to existing CHDL/DA systems.

In the two net system, 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 represents the logical flow of the algorithm.

In the hardware net, events roughly represent operations in time. A collection of one or more event are used to represent a functional unit and its temporal response to the hardware control constaints. Token movement through the hardware net represents the data and control flow of the hardware system. An example of an early P5 description is shown in Figure 2, which describes a very simple algorithm and an integer adder to be used in computing the solution.

Based on this concept, good predictions of system performance have been demonstrated (Cox, 1978). In validating this approach, the performance of FORTRAN programs of over 1000 statements to be run on Control Data 6000 and 7000 series machines was predicted to within a few percent of actual measured values. Subsequent work has focused upon making the P4 system better fulfill the requirements of the performance prediction function required for use in the "Conceptual Design and Analysis" phase of system development.

The original P5 language was developed directly from the formal definition of Petri-Nets, and was implemented to present the appearance of a procedural language. Descriptions of both hardware and software were written in essentially identical terms. While a macroprocessor was included in the P4 system, its intended use was merely convenient to take advantage of the hierarchical nature of the Petri-Net model.

In trying to make the P4 system easier to use, and to make the P5 language interface with other existing description languages, the power of the macroprocessor became evident. Highly customized user interfaces can be constructed, which continue to produce uniform Petri-Net models after expansion and translation. Since the macroprocessor operates with a default library of predefined macros as well as dynamically defined macros, custom user interfaces can be built. By making the macroprocessor sensitive to the current program context (i.e. hardware description or software description etc.) both phases of the description can present the appearance of different languages, and allow the descriptions to use terminology familiar to the particular designer. The addition of random effects has allowed the simulation of I/O and other nondeterministic phenomena, including the exercising of alternative branches in software systems. These changes allow the designer to define software and hardware structures more naturally and realistically.

The importance of these changes can be seen from the following example. A HCDD specification of a dual, synchronous Amdahl A470V/6 CPU system written in the original P5-Macro language required about 210 lines of code, 86% of which were used to specify the control and precedence of machine states. (The remaining statements defined actual hardware units.) By modifying and developing the macroprocessor, it is now possible to specify the identical system in 26 statements (a 7 fold reduction), 21 of which specify the control and precedence (73%). If it were not for the unique pipeline "flush" mechanism which the A470V/6 CPU uses for responding to interrupts, the dual system could be specified in six to ten statements of the type:

```
begin hardware;
define V6PIPE macro type PIPELINE
    of 12 states, of 30 ns;
declare CPU1 macro type V6PIPE;
declare CPU2 macro type V6PIPE;
declare SYN1 macro type SYNCHRO.S.2
    with output = CPU1;
declare SYN2 macro type SYNCRHO.S.2
    with output = CPU2;
declare CLOCK macro type 2P.CLOCK
    of 30 ns
    with output=SYN1, and
    with output=SYN2;
end hardware;
```

This system allows the user to draw from predefined macros in the library, such as "PIPELINE" and the clock macro, as well as to define new macros which are built from the existing library stock. As a specific user gains experience with the system, the continuing redefinition of library macros causes the system to adapt itself to the user and his application, without affecting the end model's structure. This, in turn, insures that the interface to CHDL's which may follow this HCDD in the design process are presented with a uniform model.

In a similar manner, the software descriptions have evolved using the macroprocessor for language enhancements and extension. It is now possible to define programs in a flowchart like manner but with concurrency explicitly defined. For example, with a suitable macrolibrary defined, a signal processing program might be defined as follows:

```
begin software;
declare DATA event type RANDOM,
    with P=0.0071;
define JFILTER macro type PASSFILTER,
    with 2 bands;
define XFORM macro type FFT
    with 1024 points;
declare SIGNALTOFREQ macro type XFORM;
declare INVERSEXFORM macro type XFORM;
DATA preceeds SIGNALTOFREQ;
SIGNALTOFREQ preceeds JFILTER;
JFILTER preceeds INVERSEXFORM;
end software;
```

While not yet perfected, languages such as these are much more usable than previous versions, and offer compatibility with the other systems shown in Figure 1.

CONCLUSIONS

Looking at the process of computer system development as a whole, optimization on a global scale appears both necessary and possible. The key to achieving this optimization is the ability to predict the performance of computer systems early in the design development. This prediction must consider the conceptual organization of both the hardware and the software, support "information adding," and interface to existing design assistance systems. The technology for such a performance prediction system has been demonstrated. The interfacing of this methodology is shown to be possible, the design has been shown to be capable of accurate prediction. Only hard work remains.

REFERENCES

Barbacci, M. and D. P. Siewiorek, "Evaluation of the CFA Test Programs via Formal Computer Descriptions," _Computer_, Vol. 10, No. 10 (October, 1975).

Commoner, F. A. et al, "Marked Directed Graphs," _J. of Computer and Systems Sciences_, Vol. 5 (1971).

Cox, L. A., "Predicting Computer System Performance Using Petri-Net Models," _Proc. 1978 Annual Conference of the ACM_ (December 1978).

Dennis, J. B., D. Misunas, and C. Leung, "A Highly Parallel Processor Using a Data Flow Machine Language," M.I.T. Computation Structures Group Memo 134 (January 1977).

Karp, R. M. and R. E. Miller, "Parallel Program Schemata," _J. Computer and Systems Sciences_, Vol. 3 No. 4 (May, 1969).

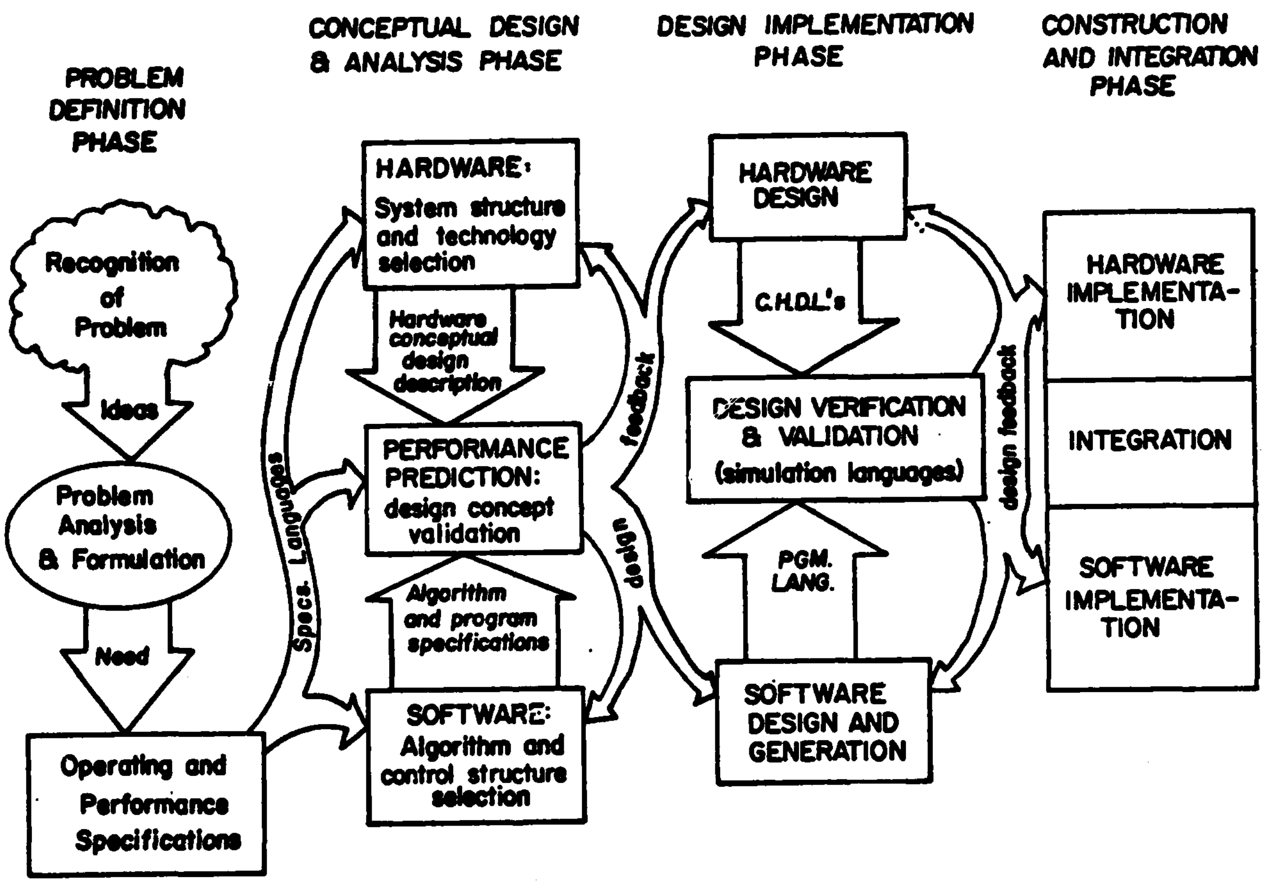
Matelan, M. N., "Automating the Design of Microprocessor Based Real Time Control Systems," Proc. ACM/IEEE Conference on Design Automation, No. 13 (June 1976).

Patil, S. S., "On Structured Digital Systems," Proc. 1975 International Symposium on Computer Hardware Description Languages and Their Applications (September 1975).

Peterson, J. L., "Petri-Nets", Computing Surveys, Vol. 9, No. 3 (September 1977).

Su, S. Y. H., "Introduction" Proc. 1975 Symposium on Computer Hardware Description Languages and Their Applications, (September 1975).

Su, S. Y. H., and M. B. Baray, "LALSD - a Language for Automated Logic and System Design," Proc. 1975 Symposium on Computer Hardware Description Languages and Their Applications, (September 1975).


PROBLEM DEFINITION PHASE
CONCEPTUAL DESIGN & ANALYSIS PHASE
DESIGN IMPLEMENTATION PHASE
CONSTRUCTION AND INTEGRATION PHASE
Recognition of Problem
Ideas
Problem Analysis & Formulation
Need
Operating and Performance Specifications
Specs. Languages
HARDWARE: System structure and technology selection
Hardware conceptual design description
PERFORMANCE PREDICTION: design concept validation
Algorithm and program specifications
SOFTWARE: Algorithm and control structure selection
feedback
design
HARDWARE DESIGN
C.H.D.L.'s
DESIGN VERIFICATION & VALIDATION (simulation languages)
PGM. LANG.
SOFTWARE DESIGN AND GENERATION
design feedback
HARDWARE IMPLEMENTATION
INTEGRATION
SOFTWARE IMPLEMENTATION

COMPUTER SYSTEM DESIGN

Figure 1

P⁴ – A SIMPLE EXAMPLE

A software program:

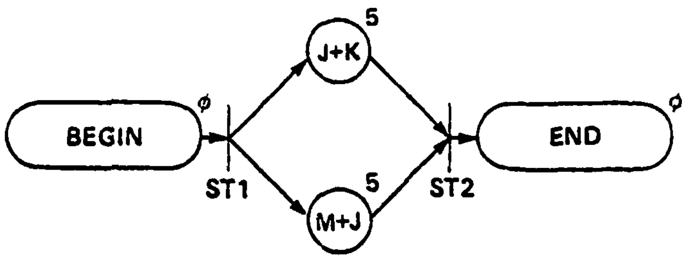
P^5

```
Begin program example;
Declare BEGIN event type φ;
Declare J + K event type 5;
Declare M + J event type 5;
Declare END event type φ;
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:

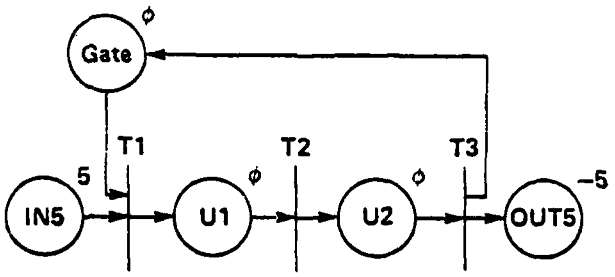


A hardware functional unit:

P^5

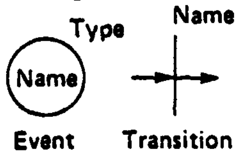
```
Begin machine net;
Declare IN5 event type 5;
Declare Gate event type φ;
Declare U1 event type φ;
Declare U2 event type φ;
Declare OUT5 event type –5;
Declare T1 transition;
   Input IN5;
   Input Gate;
   Output U1;
End T1;
Declare T2 transition;
   Input U1;
   Output U2;
End T2;
Declare T3 transition;
   Input U2;
   Output Gate;
   Output OUT5;
End T3;
End machine net;
```

Petri-network



An adder (3 minor cycles)
(NOT PIPELINED)

Legend:



(TYPE REFERS TO THE KIND OF SERVICE EITHER PROVIDED OR REQUESTED.)

Figure 2.

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.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93940	1
4.	Chairman, Code 52Bz Computer Science Department Naval Postgraduate School Monterey, California 93940	30
5.	Lyle A. Cox, Jr., Code 52Cl Computer Science Department Naval Postgraduate School Monterey, California 93940	10