

NO-A100 023

A DIGITAL LOGIC SIMULATOR WITH CONCURRENT PROGRAMMING
CONSIDERATIONS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

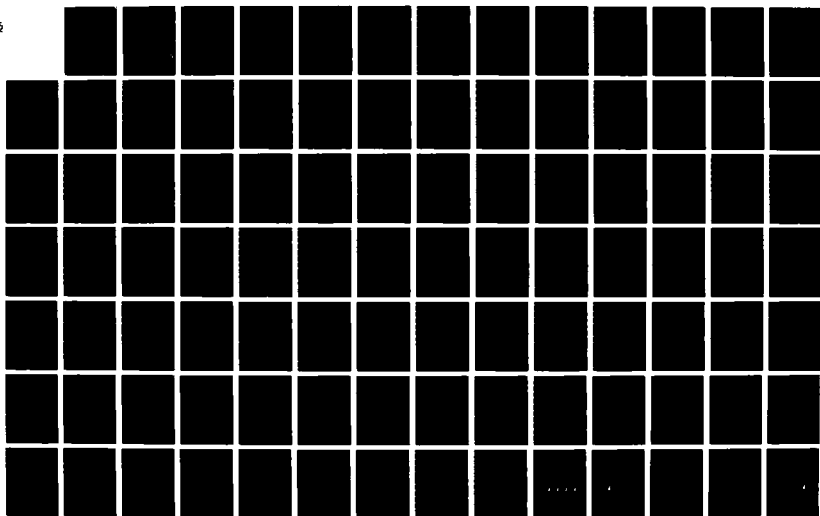
1/2

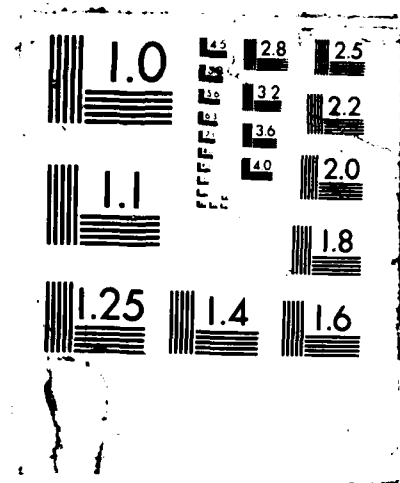
UNCLASSIFIED

W C DELORIA DEC 87 AFIT/GCS/ENG/87D-10

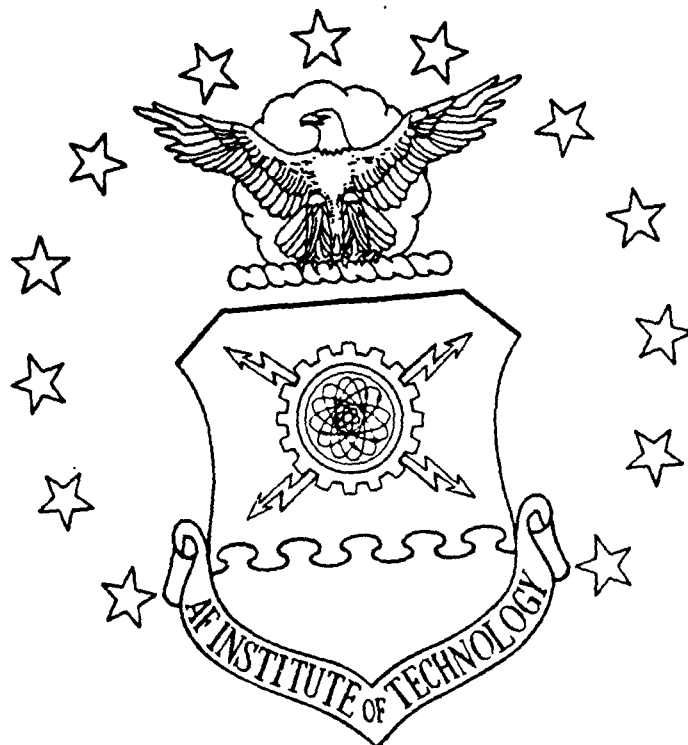
F/G 12/1

NL





AD-A188 823



DTIC FILE COPY

A DIGITAL LOGIC SIMULATOR WITH
CONCURRENT PROGRAMMING CONSIDERATIONS

THESIS

Wayne C. DeLoria
Captain, USA

AFIT/GCS/ENG/87D-10

DTIC
ELECTE
FEB 10 1988
S D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale; its
distribution is unlimited.

88 2 4 0 53

1

AFIT/GCS/ENG/87D-10

A DIGITAL LOGIC SIMULATOR WITH
CONCURRENT PROGRAMMING CONSIDERATIONS

THESIS

Wayne C. DeLoria
Captain, USA

AFIT/GCS/ENG/87D-10

DTIC
ELECTRONIC

Approved for public release; distribution unlimited

AFIT/GCS/ENG/87D-10

A DIGITAL LOGIC SIMULATOR WITH CONCURRENT
PROGRAMMING CONSIDERATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology,

Air University,

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems



Wayne C. DeLoria
Captain, USA

December 1987

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited

Acknowledgments

Perhaps one of the greatest lessons learned from an intensive period of research and study, such as this thesis effort, is the realization that such undertakings can never come to fruition without the assistance and understanding of others. We all hope for self-sufficiency, but reality proves that we are all gregarious. Many colleagues and friends were helpful throughout this effort, but some really stand out as lifesavers.

Of course, I could not have accomplished any amount of success without the guidance and assistance of my advisor, CPT N.J. Davis. The other members of my advisory committee were equally important --CPT Bruce George, whose bright disposition kept us all going, and CPT Wade Shaw, whose inspiration was responsible for this whole thing. Dr. Frank Brown, who oversaw the original version of the simulator, also supplied some direction.

Various sections of this report would never have gone to press were it not for classmates and colleagues. CPT Edward Poore saved my life (and perhaps my sanity) with his undying patience and indispensable knowledge of statistics and their applications. LT Bill Hodges' intimate familiarity with the iPSC Hypercube saved uncounted hours in the pursuit of many

elusive bugs which kept popping up in concurrent applications.

My two partners in the development of the integrated digital design tool (IDIET) were also instrumental in whatever success was realized by this project. Although they were working toward different goals, without the productive interaction with CPT Charles Adams and LT Steven Wagner, I might very well still be at the drawing board.

And last, but certainly not least, my most heartfelt thanks and love go out to my wife Sheryl, without whose support, and sometimes tearful understanding, this thesis would never have been completed.

Table of Contents

	Page
Acknowledgments	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
CHAPTER 1 - THESIS OVERVIEW AND ORGANIZATION . . .	1
1.1 Overview	1
1.2 Organization	3
CHAPTER 2 - INTRODUCTION AND OBJECTIVES	5
2.1 Introduction	5
2.2 Research Objectives	10
2.3 Summary	12
CHAPTER 3 - LOGSIM	13
3.1 Background	13
3.2 LOGSIM248 Implementation	17
3.2.1 Overview	17
3.2.2 Data Structures	18
3.2.3 Operation	22
3.3 Summary	24
CHAPTER 4 - SOFTWARE DEVELOPMENT	25
4.1 Re-engineering Overview	25
4.2 Analysis - LOGSIM, version 5.5	25
4.3 Design - LOGSIM248	31
4.4 Coding - LOGSIM248	35
4.5 IDIET Integration	39
4.6 Testing and Debugging - LOGSIM248	40
CHAPTER 5 - CONCURRENT LOGSIM	43
5.1 Overview	43
5.2 Parallel Approach	46
5.3 Implementation	47
5.4 Summary	49

	Page
CHAPTER 6 - DISCUSSION	51
6.1 Comparing Performance - LOGSIM V5.5 vs. LOGSIM248	51
6.2 Concurrent LOGSIM Performance	56
6.3 Using LOGSIM248 Independently	59
6.4 LOGSIM248 IC Library Expansion	61
6.5 Future Directions	62
Bibliography	66
Appendix A: Source Code - LOGSIM248	68
Appendix B: Data Flow Diagrams and Functional Decomposition	130
Appendix C: Statistical Information	148
Appendix D: Test Case Schematics and Graphic Interface Images	162
Appendix E: LOGSIM248 Manual for Independent Operation	175
E.1 Overview	E-1
E.2 What Is LOGSIM248?	E-2
E.3 Materials Required	E-3
E.4 LOGSIM248 File Interface	E-5
E.4.1 Circuit Configuration Input File - TEMP.CKT	E-6
E.4.2 Output Specification File - TEMP.DIS	E-9
E.4.3 Input Data File - TEMP.IN	E-9
E.4.4 Input Port Label File - TEMP.IND	E-10
E.5 Invoking The Simulator	E-11
E.6 Summary	E-12
ANNEX A: LOGSIM248 TTL IC Library	E-13
ANNEX A. Example Input/Output File Structure	E-15
Vita	193

List of Figures

	Page
B1. LOGSIM248 A0	B-1
B2. LOGSIM248 A1	B-2
B3. Build Ckt Data Structure	B-3
B4. Build Input Data Structure	B-4
B5. Build Output Data Structure	B-5
B6. Build Output Files	B-6
B7. main()	B-7
B8. buildckt(), cktinit()	B-8
B9. setpin()	B-9
B10. connect()	B-10
B11. findic()	B-11
B12. catalog()	B-12
B13. simulate()	B-13
B14. initout(), initinput(), buildinput()	B-14
B15. operate()	B-15
B16. addout()	B-16
B17. fileout()	B-17
D1. Schematic Diagram for BCD.CKT	D-1
D2. Inputs for BCD.CKT	D-2
D3. Graphic Circuit Image for BCD.CKT	D-2
D4. Binary Output Image for BCD.CKT	D-3
D5. Waveform Output for BCD.CKT	D-3
D6. Schematic Diagram for ADDER.CKT	D-4

	Page
D7. Graphic Circuit Image for ADDER.CKT	D-5
D8. Inputs for ADDER.CKT	D-5
D9. Binary Output Image for ADDER.CKT	D-6
D10. Waveform Output Image for ADDER.CKT	D-6
D11. Schematic Diagram for BCR3S.CKT	D-7
D12. Graphic Circuit Image for BRC3S.CKT	D-8
D13. Binary Output for BRC3S.CKT	D-9
D14. Waveform Output for BRC3S.CKT	D-9
D15. Schematic Diagram for DECODER.CKT	D-10
D16. Graphic Circuit Image for DECODER.CKT	D-11
D17. Inputs for DECODER.CKT	D-11
D18. Binary Output for DECODER.CKT	D-12
D19. Waveform Output for DECODER.CKT	D-12
E1. LOGSIM248 TTL IC Library	E-A-1
E2. Schematic For TEMP.CKT - Binary Coded Decimal Encoder	E-B-2
E3. Circuit Configuration Input File - TEMP.CKT . . .	E-B-2
E4. Output Monitor File - TEMP.DIS	E-B-2
E5. Input Data File - TEMP.IN	E-B-2
E6. Input Port Label File	E-B-2
E7. Binary Output File - TEMP.OUT	E-B-3
E8. Waveform Output File - TEMP.WAV	E-B-3

List of Tables

	Page
1. Data File Formats	21
2. File Format Codes	21
3. Simulation Run Time Comparison	55
4. Mean Run Time Comparison for Concurrent Implementation	57
E1. Data File Formats	E-7
E2. File Format Codes	E-7

Abstract

The digital logic simulator, LOGSIM248, a re-engineered version of LOGSIM, version 5.5, has been implemented as a component of the digital design environment, IDIET (Integrated Digital Engineering Tool). This new design expands the capabilities of the older version by improving run time performance, maintainability, and compatibility. Written in the C programming language, LOGSIM248 boasts looser coupling between functional modules while exhibiting greater functional cohesion within these modules. As an integral part of IDIET, the simulator overcomes difficulties created by the complicated user interface of earlier versions.

With greater run time performance as a goal, this new simulator was studied and adapted to produce a concurrent implementation. Here, several roadblocks were encountered which essentially showed this algorithm and data structure implementation to be difficult to "parallelize" at best. Due to communication constraints on the host computer, data structures used to simulate circuits caused large delays due to the requirement to disassemble and re-assemble them at the

various processing nodes. This program handicap coupled with communication transmission delays between processors resulted in time complexity problems.

Essentially a software engineering project, the re-design of LOGSIM, version 5.5, was necessitated by various shortcomings associated with the older version. The new implementation conforms to the proposed ANSI standard for the C programming language by utilizing only standard library functions and source code which complies with the original Kernighan and Ritchie model. This re-hosting has improved system portability allowing LOGSIM248 to run on all MS/DOS micro-computers available to the designer.

A DIGITAL LOGIC SIMULATOR WITH CONCURRENT PROGRAMMING CONSIDERATIONS

CHAPTER 1 - THESIS OVERVIEW AND ORGANIZATION

1.1 Overview

Recently, a great deal of attention has been directed toward methods which enhance efforts to design and build efficient and functional digital circuits with minimum cost and maximum performance. The primary products resulting from these efforts are a plethora of computer aided design (CAD) tools appearing on today's computer software market. These tools aid the user in circuit design, circuit board layout, and, in some cases, circuit simulation. Although not all CAD tools include simulation as a part of their working environment, simulation of digital circuits is an integral part of the design process and should be considered when building such tools.

In this thesis effort, the simulation of digital circuits is explored with the production of a logic simulator as a functional by-product. This study improves on an existing simulator called LOGSIM which was developed at the University of Kentucky and enhanced by students at the Air Force Institute of Technology (1). This design tool

simulates digital circuit designs at the integrated circuit (IC) level, allowing chip and pin level circuit descriptions to be tested for functionality and performance. Various problem areas were still present within the enhanced version produced at AFIT - particularly simulation run time for large circuit designs, and software engineering inadequacies which could hamper maintenance and future upgrades.

This thesis effort centers around two main areas of investigation. First, LOGSIM is re-engineered for inclusion as an integral part of a digital design tool which offers a graphic user interface, a circuit connectivity expert system, and the logic simulator. Secondly, methods are explored to re-host LOGSIM on a parallel processing computer in an effort to realize faster simulation run times.

The first part of this effort has been accomplished with the associated efforts of two other thesis projects. The first of these is the design and implementation of a graphic user interface which provides the user with a pictorial representation of the circuit design (2). The second project employs an expert system which studies the circuit design and reports problems in IC inter-connectivity, missing or improper connections, and possible oscillation/race conditions (3). All three of these projects have been integrated to produce a comprehensive digital design tool which is expected to be employed in the academic arena.

The second part of this thesis deals with a concurrent construction of the simulator to enhance run time

performance. A parallel model has been engineered and compiled with some improvements noted and directions for future efforts realized. This part of the project utilized the simulator developed for the integrated effort and exploited the natural parallelism contained therein to develop a "first-cut" compilation for the initial analysis.

1.2 Organization

The remainder of this thesis report provides a detailed review of all implementation efforts and analysis of results for those goals outlined in the previous section. It is followed by five appendices which contain supporting data referenced throughout the report.

Chapter 2 contains the thesis introduction and presents an overview of the research goals toward which this project is aimed. In Chapter 3, early work with LOGSIM is discussed, problem areas identified, and re-engineering efforts as related to the integrated design tool implementation discussed. Chapter 4 contains results of work performed towards meeting the requirements of the digital design tool. Methods are discussed and results presented. Chapter 5 discusses the parallel implementation, method of attack, and comparative results. Finally, Chapter 6 offers a discussion of the effort as a whole, lessons learned, and resulting contributions to the computing arena.

Appendix A contains a complete listing of the source code for LOGSIM248, the simulator implemented for this project. Appendix B is comprised of all data flow and functional decomposition block diagrams used during the design phase of the project. Statistical information which compares an older version of the simulator to the new version is contained in Appendix C. Test case documentation is contained in Appendix D. A manual for independent use of the new simulator is included as Appendix E.

CHAPTER 2 - INTRODUCTION AND OBJECTIVES

2.1 Introduction

The development process for any end item is most effective when the item produced performs those functions described by the applicable requirements while conserving resources such as time, money, and materials. Design engineers are successful during the research phase of product development when functional prototypes perform in accordance with requirements. Productive engineering practices must be employed during all phases of the development cycle in order to satisfy these goals.

It would be most effective to build prototypes which are already known to function properly. This way, only those resources necessary to build the minimum number of required prototypes will be expended and delays or cost overruns due to extended research can be avoided. Trial and error methods not only waste time, but are expensive and may expend costly and scarce materials.

Computer simulation of functional end items is a method through which engineers may confirm or expand the operational capabilities of the end product without first building expensive prototypes. Simulators can be used for fault isolation and detection, optimization of design, and confirmation of component interaction in complex designs.

After simulation results have shown that the product performs in accordance with requirements, the prototype may be built with greater assurance of functional success (4:118).

One research area where simulation is most appropriate is in digital circuit design. Here, especially for large, complicated circuits, the design engineer could spend a great deal of time debugging physical prototypes without the aid of computer simulation.

Research and academic environments are two areas where simulations of basic digital designs would aid in the development process. The effects of simulation tools in the research environment was noted earlier in this section.

In the academic world, materials are a precious commodity and simulations which guarantee the proper functioning of digital circuits prior to building prototypes would be beneficial. Additionally, laboratory resources (space, time, etc.) are difficult to schedule on most campuses. The time/space required for building and testing digital circuits can be lessened by using simulation techniques prior to prototype construction. This would free crowded laboratories and allow greater flexibility in academic scheduling practices.

Effective digital circuit simulation tools should be composed of three distinct sub-functions. The first is the user interface. Here the user describes the circuit elements and interconnections. The input streams, output specifications and circuit layout may also be described.

Graphical representation of the circuit is a helpful aid for future conversion to physical prototypes and also allows the user a pictorial view of the circuit. This view can aid in circuit debugging and may later be included as a integral part of a larger circuit.

The second area to be considered in a digital design simulation tool is assuring that the input circuit is correct and consistent. Such problems as input pin to output pin connections, ground to electrical high, disconnected power, and improper fan-out are examples of errors in the design which must be isolated before proper simulation can take place. A routine which checks the circuit design for these problems should be incorporated into the simulation tool to insure proper functionality. Debugging a design prior to simulation allows the user to better understand simulation results. Incorrect or unexpected results may stem from two areas. The circuit may be incorrectly designed, or a connection error may have been made during the construction of the graphic representation of the circuit. The user can be alerted to many design errors through the use of such a connectivity checker, giving a greater assurance of circuit correctness.

Finally, the heart of the design tool is that routine (or group of routines) which actually performs the functional simulation of the circuit. Using the circuit design, input

stream, and output specifications, this routine may perform either logic or timing (or both) simulations by following the circuit path from inputs to outputs and returning output values for a designated series of clock cycles. Logic simulators are used to verify the functioning of a given circuit while timing simulators are used to confirm time relationships between various signals within the circuit.

Various constraints must be realized before designing and implementing a digital design simulation tool. The first and perhaps foremost consideration should be the target audience. A large number of Computer Aided Design (CAD) tools presently exist --each configured for a specific purpose. Tools used to simulate simple circuits designed in an academic setting, by nature of their requirements, differ from those used to simulate Very Large Scale Integration (VLSI) designs used in complex production environments.

The host computer, i.e., the computer on which the simulation tool will be run, is also a major consideration. The screen size, resolution, and underlying graphics hardware affect the graphical representation created by the user interface. The simulation designer must be concerned with expected circuit size, layout, and complexity. Circuit designs may be limited in these areas due to screen, memory, and processing speed constraints. Computer memory could limit circuit size and the necessity to utilize large, complicated, data structures, while the processing speed of the computer could affect the simulator's ability to produce

results in an acceptable time period. Large complicated designs may be impractical to run on small, slow machines.

The user must have at his disposal a wide range of electrical and/or digital components to use in designing circuits. How these components are represented within the simulation program is a major consideration for the programmer. Hardcoded representations of component descriptions and/or operation may improve the speed of the simulation but hamper easy addition of new components to the simulators library. Component libraries stored on disk in retrievable data base format may slow the simulation but are easily updated and may be used universally by all modules of the simulation tool.

Logic simulations of circuits with only a few components may produce results using sequential algorithms relatively quickly. However, the combinatorial explosion apparent in simulations of circuits employing a large number of integrated circuit (IC) packages or electronic components may make sequential simulation routines impractical. Parallel processing computers now being introduced into the computing community may hold the solution to this problem of the Von Neuman "bottleneck."

Methods by which simulation routines are "parallelized" is a research area gaining a great deal of attention. Because digital circuits are inherently parallel in their design it would seem apparent that simulators which capitalize on this parallelism must improve performance and

produce results more quickly. The question gaining researcher's attention deals with the most efficient and practical method to employ in working towards this end.

2.2 Research Objectives

This thesis effort is divided into two separate, but related, objectives. The first objective is the re-engineering and re-design of an existing digital logic simulator for inclusion into an integrated design tool which combines the three sub-functions referred to earlier -- the user interface, the circuit connectivity checker, and the circuit simulator. The user interface and connectivity checker are being implemented through associated thesis efforts. The user interface is being designed as a graphic oriented interface employing the EGA graphics standard (2). The connectivity checker (3) is being designed as an expert system which utilizes Micro Data Base System's GURU expert system tool (5).

The host computer for this integrated design tool (and ultimately for the logic simulator) is an Intel 80286 based Zenith 248 micro-computer utilizing the MS/DOS operating system. However, it is desirable to insure the functionality of both the simulator and the entire design tool on as many compatible (MS/DOS) computers as possible.

The digital design tool is envisioned to be used primarily in an academic environment to design, test, and simulate

basic digital circuit designs which employ TTL IC packages. This tool will allow the student to work a level of abstraction higher than that of most digital design tools. Other available digital CAD systems usually require design work at the gate level. In this project the designer will work with those TTL components commonly found in digital design laboratories. Specifically, the 7400 family of IC packages is used as an initial library of available circuit components. A modest collection of 32 separate packages was identified for the initial IC library. This insured that designs of a general nature could be simulated by the tool while permitting the designers more time to develop system capabilities and optimize the algorithms involved.

The user interface is being implemented as a graphic oriented, menu driven system through which the user creates a pictorial representation of the circuit design using IC graphic icons and interconnecting lines. The design is produced interactively with the aid of a mouse and keyboard input to system produced prompts. After completing the design, the user may invoke either the connectivity checker or the logic simulator through one of the interactive menus (2).

The connectivity checker consists of an expert system which inspects subject circuits for possible illegal or erroneous connections. Connections which are suspect but not necessarily illegal are annotated and presented to the user as 'questionable.' Absent connections are labeled as

'missing.' The results of both this effort and the simulator are presented to the user on separate screens. This expert system and the logic simulator have been designed to function independently or as an inclusion to the integrated design tool (3).

The second objective of this thesis deals with exploration into methods by which this re-designed simulation program may be "parallelized" in order to improve run time performance. The host computer for this effort is a 32 node Intel iPSC hypercube parallel processor in which all or groups of the 32 processors may be used during simulation.

2.3 Summary

Simulation of digital circuits is not a new venture. Many CAD systems exist for the design and simulation of digital circuits. Academicians and researchers are continuously exploring methods to improve digital design and shorten the development cycle. This thesis effort provides a digital logic simulator as part of an integrated design tool which will aid design students in first year college design courses. The design tool will incorporate three functional systems -- a graphic user interface, connectivity checker, and logic simulator. These three sub-systems provide the user with a robust tool which allows the user to produce correct, functional circuits prior to prototype development.

CHAPTER 3 - LOGSIM

3.1 Background

LOGSIM, version 5.5 (6), is a digital design logic simulation program which has been used in the design of simple circuits employing a very limited library of integrated circuit (IC) packages. LOGSIM is unique in that all simulations are performed on circuit designs represented at the IC level. This approach differs from most digital logic simulators which perform simulations of circuits represented at the gate level. This higher level of abstraction allows the user to visualize circuit layout, economize on pin and chip count, and become familiar with commercially available IC packages.

The original program was created at the University of Kentucky by Samuel A. Smith, under the auspices of Frank M. Brown, PhD. This program, though functional, was poorly documented, confusing to use, and difficult to maintain. "The code is an excellent example of a very clever program that was obviously difficult to write but it is equally difficult to read. The original code had only 138 comments out of approximately 2800 lines of code (2:2)." It was implemented in Sheffield Pascal and initially hosted on a PRIME S850 computer (2:2).

In late 1985, CPT Mark C. Rowe from the Air Force Institute of Technology, re-hosted LOGSIM onto a 16-bit microcomputer utilizing the MS/DOS operating system and onto the VAX 11/780 utilizing UNIX. He wrote these new versions in TURBO Pascal (7) for the microcomputer and Berkely Pascal (8) for the VAX, re-engineering the entire project. The end result of CPT Rowe's effort is a more thoroughly documented simulator which is markedly easier to use and maintain. This re-engineered version of LOGSIM (version 5.5), although a usable tool, still contains many problem features making it unattractive to inexperienced users.

First, the user interface is very cumbersome, requiring the circuit designer to traverse a large number of menus to completely specify the circuit. For larger circuits, this process must be interrupted to store sections of the design on disk. Circuit connections are specified on a pin-by-pin basis wherein the user is prompted by IC and pin number and responds with the IC and pin number of the connected component. The user has no means of visualizing this and must complete a detailed design by hand prior to simulation.

Secondly, because of the manner in which pin-to-pin connections are referenced within the program, the number of IC's which may be included in any one circuit design is limited to only 32 of the more basic packages. Specific chip/pin relationships are represented by a five digit integer. This integer identifies the IC and pin number of

the source. The first three positions of this integer indicate the pin number while position four and five depict the IC number. This representation allows the use of modulo arithmetic and integer division to extract IC and pin numbers for inter-package connections. Because of the integer overflow problem with integers greater than 32767, only 32 packages may be included in any one circuit design.

A third problem deals with the limited number of IC's available for inclusion in the design. All IC's used in design simulation within LOGSIM must be contained in the present IC library of 32 standard TTL packages. If the designer wishes to incorporate an IC into the circuit design which is not present in this library, LOGSIM cannot be used. No automated capability presently exists for the user to specify parameters for a new IC or to add new IC's to the existing library. Any additions to the library would have to be hardcoded into the program. The program must then be recompiled, forming a new executable version of the simulator.

Another problem lies in the area of proper connectivity. Upon completion of the design and prior to running the simulation, the circuit is not checked by the program for design defects, such as incorrect or incompatible pin connections, missing inputs/outputs, improper inputs/outputs, etc.. If problems of this sort do exist, the simulation will run with the erroneous circuit configuration, possibly generating incorrect results. Because the simulation may run to conclusion, the user has no way of knowing if the results

are incorrect due to design defects or improper circuit connections specified within LOGSIM.

If the circuit to be simulated has no external inputs, (i.e., the only connections to IC inputs pins are power, ground, and clock) LOGSIM, version 5.5, cannot be used. At least one input stream must be specified for every simulation run. Additionally, it is not possible to monitor power, ground, or clock for output display purposes. Only the values at the IC pins and user supplied input values may be monitored. If power, ground, or clock are connected to an input pin and this pin is monitored, only zeros (logical low) will be displayed for that pin. The actual values of these pins are assumed by the attribute of the pin descriptor field in the circuit node data structure. These descriptors are bound during configuration of the circuit and remain static for the duration of the simulation. This problem limits the user to circuits which have external inputs and prevents the user from displaying all results which may be required for proper analysis of the circuit.

The current version of LOGSIM is a sequential program, designed and written in TURBO Pascal and implemented on MS/DOS (Intel 8086/8088-based) compatible micro-computers. Digital circuits, by nature of their construction, may contain many sections which perform operations concurrently. Large complicated designs, if forced to execute sequentially, may require long periods of time to perform the desired simulation and produce usable results. LOGSIM, version 5.5,

operates relatively quickly for small, simple designs which utilize only a few IC's. However, as the size of the input circuit grows, simulation time appears to increase exponentially.

Finally, LOGSIM, version 5.5, does not run on all MS/DOS compatible micro-computers. Because of memory constraints inherent in the TURBO Pascal (version 3.0) compiler, overlay files were required to allow the execution of the entire simulator. The methods used to compile the implementation of these overlay files cause segmentation errors in Intel 80286-based computers. It is an expressed objective of this project to re-host LOGSIM onto a Zenith 248 which, due to these segmentation problem, does not support the present version of LOGSIM. Re-engineering the simulator with this and other design considerations will vastly enhance capabilities and performance.

3.2 LOGSIM248 Implementation

3.2.1 Overview

LOGSIM248 is a digital design logic simulation program, developed for this thesis effort, which is a new implementation based on LOGSIM, version 5.5. The original source code was written in the C programming language and developed using the Borland International TURBO C integrated development environment (9). TURBO C conforms to the

proposed ANSI standard for C compilers and also contains additional function libraries which enhance it's capabilities. LOGSIM248, however, does not incorporate any TURBO C specific library functions. Portability of LOGSIM248 source code is, therefore, not compromised.

This simulation program may be run independently or as an integral part of the larger digital design tool, IDIET (Interactive DIgital Engineering Tool). If run independently, all files normally created by IDIET's graphic user interface must be created by the user as ASCII files. Because LOGSIM248 performs no checking of input file correctness, all files created as input for the simulator must be formatted in accordance with predefined interface specifications (see Appendix E). When run as an integral part of IDIET, LOGSIM248 is invoked through the graphic user interface. All input files are created by the user interface prior to simulation run and are supplied to LOGSIM248 properly formatted.

3.2.2 Data Structures

LOGSIM248 operates on three main data structures; the circuit configuration list, the input data list, and the output display list. Each of these is implemented as a circular linked list with a dummy header node. This configuration is useful when complete list traversals are

required as the index is left pointing to the head of the list after completion of the traversal.

The circuit configuration list consists of linked nodes, each representing one IC package of the input circuit design. These nodes are modeled after similar data structures in LOGSIM, version 5.5, and contain fields for IC type, IC number, pin types, initial and subsequent pin TTL values, and pin-to-pin connection descriptors.

The input data list consists of nodes which correspond to input source descriptions. These nodes contain two fields -- the input number and the input value of the current clocked input from that source.

The output display list consists of nodes corresponding to user requested output monitor points. Each node is represented by a string of two parts. The first 32 elements of the string consist of a description of the monitor point. The remainder of the string consists of output values derived from the simulation which correspond to the clocked input values.

Four associated data files are required for proper functioning of the simulation program. These files communicate the circuit configuration, input data stream, input port labels, and output monitor points to the requesting LOGSIM248 modules (see Table 1).

(1) TEMP.CKT contains all connections required to complete the circuit design.

(2) TEMP.IN contains input numbers and their corresponding input streams.

(3) TEMP.IND contains the user supplied, two character, input port labels used to identify corresponding input streams.

(4) TEMP.DIS contains a listing of all user requested monitor points to be used in constructing the output file.

LOGSIM248 creates two output data files. These files contain simulation results in two separate formats:

(1) TEMP.OUT contains a description of the requested monitor points in a usable format (different from that of TEMP.DIS) followed by the corresponding output stream of ones and zeros.

(2) TEMP.WAV is divided into two parts. The first is a listing of the input ports (again in a usable format, with user assigned port labels) followed by the input data stream. The second part consists of the output monitor points and associated data streams as described for TEMP.OUT above. The data streams for this file, however, are made up of graphic characters which give the waveform equivalent of the binary data.







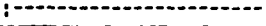

Table 1. Data File Formats (see Table 2. for format codes)		
<i>temp.ckt</i>	-- circuit input file - format: [dnnnttttttpp comment:dnnnttttttpp comment] example: P000 power +5Vdc :T001 740014 power Vcc	
<i>temp.dis</i>	-- output monitor file - format: [dnnnttttttpp comment:] - examples: T001 740003 output pin #3 : I003 input #3 :	
<i>temp.in</i>	-- input data stream file - format: [nnn:iiiiiiiiiii...] - example: 001:101010101010101010101010101010	
<i>temp.out</i>	-- binary output file - format: [IC #nnn (SNnnnnn) PIN #nn :ooo...] [Input nnn :ooo...] [Clock :ooo...] [Power :ooo...] [Ground :ooo...] - examples: IC # 1 (SN 7400) PIN #13 :01101110011110000110011 Input 003 :10011000011100110001101 Clock :01010101010101010101010101010 Power :11111111111111111111111111111111 Ground :00000000000000000000000000000000	
<i>temp.wav</i>	-- waveform output file - input stream description format: [Input #nnn Port cc :iii...] - output stream description format: [IC #nnn (SNnnnnn) PIN #nn :www...] [Input nnn :www...] [Clock :www...] [Power :www...] [Ground :www...] - example: Input file contents: Input #001 Port A1 :  ----- Output data streams: IC # 1 (SN 7400) PIN #13 :  IC # 13 (SN74181) PIN #13 :  Input 003 :  Clock :  Power :  Ground : 	

Table 2. File Format Codes	
codes:	d = connector descriptor (1 character) T : TTL IC I : input port P : power (+5Vdc) G : ground (0Vdc) C : clock n = IC or input number (3 digits) t = IC type (6 digits) optional for input, power, ground, clock p = pin number (2 digits) optional for input, power, ground, clock comment = optional 20 character comment field i = input value, logical value 1 or 0 o = output value, binary logical value 1 or 0 w = output value, waveform logical value 1 =  logical value 0 = -

3.2.3 Operation

LOGSIM248 first opens the circuit configuration file (TEMP.CKT) and reads the circuit connection list one entry at a time. Each entry represents a two ended connection. This may be pin-to-pin, input-to-pin, power-to-pin, etc.. Circuit configuration nodes are initialized and added to the circuit configuration list for each new IC number encountered. Initialization of these nodes includes dynamic allocation of storage for the node, linking of the node to the front of the circuit configuration list, and assigning initial values to all fields of the node data structure. After this initialization has been accomplished, connections are annotated by assigning IC/pin numbers, power or ground indicators, or input port numbers to the appropriate field of the node. If new IC numbers are encountered for the next connection, only the connection field of the already existent IC nodes are updated.

The next step is to initialize the input data list by creating a linked list header, opening the input data file (TEMP.IN), and dynamically allocating storage for each input node indicated by a input number and associated data stream from the input data file. As each input node is created, the input number corresponding to the next entry of the input data file is assigned to the input number field of the node.

As the simulation progresses through the associated data streams the data field of the input node is updated for each clock pulse until all data is exhausted.

Two coded functions are required for each IC type (i.e., SN7400, SN7402, etc.). The first of these functions is used when initializing circuit configuration nodes. This function contains initial values for certain fields of the data structure. These include IC type, pin types, and some initial pin values and connections. All fields not initialized by this function are assumed general in nature and are initialized by a general initialization function.

The second function required for each IC type deals with the operation of the IC. This function is used during the simulation to equate output pin values to existing input values using functional descriptions obtained from TTL data manuals. During each clock pulse of the simulation and immediately following the assignment of all new input values, the operation function related to each IC of the configuration circuit list is called in turn and the new values of the output pins assigned to the corresponding fields of the appropriate circuit nodes. This is repeated until two consecutive operations of the circuit produce the same output values. At this point the clock can be advanced and the next data set evaluated. Failure to produce two consecutive constant output pin value sets after a specified

number of iterations causes termination of the simulation. The user is notified that the circuit is in an oscillation condition and is presented with the clock pulse containing the errant data set.

The final function of the simulator is to produce the two output files discussed in section 3.2.2. After each set of input values (for a particular clock pulse) is simulated, and prior to moving on to the next set, the circuit configuration list is searched for those pins identified in TEMP.DIS. These are the pins which the user identified as monitor points for inclusion in the output files. After locating these points the corresponding values are appended to the appropriate output stream in the output display list. After the last set of input values has been simulated, the output display list is used to create TEMP.OUT and TEMP.WAV.

3.3 Summary

Although a usable tool, LOGSIM, version 5.5, was in need of a major overhaul. Problems in software design, portability, and performance limited its' use and impaired its' maintainability. LOGSIM248, as will be seen in the next chapters, has overcome these problems, and presents a more efficient and usable tool. Re-designed data structures and a new data interface have allowed the integration of LOGSIM248 into a total digital design tool while improving maintainability.

CHAPTER 4 - SOFTWARE DEVELOPMENT

4.1 Re-engineering Overview

The re-engineering effort which resulted in the design and implementation of LOGSIM248 proceeded in five distinct phases;

- (1) analysis of LOGSIM version 5.5
- (2) design, LOGSIM248
- (3) coding, LOGSIM248
- (4) integration of LOGSIM248 into IDIET
- (5) testing and debugging, LOGSIM248 and IDIET

4.2 Analysis - LOGSIM, version 5.5

The analysis of LOGSIM, version 5.5 was divided into two separate parts. The first part consisted of a "hands-on" approach which was necessary to become familiar with user interaction and program performance. Several small digital circuits were used to accomplish this objective. These circuits consisted of one or two IC's which performed very basic functions and are of no consequence here.

The second part of the analysis involved an in-depth study of the simulation source code and accompanying documents. This proved to be a time consuming effort due to the length of the source code and those software engineering shortcomings discussed in Chapter 3.

During the "hands-on" familiarization with LOGSIM, version 5.5, most problems encountered pertained to "ease of use." These problems included mastering the menu driven user interface, visualizing the overall circuit layout, dealing with input inadequacies, and interpreting output data. A short discussion of each of these problems follows.

In order to build the circuit, specify input data and output format, and run the simulation, the user is required to navigate through a myriad of menus which offer options pertaining to these operations. After spending some time working with the program it becomes apparent that these menus are related in a tree-like manner. Movement through the menus consists of no more than moving up and down this tree. Although it is easy to get lost, upward movement (i.e., repeatedly exiting the present menu) will eventually return the user to the root where circuit manipulation and simulation may be started again. After running a number of simulations, the user should be able to learn this menu mechanism well enough to accomplish his objective. However, the primary audience envisioned for LOGSIM248 is the academic (collegiate) arena. Here students will be expected to

quickly master the various tools of the laboratory and may not have the necessary time to become thoroughly familiar with the complexities of this intricate menu system.

Another difficulty noted during the "hands-on" phase was the inability to visualize the circuit during construction. The user is first prompted for an IC package type and identification number, after which, more prompts for all connections to this package are presented, pin by pin. After entering data for a particular chip, the user may start over at the first pin of the present chip or request chip data previously entered. Here he may either check or correct circuit data. This accomplishes the objective of configuring the circuit, but makes it difficult to visualize the circuit as it is created. If the user were able to see a graphic image of the circuit design as it was being created, comparisons could be made directly to design support material to insure correct entry of circuit data and to visualize IC layout and pin orientation.

Simulation output also has a presentation problem. All requested output data are presented in columnar fashion. Each clocked output data set appears on consecutive rows in the display, resembling a truth table. These data sets may include input streams, output streams, and/or any other pin values, all of which may be viewed at the same time for a particular clock pulse.

For some circuit designs, the individual input/output data streams are better viewed with the data presented from

left to right. This is especially true when performing various waveform analyses. Some sort of waveform display option would give the user the ability to see input/output relationships in a different, and, perhaps, more usable format.

It was during this phase of the analysis that circuit input limitations were discovered. After designing a three stage binary ripple counter which only requires power, ground, and clock values as input to the circuit, LOGSIM, version 5.5, would not allow the simulation to proceed because no external input streams had been supplied. Additionally, the program provided no apparent means of presenting power, ground, or clock values as part of the output display. This problem, discussed in Chapter 3, prevents the user from comparing output values with the system supplied clock, and requires some sort of external input even for those circuits which do not require it.

This "hands-on" phase of the LOGSIM, version 5.5 analysis showed the simulator to be a useful, but sometimes confusing tool which presents simulation output in a manner that could prove difficult to analyze by the less experienced designer (i.e., first year design students).

The second part of the LOGSIM, version 5.5 analysis consisted of the inspection and analysis of the source code produced by Captain Rowe (6). Although some documentation did exist, a great deal of time was required to organize,

analyze and understand the various modules which comprised the simulator program.

The following are the major software engineering deficiencies discovered during this phase:

- (1) A 15 page report and 12 pages of block diagrams are all the design documentation that accompanies over 4,500 lines of code. Although each procedure and function contains a header block listing a small amount of information about the function of that procedure, little to no comments can be found in the code.
- (2) Some procedures contain as many as 500 lines.
- (3) Block indentation is inconsistent creating confusion in source code structure.
- (4) Global coupling is prolific throughout the program. This type of coupling requires a large amount of globally declared variables and wasted memory space.
- (5) Because TURBO Pascal, version 3.0, allows executable files to occupy no more than 64 Kbytes of storage (due to 16-bit addressing limitations), and because of the size of the simulation program, overlay files were needed to allow the entire program to be compiled and run within the 64 Kbyte limit.

Because of these deficiencies, maintenance, modifications, and/or upgrades to this source code would certainly be difficult. The need for re-engineering was apparent, in spite of the requirement to integrate a re-engineered simulator into IDIET. Toward this end, all user interface modules needed to be stripped from the existing code, data structures rebuilt to accommodate the IDIET interface, all modules translated into the C programming language, and a great deal of documentation supplied.

The C programming language was chosen for a number of reasons. The user interface to IDIET is also written in C in order to use the underlying graphics package. For uniformity and future maintenance/upgrade reasons, C was also chosen for LOGSIM248. As stated earlier, the Pascal version of LOGSIM was not entirely compatible, i.e., this version was not supported by Intel 80286 micro-computers (Zenith 248, IBM AT, etc.). By using the C programming language (in particular TURBO C) compatibility among more computers was expected. In addition, the Intel iPSC Hypercube does not support any version of Pascal, but does support the C programming language. In order to complete the concurrent version of LOGSIM on this computer system, re-hosting the simulator in C was necessary.

4.3 Design - LOGSIM248

LOGSIM248 easily decomposes into four distinct functional areas:

- (1) circuit configuration
- (2) input data structure configuration
- (3) circuit simulation
- (4) output file construction.

LOGSIM248 has been designed with these four functional areas as the initial decomposition. Four input files (see section 3.2.2) are created by the user interface module of IDIET and are used to communicate data between the user interface and the simulator. These four files are the only input or control structures required by LOGSIM248. Output consists solely of two output data files. These two files are supplied to the user interface for screen presentation. Data flow diagrams and functional decomposition block diagrams illustrating the design methodology for LOGSIM248 are contained in Appendix B. The data flow diagrams (SADT diagrams) depict information flow and the transformations that are applied as data moves from input to output (10:99). These diagrams decompose the simulator into the various activities which occur during program execution. The functional decomposition block diagrams show the actual decomposition used while writing the source code. Data flow

diagrams help to visualize those transformations that input data undergoes during program execution. These diagrams are immediately translatable into the functional decomposition.

As discussed in Chapter 3, three data structures are the primary objects used by the simulator. Each of these is a circularly linked list configured with a dummy header node. These data structures are used to store and maintain the circuit configuration data, input data, and output data. The linked lists used in LOGSIM248 differ from the corresponding data structures used in LOGSIM, version 5.5, in three ways:

- (1) All three lists are circular. Resetting an index to the beginning of the list is not required for iterative processes that repeatedly traverse the entire list. The index is properly positioned at the beginning of the linked list after completion of each iteration. LOGSIM, version 5.5, uses nil terminated lists that required re-initialization after each traversal. This may not be time effective -- especially for those procedures that must traverse the linked lists repeatedly.
- (2) LOGSIM, version 5.5, is designed to build an input linked list early in the simulation which contains the entire input stream for all inputs specified by the user. This requires the use of a

great deal of memory (especially for those simulations which required large amounts of input data) due to the manner in which TURBO Pascal allocates dynamic memory. If the input data streams are large, and the circuit configuration linked list (already created) is also large, internal memory could easily be exhausted before all data structures are completed. If memory does become exhausted, LOGSIM, version 5.5, contains a routine which stores incomplete data structures on disk. When the simulation is run, this incomplete information must be swapped in and out of memory, as needed. This system does work, however, it is obvious that a time penalty for this file I/O will be incurred.

LOGSIM248 uses a different strategy for input data. As the simulation progresses only the data-set required for the clock pulse currently being simulated is stored in memory. This input data linked list is updated with new data for each clock pulse of the simulation. Here, less memory is used and the traversal of the present input data set linked list requires less time.

- (3) The output data list in LOGSIM248 is also a linked list which contains the output monitoring points (as specified for TEMP.DIS in Table 1)

followed by the output data stream. This data stream is built one element at a time as the simulation progresses through each clock pulse of the input data list. After the simulation is complete, the monitor points are reformatted (as shown for TEMP.OUT and TEMP.WAV in Table 1) and the reformatted output points are written to both TEMP.WAV and TEMP.OUT followed by the output data stream (in waveform and binary format, respectively).

LOGSIM, version 5.5, writes each output data set to the requested I/O device (disk, screen, printer) as it is created. A large time penalty is incurred, especially when writing output data to a disk, due to disk spin-up and seek time.

All coupling between functional modules has been improved. Most modules are simply data coupled. The problem of global (or common) coupling, discussed earlier has been almost completely eradicated. Only three global variables are used, each of which is a pointer to the three data structures explained above. These global variables are available for manipulation by any module in the program.

Another problem of LOGSIM, version 5.5, noted above, is the length of the procedures. Many of these procedures exhibit cohesion problems. Too many functions are

accomplished within one procedure making it difficult to understand the purpose of each module. All of the functions defined by the decomposition for LOGSIM248 are designed as single-function modules. This functional form of cohesion enhances understanding of each module and improves maintainability.

These improvements in both coupling and cohesion are evidenced in the functional decomposition illustrated in Appendix B. Excepting the three global variables which are used throughout the program to point to the three linked lists, all data required for each module is passed when that module is called. The arrows associated with links between blocks illustrate both control (solid head arrows) and informational data which is passed between functional modules. This is indicative of good data coupling. As can be seen, all modules present this form of coupling, or (in the case of no passed data) no coupling at all. The improvements in functional cohesion (i.e., each module performs one function) is indicated by the module names, but is better seen in the source code itself (Appendix A).

4.4 Coding - LOGSIM248

All source code for LOGSIM248 is contained in six separate files (see Appendix A). These files can be directly correlated to the decomposition referred to in the previous section and are organized in a functionally cohesive manner.

- logsim.h: This is a header file which contains all constant declarations and data structure type declarations used in the program modules.
- simlib.c: This file contains low level functions used throughout the program to perform parsing, data type conversions, random number generation, etc..
- exec.c: This file contains the function main() and all debugging functions. This file also contains the error handler used throughout the program.
- makeckt.c: This file contains all functions associated with the construction of the circuit configuration linked list. Here the TEMP.CKT file is read, all circuit connections parsed from this file, and all circuit information plus initial pin values stored in the circuit configuration list.
- simio.c: This file contains those functions associated with the initialization and maintenance of all input and output data structures. Additionally, those functions used to construct the output files, TEMP.OUT and TEMP.WAV, are contained in this file.

- `srun.c`: This file contains the heart of the simulator program. Here pin values of those IC packages contained in the circuit configuration linked list are updated for each clocked input data set, after which each IC is "operated" to produce new output values at the output pins. Oscillating circuit conditions are also detected here.
- `config.c`: This file contains all chip configuration functions and an executive function which chooses the appropriate configuration function. These functions are called by the `makeckt.c` module upon initialization of a new node in the circuit configuration linked list. These nodes represent a single IC package in the present circuit design. After adding the node to the list, each field of the structure is bound to initial values for the corresponding IC by functions in this file.
- `opic.c`: This file contains all those functions required to operate the IC packages used in the circuit configuration list. These functions perform the appropriate boolean, arithmetic, and other assorted TTL operations for those IC's presently contained in the simulator IC library.

All modules included in the above source code files (with the exception of configic.c and opic.c) contain an average of nine functional statements apiece. Because these modules are functionally cohesive they are designed and coded to perform one function. The modules used for IC configuration and operation also perform a single function. However, due to the requirement for functional completeness, some of these are substantially longer.

Optimization of source code was attempted, wherever possible. Register variables were used for counters and loop indices to insure rapid incrementing. Increment and decrement operators (++ and --), and pointer arithmetic were used wherever possible to speed up these operations. To insure greater maintainability, a great deal of modularization was achieved. This was expected to detract from the overall speed of operation due to the stacking operations of local variables at each function call. Statistical analysis of the comparative speed of LOGSIM248 to LOGSIM, version 5.5, however, does not bear this out (see section 6.1).

4.5 IDIET Integration

After successful compilation and limited testing of LOGSIM248, integration of the simulator into the digital design tool, IDIET (Integrated Digital Engineering Tool)

became the next goal. Here, not only were design and code considerations important, but insuring a productive working environment for all project designers became the dominant theme. Prior to the start of the individual component projects, the three project designers (graphic user interface, connectivity checker expert system, and simulator program designers) agreed on a set of specifications for all interface requirements. Due to unforeseen difficulties during each individual's design and coding phases, these specifications sustained a small amount of modification. Through good communication of modified requirements, all interface changes were effected painlessly. The resulting specifications are contained in the file formats for those files listed in Table 1.

At this point it was decided that the production of a waveform output (TEMP.WAV) would be more easily accomplished by the simulation program and the required additions to LOGSIM248 were incorporated into the source code file containing all I/O modules (simio.c). Additionally, due to memory constraint problems with the host computer, it was not possible for all three programs to remain resident at the same time. Therefore, a small executive program was created to run the graphic interface and simulation components independently of the expert system.

4.6 Testing and Debugging - LOGSIM248

As with any software engineering project, finding test cases which test the proper conditions and provide results which can be used to evaluate program performance was a difficult task. The test cases used were grouped into three separate classes.

The first class consisted of circuit designs used during the development phase of the project. These designs needed to be simple enough so that time was not wasted implementing them but large enough to insure some measure of proper performance.

The second class of test cases consisted of those circuit designs implemented during the testing phase of the integrated design tool, IDIET. These cases consisted of circuit designs which taxed the power of the simulator and provided measurable performance indicators used for debugging and performance enhancement.

The third class of test cases consists of circuit designs created by those students of the Air Force Institute of Technology chosen to field-test the digital design tool as a whole. These circuit designs, along with the inexperience of the users, were expected to provide a myriad of insights into software design flaws for all aspect of the design tool.

Because of the combinatoric explosion involved with testing circuits which use all IC's contained in the IC library, more emphasis was placed on the execution of those

modules involved in circuit configuration, input/output specification, and simulation execution. All functional modules (excepting those modules contained in config.c and opic.c) were completely tested using dummy stubs at each stage of the coding phase.

Because of limitations of the graphic user interface, circuits comprised of more than 10-12 IC packages are difficult to create with IDIET. The size of the screen prohibits larger circuits with many connections. As the circuit grows, the screen takes on the appearance of a "rat's nest" and becomes impossible to sort out connections for debugging or modification. Subsequently, the four test cases used during the IDIET testing phase contain 3-6 IC packages with 20-75 connections between packages. This proved to be a manageable circuit size through which productive program debugging could be accomplished.

The four test cases used are (1) a binary coded decimal encoder, (2) a two-bit full adder, (3) a three stage binary ripple counter, and (4) a 3 by 8 decoder. Schematics and graphic screen images of these circuits are contained in Appendix D.

The Digital Circuit Design class of the Air Force Institute of Technology was chosen to perform user "hands-on" testing of IDIET. Their first assignment included the design of a Binary Coded Decimal to Excess 13 encoder. All students

easily mastered the design tool, producing results which either confirmed their designs or indicated circuit flaws. No appreciable negative comments were made with regards to LOGSIM248. More user testing is forthcoming and will be used to upgrade and/or maintain the system.

CHAPTER 5 - CONCURRENT LOGSIM

5.1 Overview

Although applying concurrent processing techniques to a digital circuit simulator like LOGSIM248 would appear to be advantageous in producing improved performance, other factors peculiar to this implementation need to be considered. The most practical way to produce LOGSIM248 test cases which can be verified as correct is to use the integrated design tool, IDIET. After configuring and simulating a digital circuit, all relevant files (containing circuit configuration and input/output data) used during the simulation may be saved for further study. These files contain all data required to recreate the circuit, provide input and output specifications, and simulate circuit performance (see Chapter 3). It would be impractical to compare the performance of the LOGSIM248, divorced from IDIET, to a "parallelized" version designed on a concurrent processor due to those difficulties involved with independent use of LOGSIM248 (see Chapter 6).

IDIET, however, presents some limitations which may devalue the implementation of a concurrent LOGSIM248. The most obvious of these limitations is the inability to produce circuits (using IDIET) large enough to tax the speed of the

host computer -- the Zenith 248 (see section 6.1). Test cases designed with IDIET, which were constructed from only six IC packages, began to clutter the graphic design area of the screen. With careful planning, and judicious use of the graphic design area, circuits consisting of nine or ten separate IC packages represent the upper limit of workable circuit size using IDIET. However, based on the statistics produced for test cases used during program implementation (see section 6.1), it is not envisioned that circuits of this size would produce time delays of any consequence. It is obvious from those statistics, however, that simulation speed is inversely proportional to circuit design size. Thus far, circuits designed using IDIET have not slowed the system. So, how slow is too slow?

The primary purpose of concurrent processing is to reduce the time complexity limitations of sequential computers for large, CPU intensive problems by employing multiple CPU's, all working concurrently on decomposed areas of a compound problem. As stated in Chapter 2, digital circuits are, by their very nature, parallel constructs that would appear to conveniently lend themselves to concurrent applications. However, the trade-off between time savings and computing cost must be considered, especially if no inconvenience is perceived by a sequential implementation. At this writing, IDIET stands as an integrated system whose capabilities are limited in such a way as to present no such time inconvenience.

Looking at the problem from another perspective, a particular simulator program design and implementation may limit concurrent applications. Digital circuits are inherently parallel in design, however, the algorithm and data structures used to simulate these circuits may not be. LOGSIM248, and it's predecessors, were designed with this structural limitation, at least insofar as re-hosting the simulator on the Intel iPSC Hypercube is concerned. As detailed in Chapter 4, three linked list data structures are used to completely describe the circuit configuration, input data sets, and output specifications. Communication between the cube manager and the individual cube nodes of the iPSC, as well as, communication between the cube nodes themselves, involves passing all data structures which are to be processed in the nodes. For a single node of the cube to perform operations on all, or even a significant section, of the subject circuit, a great deal of time consuming disassembly and assembly of the basic data structures would be required.

There is no practical way of passing an entire linked list to a cube node for processing. This must be accomplished by disassembling the linked list into separate entities which are equivalent to the individual nodes of the linked list. Then these separate entities (C language structures, in the case of LOGSIM248) must be individually passed to one of the cube processors, requiring a separate inter-nodal message for each. Once at the node, these

individual structures must be re-assembled as a replication of the original linked list prior to processing. The time involved in disassembly and re-assembly alone would render concurrent processing ineffective. The added time required for multiple message passing would devastate any advantages gained by parallel operations.

5.2 Parallel Approach

Because future improvements to IDIET's user interface could alleviate circuit size limitations, concurrent programming techniques for LOGSIM248 should be explored. Although the sequential implementation of the simulator presents problems in decomposition of data structures for concurrent applications, there are sections of the program which might be exploited. During the simulation, the circuit configuration linked list is traversed and each node representing an IC within the circuit is visited. This is when the functions which operate the individual IC's are called and output pin values for the visited IC changed to reflect the functional relationship to all inputs. Each IC is operated independently of the others. However, because the linked list is traversed in a sequential fashion, each IC is also operated sequentially. Here is a good example of a situation in which a parallel approach might save some time.

The parallel approach used for this part of the project will involve the operation functions associated with each IC

type. During the simulation module (see section 4.4, `srun.c`) function calls are made to function `operate()` which in turn, calls the module which contains all IC operation functions. The function calls to `operate()` are made for each input data set, updated after each clock pulse. Function `operate()` disassembles the configuration linked list and independently passes each IC of the circuit to the operation module. Function `operate()` is blocked until control is returned from the operation function called for each IC. It is possible here to pass each independent IC data structure to a different processor wherein they will be operated concurrently. After completion of each individual IC operation, the process which accomplished the operation can pass results (the modified IC data structure) back to the main process. Here, the function `operate()` will re-assemble the circuit configuration list with the IC structures containing new output pin values. Although this is a small portion of the entire program, success in this initial design may be used as a model for implementing similar concurrent structures throughout the simulator.

5.3 Implementation

The implementation for this concurrent approach consists of two main concerns. The first involves the construction of a program which contains a new executive module and the IC operation modules extracted from LOGSIM248 (`opic.c`). This

program, when compiled and properly loaded, resides on all nodes of the hypercube allowing the simulator to send IC structures to as many as 32 different processors. The executive module of these node processes receives a message containing an IC structure for processing from the manager process (residing on the cube manager). After this IC is operated upon by the appropriate operation function, the executive module returns the updated IC structure to the manager process in a new message.

The program residing in the cube manager (the processor used to manage all node processors) is essentially the same as that used in the sequential version of LOGSIM248. The file that contains all IC operation modules (opic.c) however, is not needed here as all those operation functions are used only by the individual node processes. The call to the IC operation functions (which resides in the function operate() in the sequential version of LOGSIM248) has been replaced by a call to the function sendmsg() which sends one of the IC structures from the current circuit configuration list to a node process. The node process is blocked until it receives one of these messages from the manager.

The first approach taken disassembled the configuration linked list in the cube manager, sent the individual IC structure to the appropriate cube node for processing, and awaited the receipt of the updated IC structure back in the cube manager. As each IC structure was returned to the cube manager a portion of memory was dynamically allocated and a

the configuration linked list recreated. This approach failed for large circuits or circuits with a large amount of input data. These circuits required a great deal of dynamic memory due to the repetitive recreation of the configuration linked list or recreation of large linked lists. As this dynamic memory allocation was repeated memory would become exhausted and the program would abort.

A more successful approach was implemented in the cube manager by sending an image of each IC data structure to a node process. When the updated structure returned, the updated fields of this structure were copied into the original structure still contained in the configuration linked list. This kept memory from becoming exhausted but suffered a time penalty for the copy operation. Performance of this concurrent implementation is discussed in Chapter 6.

5.4 Summary

Because of the limitations that IDIET places on the circuit size of design circuits created by the design tool, combined with the difficulties in converting LOGSIM248 to a concurrent implementation, it would not appear as though anything is to be gained by this effort. However, due to the expectation of improvements to the integrated design tool, IDIET, some exploration into a concurrent implementation should be attempted. A "first-cut" design has been implemented wherein the individual IC's of the design circuit

are removed from the circuit configuration list and operated concurrently. A performance evaluation and comparison to the sequential version is presented in Chapter 6.

CHAPTER 6 - DISCUSSION

6.1 Comparing Performance - LOGSIM V5.5 vs. LOGSIM248

Appendix C contains statistical information comparing the run time performance of LOGSIM, version 5.5, to that of the new LOGSIM248. Because of the inability of LOGSIM, version 5.5, to run on the Zenith 248 micro-computer, all times used for these comparisons were generated by running both simulation programs on a Intel 8088-based, Leading Edge Model D micro-computer operating at 4.77 MHZ. Due to the improved portability of LOGSIM248, run times generated on the Zenith 248 are also presented, providing additional data which helps analyze performance improvements.

All run time samples reflect only the time elapsed for the circuit simulation and file output modules of each simulator. Because of the major differences in processing implementation for circuit configuration and input/output specification, any run times which incorporated these parts of the simulators would not reflect equivalent performance parameters.

Run time samples for each of the test cases described in section 4.6 are contained in Appendix C. These samples represent multiple simulations of each test case using the

same input data for each run. As can be seen, LOGSIM248 sample times are consistently similar, as opposed to those of LOGSIM, version 5.5. This inconsistency among the simulation run times of LOGSIM, version 5.5, is attributed to the manner in which circuit output is written to the output file. This simulator writes to the output file as each output data set (i.e., one bit for each output monitor point) is computed. Seek times combined with disk spin-up contribute to the variance in the samples recorded in Appendix C.

LOGSIM248 uses a different method for circuit output. As the output data sets are computed, they are stored in the output display list described in section 3.2.2. All resultant output data streams are then formatted and written to the output files upon completion of the simulation.

The first task of developing reliable statistics for this comparison was to determine the proper number of simulation run time samples which insured that the sample mean did not deviate, within a reasonable tolerance, from the true mean. A tolerance of 0.1 seconds was chosen in an effort to keep the required number of samples from being too great. A sample size large enough to guarantee, at a 95% confidence level, that run times exhibit a normal distribution was the desired result of the first test. Using the equation for the Student t distribution (best used for sample sizes from 0 to 100) yields the formula:

$$T = \frac{\bar{X} - \mu}{\sigma / \sqrt{n}}$$

where $\bar{X} - \mu = e$

e

(or error tolerance within +/- seconds) represents the allowable deviation of the sample mean from true mean.

σ

(true deviation) is replaced with the sample deviation to produce the equation:

$$T = \frac{e}{s / \sqrt{n}}$$

which may be rewritten to produce the equation for required sample size:

$$n = \left[\frac{Ts}{e} \right]^2$$

Appendix C contains three separate reports for each of four test cases. These test cases are:

- a) BCD.CKT - Binary Coded Decimal encoder
- b) ADDER.CKT - 2-bit full adder
- c) BRC3S.CKT - 3-stage binary ripple counter
- d) DECODER.CKT - 3X8 decoder

The first report was generated using the S statistical package (11), and contains information derived from the formula above. In the case of BCD.CKT, for example, 50 simulations runs were performed. LOGSIM248 produced a mean run time of 4.174399 seconds, with a standard deviation of .02459135 seconds. Only 0.1700558 (or one run) was shown to be required to guarantee that the deviation from the mean did not exceed the prescribed tolerance (at a 95% confidence level). However, because of a wider variance in the sample data for LOGSIM, version 5.5, at least 35 runs were required to guarantee the same deviation limit. The three other test cases can be evaluated in the same manner.

Being confident that the sample distribution conforms to the normal distribution and having determined that the required sample sizes lie between 0 and 100 run times (with a deviation lying within prescribed tolerances), a paired t-test can be used to test the hypothesis that LOGSIM248 performs faster than LOGSIM, version 5.5. The second report for each test case in Appendix C uses the SAS statistical

package (13) to perform such a paired t-test on the first two columns of the sample data list (supplied in the third report for each test case). This SAS report also displays the mean and standard deviation for each sample. The difference in the two sample means is computed and t-tests performed. The label ``PR>|T|'' shows the probability of the difference occurring by chance. For all four test cases this probability was 0.01%. In other words, this test showed that there is a 99.99% probability that the difference in the sample means was due to LOGSIM248 performing faster. In fact, as the number of IC packages in the design circuit increases (for those test cases shown), so does the run time difference, as shown in Table 3. When hosting LOGSIM248 on the Zenith 248 an even greater increase in run time is noted.

Table 3. Simulation Run Time Comparison				
TEST CASE	LOGSIM V5.5 Leading Edge	LOGSIM248 Leading Edge	LOGSIM248 Zenith 248	# IC's
BCD.CKT	6.2114 sec	4.1744 sec	1.4598 sec	3
% increase		148%	425%	
BRC35.CKT	6.3808 sec	3.1646 sec	1.8072 sec	3
% increase		201%	353%	
ADDER.CKT	15.388 sec	6.7107 sec	2.0900 sec	4
% increase		229%	736%	
DECODER.CKT	21.617 sec	8.7805 sec	2.8257 sec	6
% increase		246%	765%	

These performance improvements may have been produced by a variety of variables. As discussed in Chapter 3 and 4, the main data structures in LOGSIM248 have been improved and source code optimized. However, the noted improvement in performance is probably due more to compiler optimization than source code optimization. Even with this consideration, the objective of improved performance in simulation run times has been realized.

6.2 Concurrent LOGSIM Performance and Recommendations

As noted in the previous chapter, the algorithm and data structures utilized in the design of LOGSIM248 do not easily lend themselves to a concurrent implementation. The specific implementation chosen produced rather disappointing results in terms of run time performance. However, the lessons learned from this effort can hopefully be used to improve on implementations used in the future.

Table 4. shows a comparison of the mean run times for four different simulation implementations. Here the mean run times for LOGSIM, version 5.5, and LOGSIM248 gathered from simulations implemented on the Leading Edge Model D can be compared to those mean run times gathered from both the Zenith 248 and the iPSC Hypercube. Two separate times are presented from the Hypercube. The first (labeled Intel 286/310) gives the mean simulation run time for an implementation of LOGSIM248 run entirely on the cube

manager. None of the node processors were used for this implementation. The mean run time shown in the table provides a basis of comparison for any concurrent implementation used. If, in fact, this time is faster (and can be shown statistically to be faster) than that of a concurrent implementation, then there is reason to believe that no improvement has been made. This is the case for the concurrent implementation used for this project.

Table 4. Mean Run Time Comparison for Concurrent Implementation			
Test Case	LOGSIM248 Zenith 248	LOGSIM248 Intel 296/310	LOGSIM248 Intel iPSC
BCD.CKT	1.4598	0.8452	2.1940
BRC3S.CKT	1.8072	0.5232	1.9291
ADDER.CKT	2.0900	1.7217	4.1557
DECODER.CKT	2.8257	2.5115	7.6036

The primary reason for this lack of improvement in run time performance lies in the communication delay between the cube manager and those node processors used by the simulator. In the concurrent implementation used, only one IC data structure from the circuit configuration list is sent to a node processor per message (communication from the manager). The time required to execute the operation

function used by any one IC data structure is significantly less than the communication time required to send that data structure from the cube manager to the node processor. A proposed solution to this time disproportion may involve decomposing the circuit configuration list into groups of IC data structures. These groups of data structures could be placed into a contiguous array and then passed to the node processor. This would require the node processor to execute a number of operation functions, one for each IC data structure in the array. By decomposing the circuit configuration list into different numbers of IC data structures the optimum implementation could be found wherein the communication problem is overcome while the power of the concurrent machine is exploited.

This proposed implementation, however, is constrained by the size of the circuit to be simulated. If the number of IC data structures required for efficient use of the node processors is large, then circuits which require few IC packages in their overall design cannot be simulated concurrently with any expectation of improved performance over sequential implementations. Presently, as discussed in Chapter 5, the means for constructing large circuits (using the integrated design tool, IDIET) does not exist. As IDIET's graphic user interface is improved and circuits consisting of a larger number of IC packages can be configured, then research in this concurrent implementation can be initiated.

The concurrent implementation described above, as well as the one actually implemented, only deals with a small part of the entire simulation program. Other functional areas of the program exhibit the potential for concurrent operations. During creation of the configuration linked list, each new IC type encountered in the circuit input file (TEMP.CKT) requires two operations. The first operation creates a structure containing fields for IC type, IC number, pin types (input, output, clock, Vcc, etc.), pin connections, and pin values. The next operation initializes these fields to reflect those values which are required for simulation but may not be altered by the simulation (such as the indication that a certain pin will have a clock input). For larger circuits these operations could be parceled out to the node processors of the iPSC Hypercube in much the same way as described above.

The input data list and the output display list are constantly updated and maintained by various functions within the program. These functions may also be exploited in an effort to further "parallelize" the simulator.

6.3 Using LOGSIM248 Independently

LOGSIM248 was essentially designed as an integral part of the digital design tool, IDIET. As an integral part of this tool, LOGSIM248 performs in its' optimal environment, is assured of correct input via the input file interface, and

requires no user involvement to accomplish its' task. However, LOGSIM248 may be operated in the absence of the other components of IDIET, provided the user has an in-depth knowledge of the input file interface. It is not the purpose of this section to educate the potential user in the independent operation of LOGSIM248. However, a few points pertaining to this capability should be mentioned.

Because LOGSIM248 was designed to accept input from three ASCII files created by the graphic user interface component of IDIET, the assumption is made throughout the simulation program that all input information is correct and formatted in accordance with the agreed interface specifications. The only error checking performed by LOGSIM248 is that of file content and availability. If the required input files do not exist or contain no information, LOGSIM248 enters the appropriate error message into two output files normally containing simulation results. All other error checking is done at the user interface level prior to the creation of the interface files. To replicate this error checking would partially invalidate the requirements for the user interface, degrade performance, and waste precious computing space. It should be noted here that any improperly formatted file information may cause improper output data to be generated, and may even cause program abortion.

LOGSIM248 utilizes input information obtained from the three files TEMP.CKT, TEMP.IN, and TEMP.IND. All output produced by the simulator is written to TEMP.OUT and

TEMP.WAV. After each simulation run it is imperative that the user rename or copy these output files. During the next simulation run these files will be destroyed prior to re-writing them with new output information.

As a sideline to those difficulties outlined above, independent use of LOGSIM248 requires some sort of file editor to produce the input files used by the simulator. Of course, this means that all users must be familiar with an editor capable of producing ASCII files.

6.4 LOGSIM248 IC Library Expansion

LOGSIM248 contains a library of 32 SN7400 family TTL Integrated Circuits. This library is hard-coded into two source code files, configic.c and opic.c (see Chapter 4 for an explanation of the operations of these files). Each IC requires two functions, one to initialize the chip and one to operate it. These functions are located, one each in configic.c and opic.c. In addition to these two functions, each IC requires a case statement in each file which is used to call the appropriate IC operation or configuration function. In configic.c this case statement is located in the module catalog(), and in opic.c, module icfunction().

Difficulty in upgrading this library has been significantly reduced due to the program structure. All that is required to add a new IC to the simulator's library is the creation and addition of two functions -- one to configic.c

and one to opic.c. Additionally, the appropriate case statement must be added to the functions described above. This in itself is not an improvement. However, because the C programming language allows source files to be compiled separately, only these two file need be re-compiled. Because object code files exist for all other source code files, these may be linked to the newly compiled files without re-compilation. By separating all IC functions from the control functions of the simulator some upgrade difficulty has been alleviated.

6.5 Future Directions

Realizing the full potential of LOGSIM248 relies on upgrades to both the simulator and the integrated design tool, IDIET. Presently, LOGSIM248 works efficiently on those small circuit designs which can be create through IDIET, but lacks the challenge of larger, more complicated circuit simulations. When the graphic user interface gives the user the ability to input large complex digital circuits by incorporating a pan and zoom function within the graphics package, then simulations on circuits which tax the full capability of LOGSIM248 may be realized. To this end, concurrent processing of these large circuits may prove beneficial in reducing the time complexity of the simulation program.

Even though it is somewhat easier to upgrade the IC library with those design changes incorporated into LOGSIM248, the major difficulty is the actual construction of the functions which must be added for each new IC. Some of the chip operation functions (opic.c) are very involved and require in-depth knowledge of the functions and hardware involved. This difficulty, when added to the annoying problems of re-compiling source code, cause library upgrades to be burdensome and, most likely, very complex.

A future addition to IDIET in the form of a library expansion routine would afford those users who require more design power (but have neither the desire or know-how to re-write the source code) the ease of library upgrades. This could possibly be implemented by presenting the user with a function template equating input to output pins, prompting for gate level functions, inquiring about previous flip-flop conditions, etc..

For users not capable of installing the entire IDIET tool due to computer system configuration problems (such as, the lack of an EGA graphics capability) it would be advantageous to incorporate a limited version of the user interface into LOGSIM248 to facilitate circuit configuration. This interface could be designed in a similar fashion to that of LOGSIM, version 5.5, with modifications implemented to produce correctly formatted output files. A simple interface would relieve the user of the burden of insuring input file

integrity while allowing simulator familiarization in a more timely manner than is presently allowed by LOGSIM, version 5.5.

6.6 Conclusions

The initial objectives of the thesis effort were two-fold; (1) to re-engineer and re-host LOGSIM, version 5.5, for the inclusion in an integrated digital design tool, and (2) to explore concurrent simulation implementations which might further enhance performance. Unexpectedly, the efforts of the first part impacted those of the second.

The re-engineering of LOGSIM, version 5.5, was undertaken with the expressed desire to improve the portability, performance, and maintainability of the simulator. As has been previously discussed, these objectives have been successfully accomplished. The performance of LOGSIM248 is markedly improved, the readability and maintainability of the source code enhanced by a complete redesign, and the ability to port the simulator to other hosts documented. The integrated digital design tool, IDIET is built, functional, and already being demonstrated on many college campuses. Several obstacles were overcome through effective interaction between the designers of the three components of this tool with all components integrated well.

These re-engineering efforts, however, exposed the difficulties to be encountered with the exploration into concurrent implementations. It was discovered that not all sequential programs, regardless of their seemingly inherent parallelism can be easily implemented as such. Even though performance was not enhanced by concurrent implementations, many lessons were learned and future directions brought to light. As the capabilities of the integrated tool increase so will the concurrent applications.

Bibliography

1. Rowe, Mark C. EE650: Special Studies LOGSIM Software Engineering. Class Report. Air Force Institute of Technology, Wright-Patterson AFB OH, June 1985.
2. Adams, Charles A. Jr. A Digital Circuit Design Environment. MS thesis, AFIT/GCS/ENG/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
3. Wagner, Steven M. An Expert System for Discrete Component Digital Circuit Design. MS thesis, AFIT/GCS/ENG/87D-28. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
4. Miczo, Alexander. Digital Logic Testing and Simulation. New York NY: Harper & Row, Publishers, Inc..
5. Micro Data Base Systems, Inc.. GURU Reference Manual, Volumes 1 and 2. Lafayette IN, October 1986.
6. Rowe, Mark C. LOGSIM, version 5.5. Computer Software. Air Force Institute of Technology, Wright-Patterson AFB OH, June 1985.
7. Borland International. TURBO Pascal Reference Manual Version 3.0. Scotts Valley CA: Borland International, 1984.
8. University of California, Berkely. Berkely Pascal. Computer Software. University of California, Berkely, Berkely CA:
9. Borland International. TURBO C Reference Guide. Scotts Valley CA: Borland International, 1987
10. Pressman, Roger S. Software Engineering: A Practitioners Approach. New York NY: McGraw-Hill, 1982.
11. Becker, Richard A. and John M. Chambers. S Language and System for Data Analysis. Murray Hill NJ: Bell Labs, 1981.
12. Rowe, Mark C. The LOGSIM Simulator Revised User's Guide. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1985.

13. Cody, Ronald P. and Jeffrey K. Smith. Applied Statistics and the SAS Programming Language. New York NY: North-Holland, 1985

Appendix A: Source Code - LOGSIM248

Source code available upon request.
Write to:

Air Force Institute of Technology
School of Engineering
Department of Engineering and Computer Science
Wright Patterson Air Force Base, Ohio 45433

Appendix B: Data Flow Diagrams and Functional Decomposition

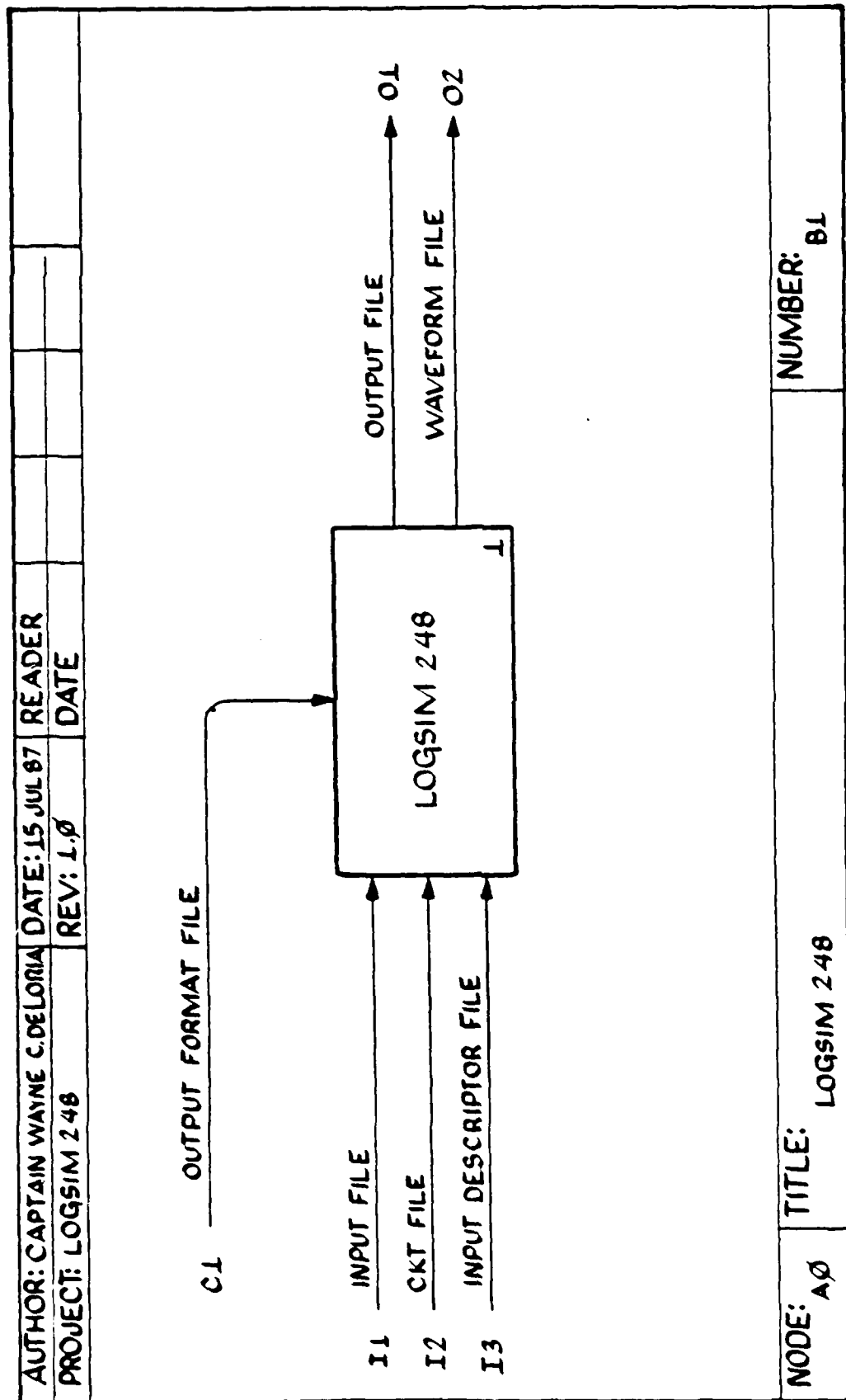


Figure B1. LOGSIM248 A0

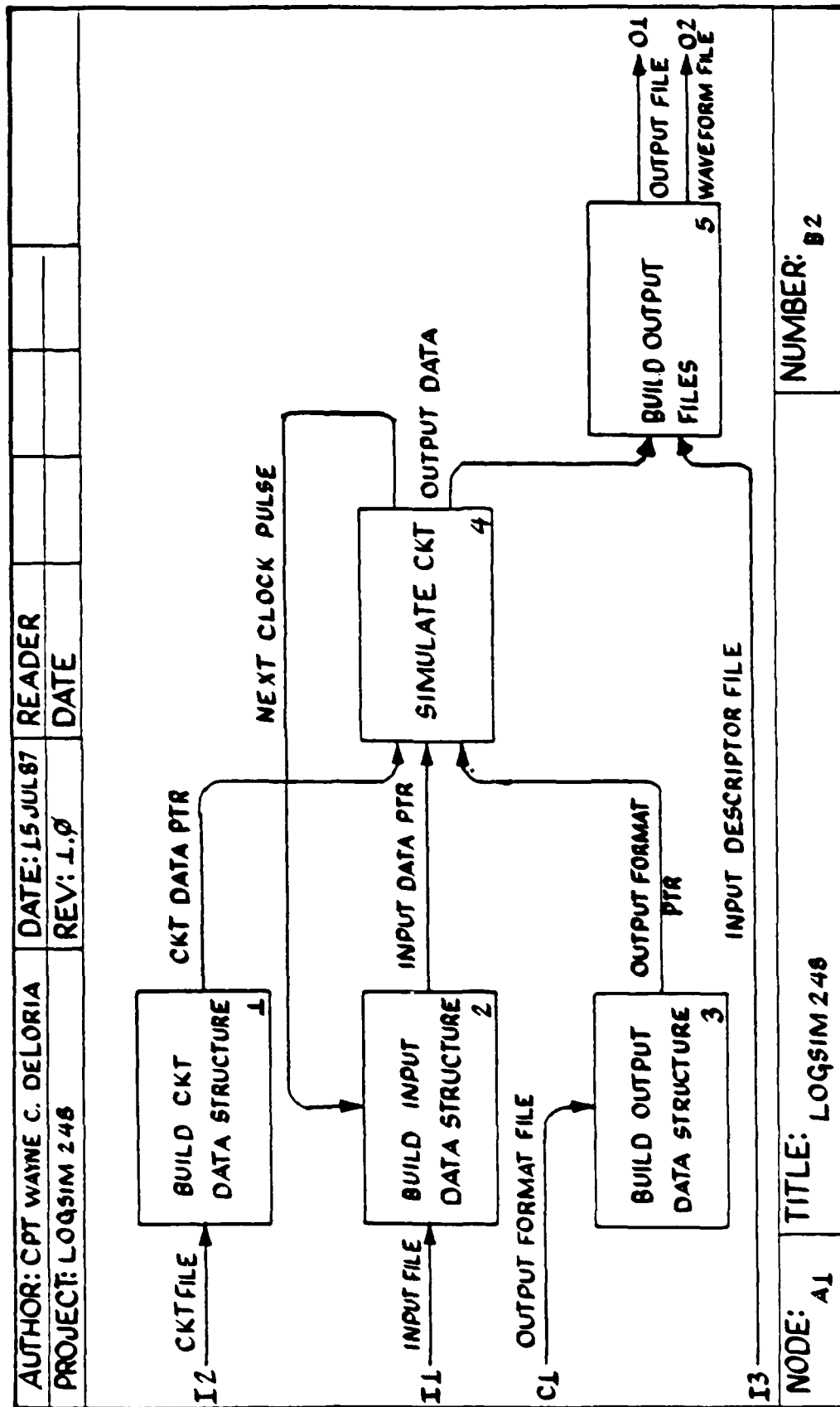


Figure B2. LOGSIM248 A1

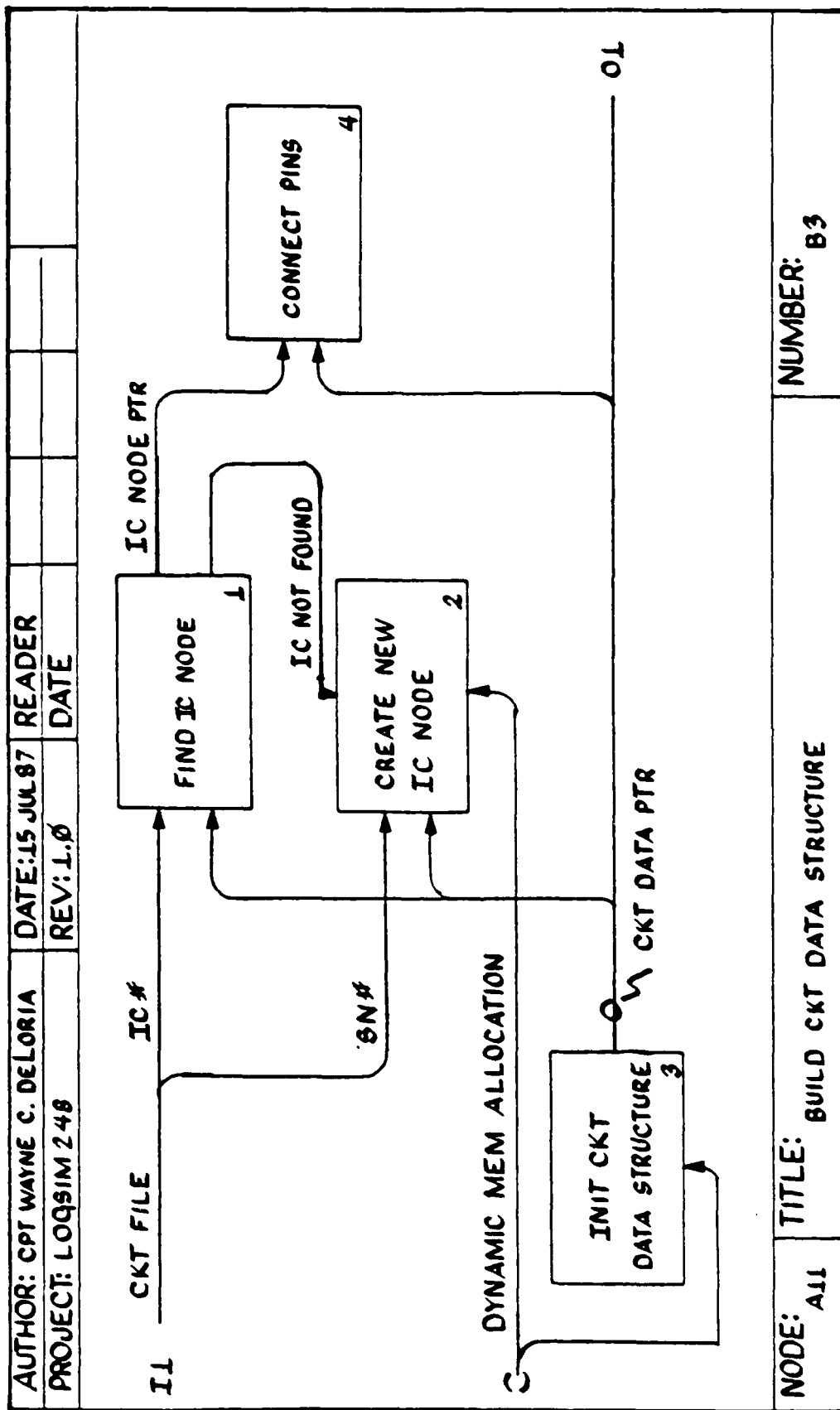


Figure B3. Build CKT Data Structure

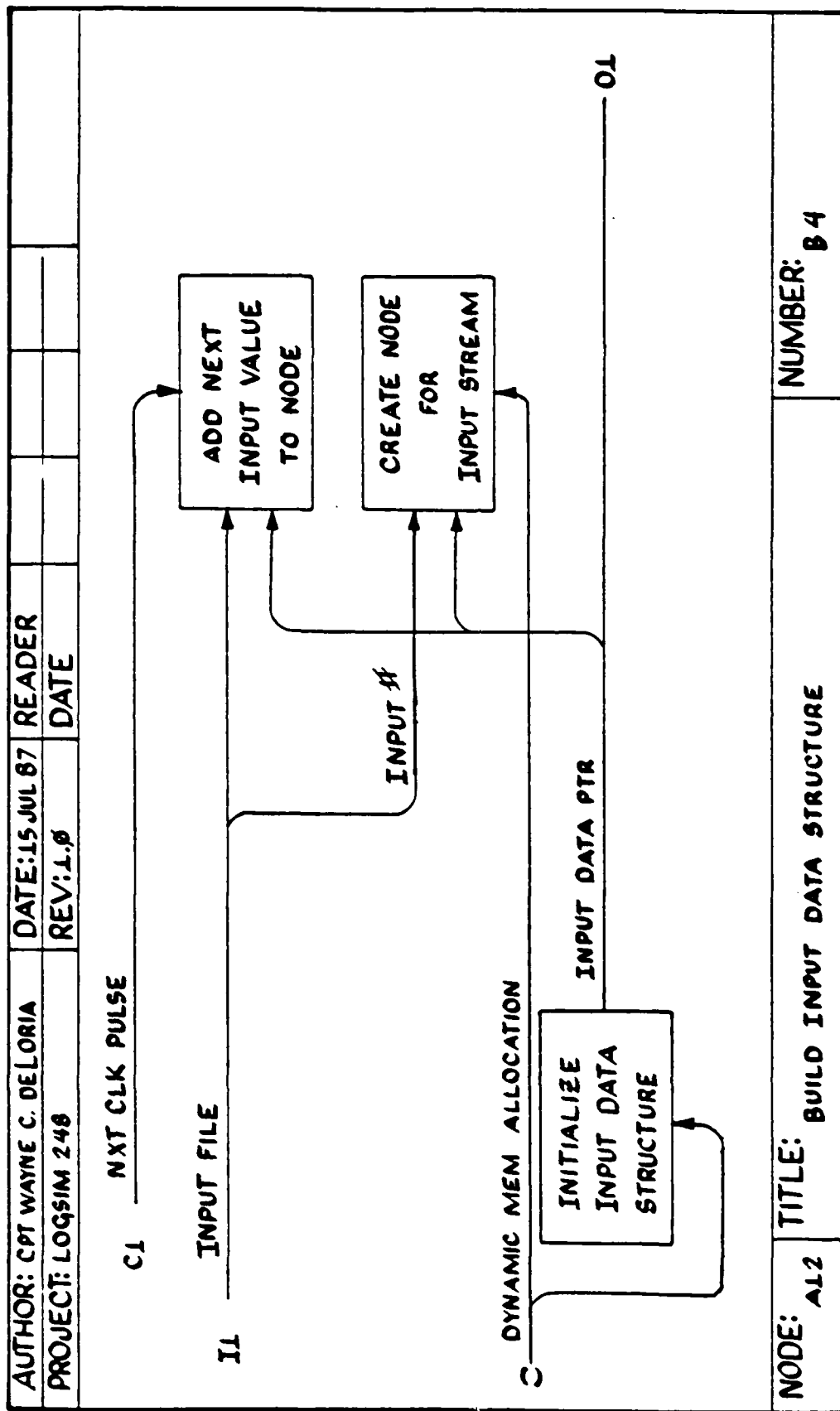


Figure B4. Build Input Data Structure

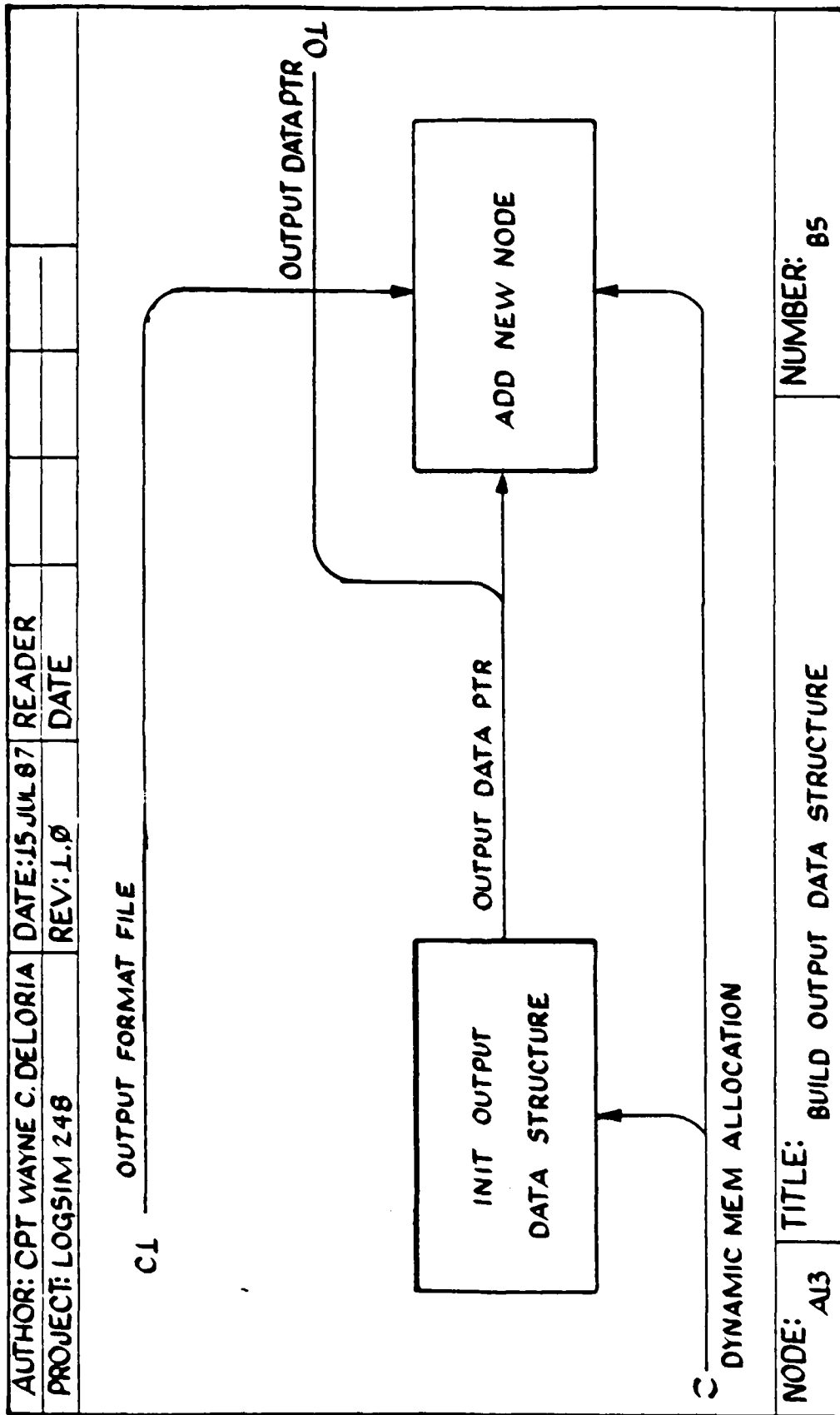


Figure B5. Build Output Data Structure

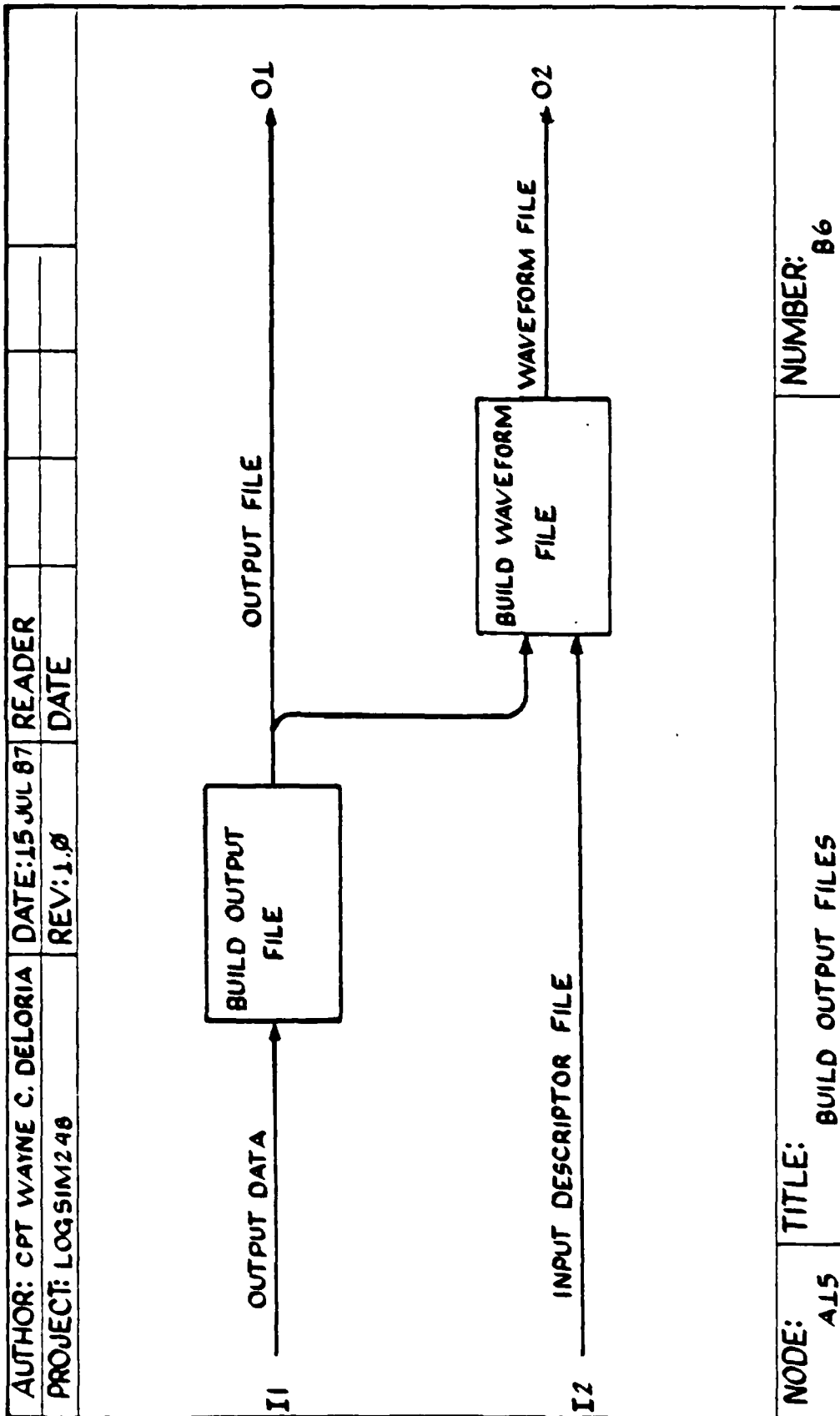


Figure B6. Build Output Files

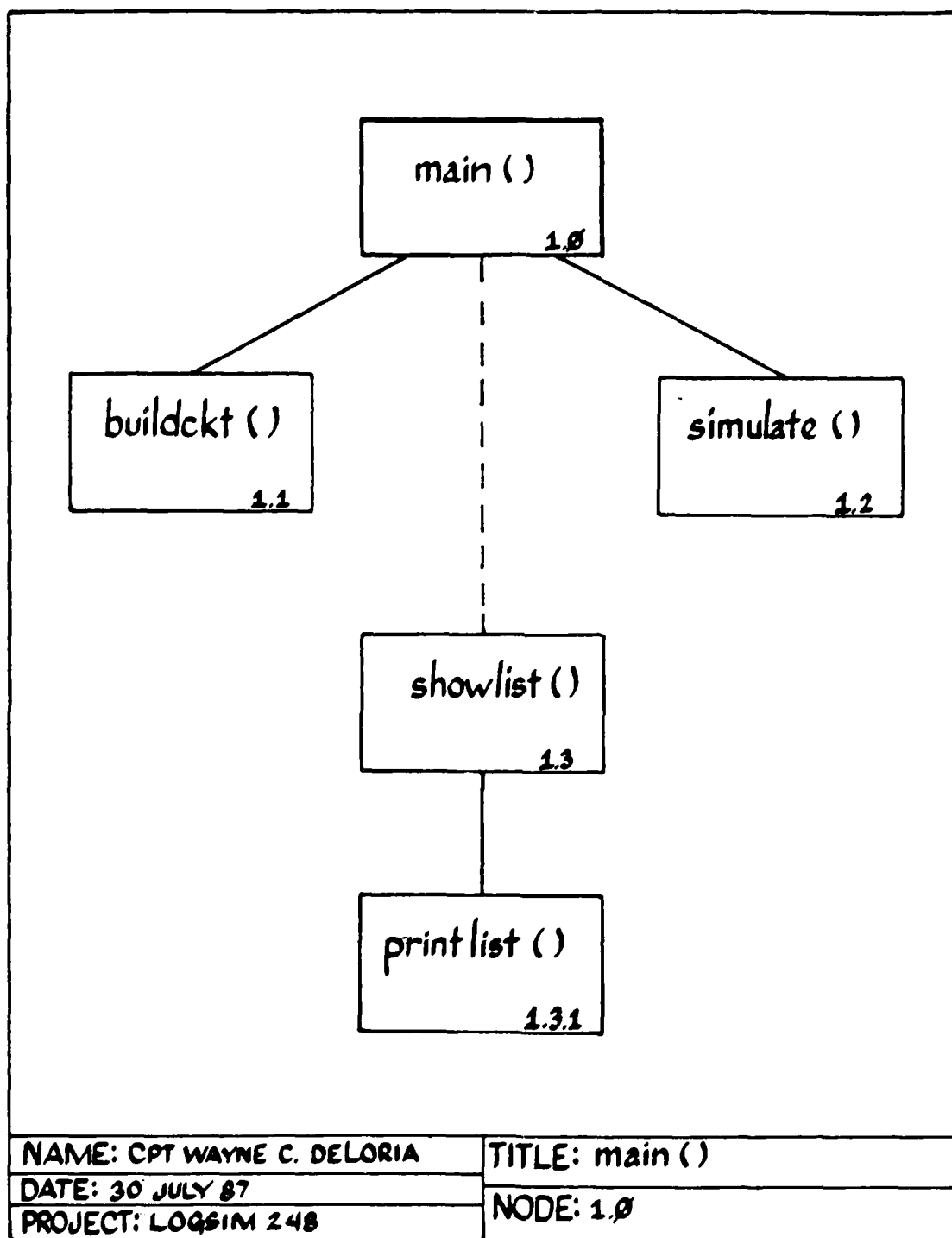


Figure B7. `main()`

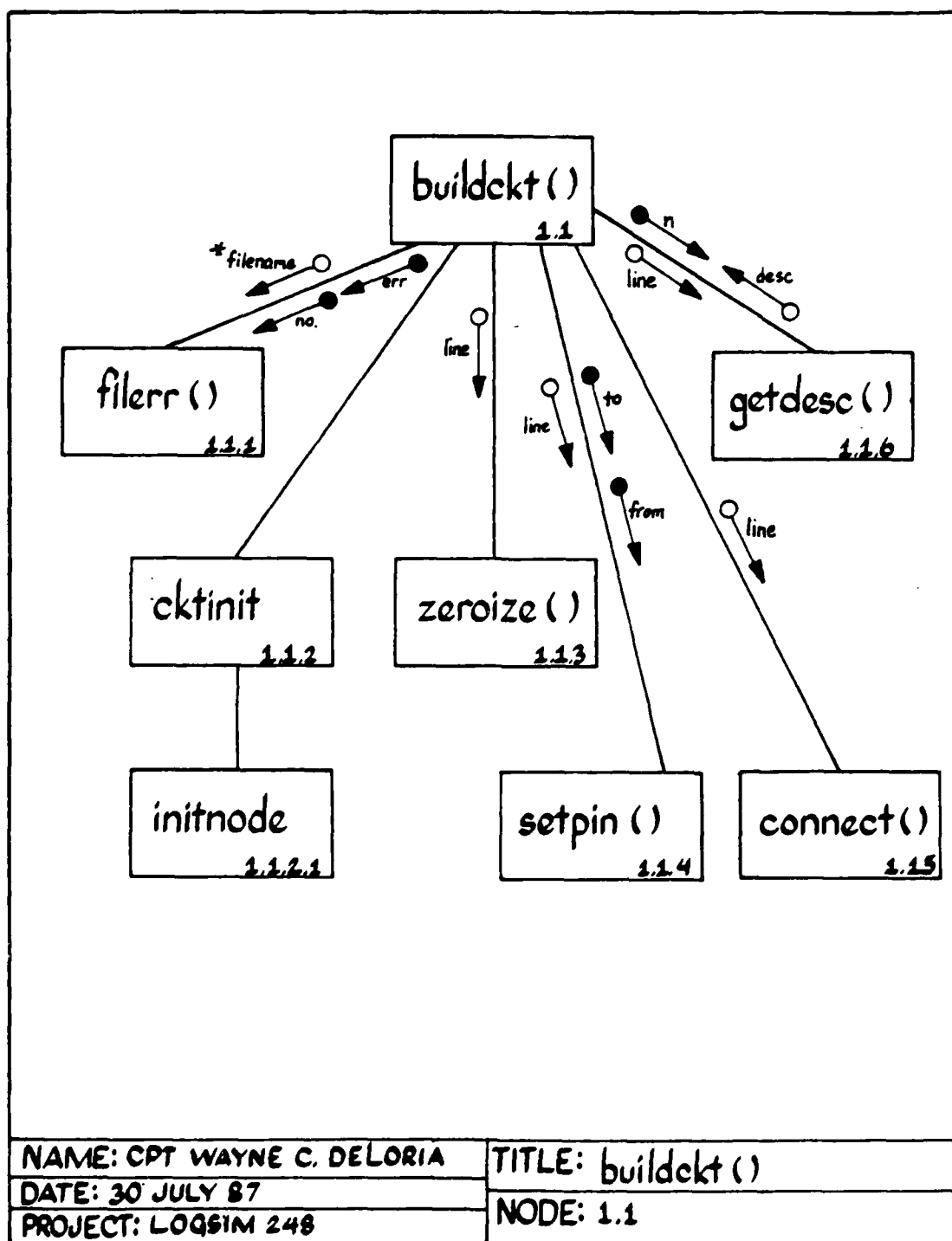


Figure B8. buildckt()

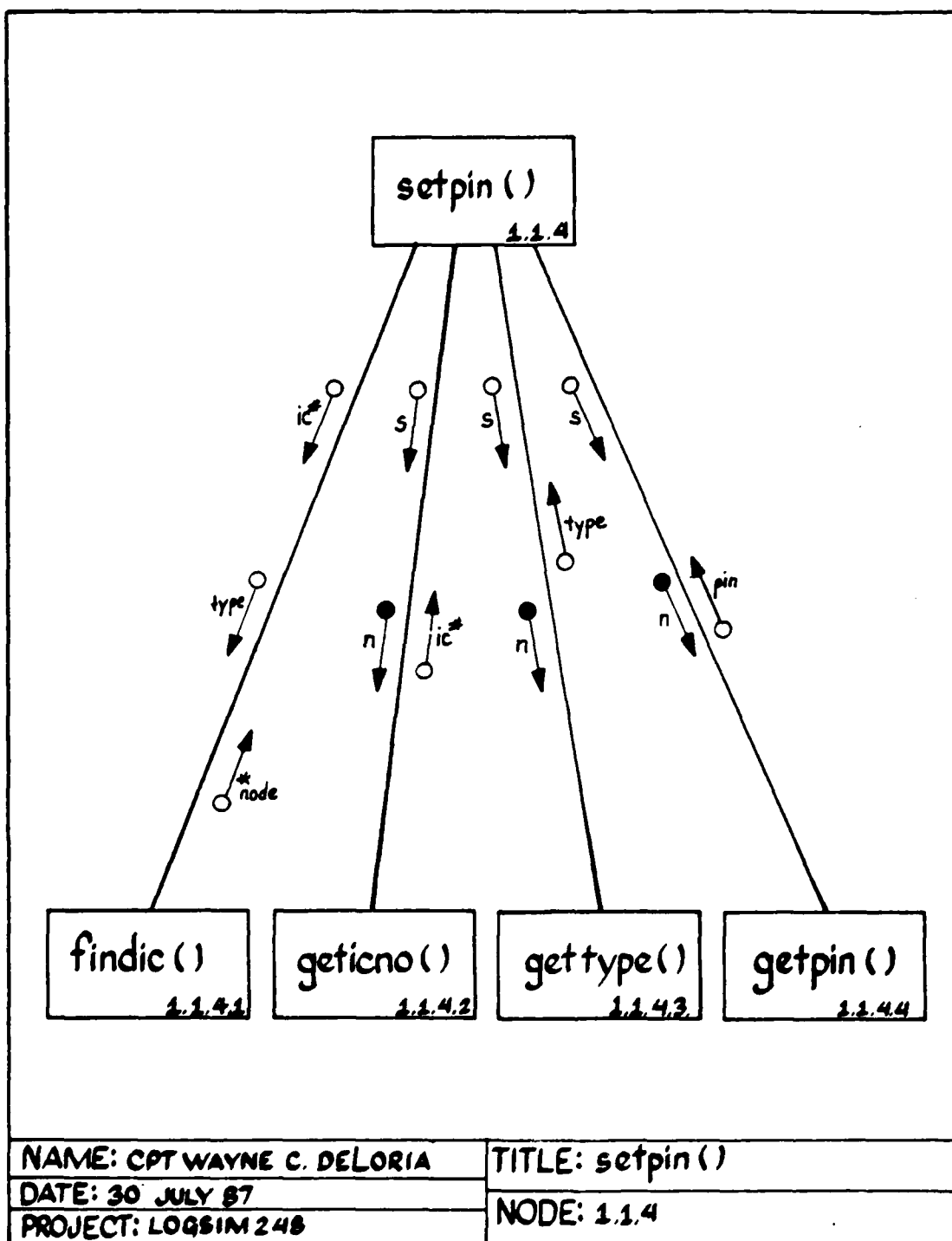


Figure B9. `Setpin()`

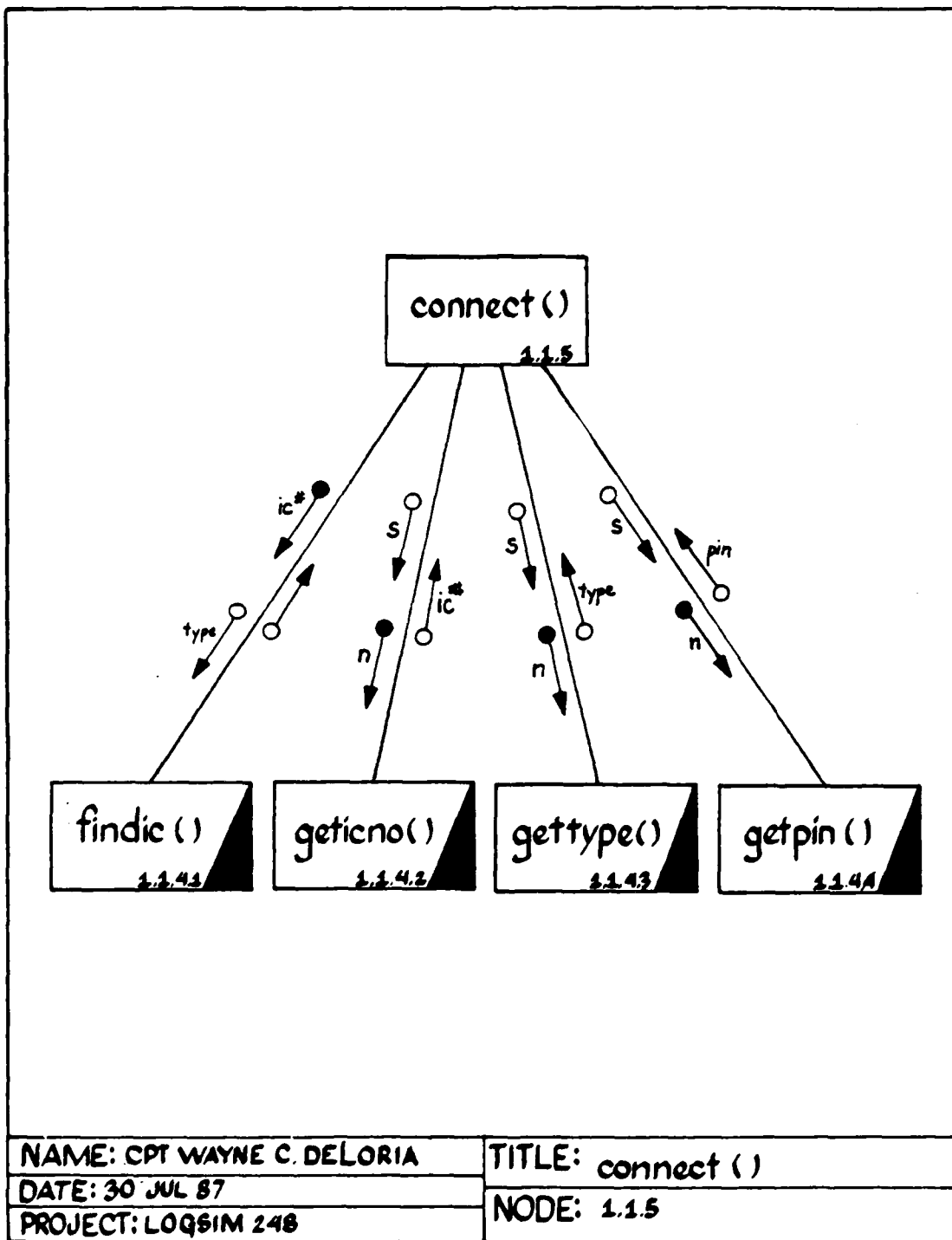


Figure B10. connect()

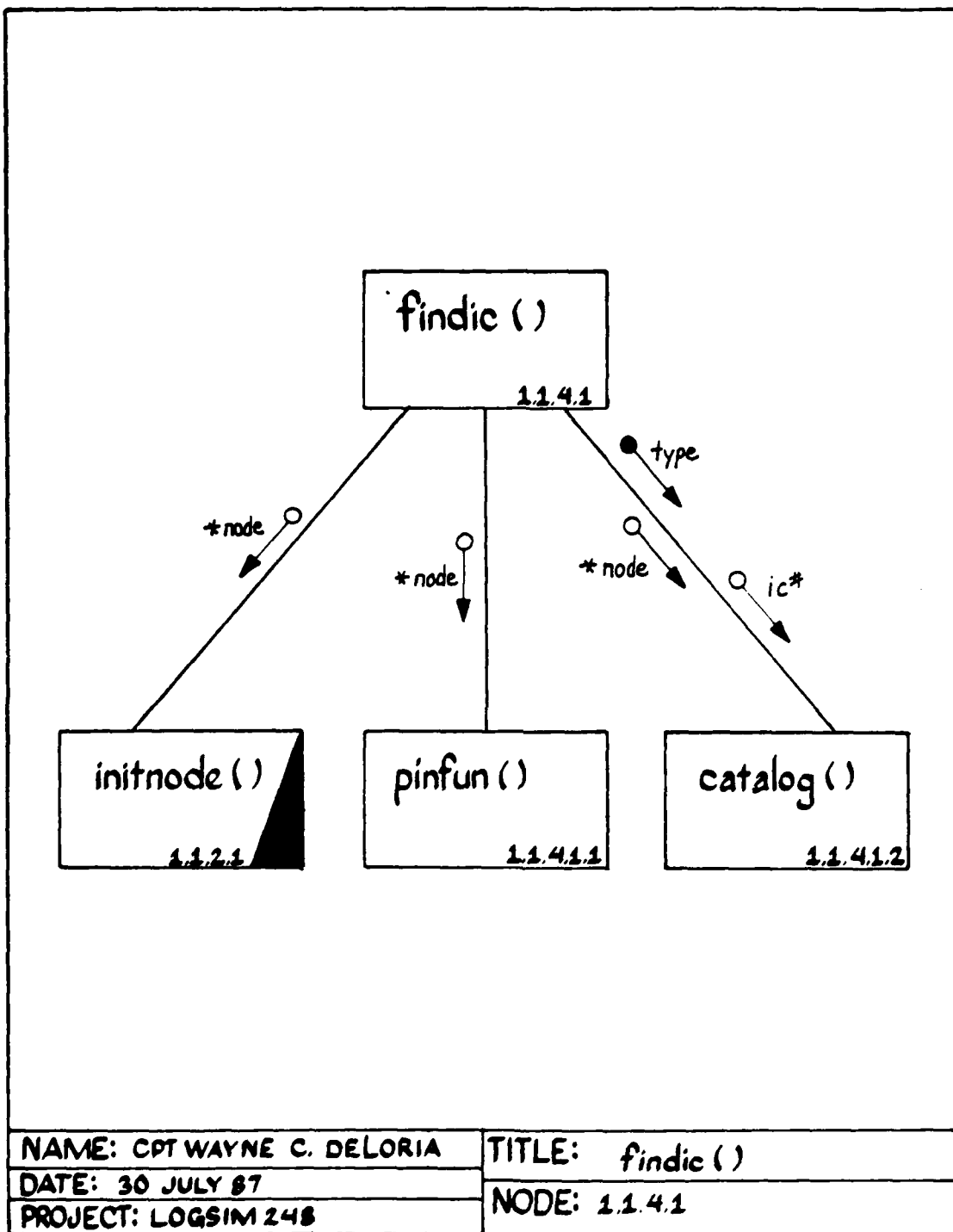


Figure B11. *findic()*

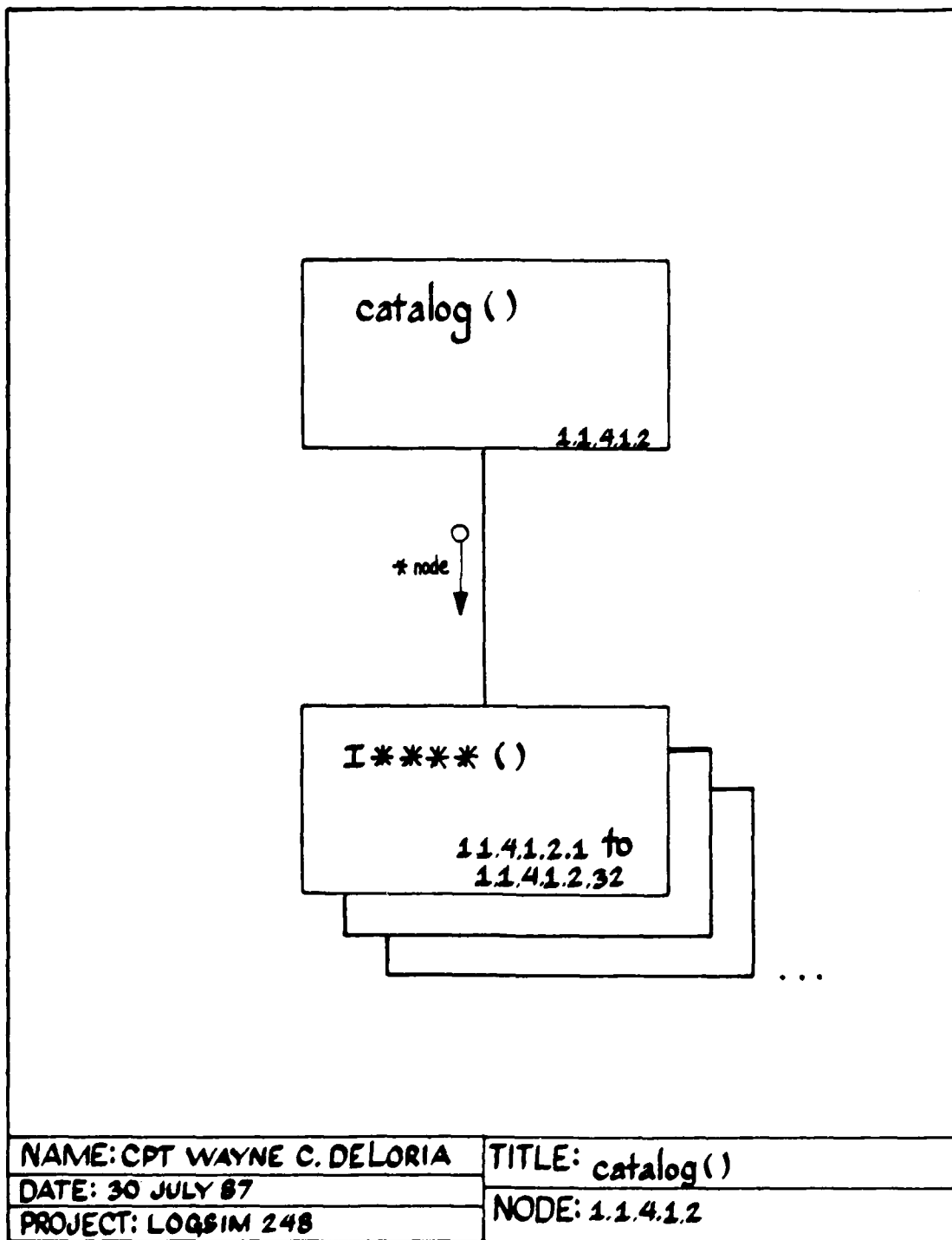


Figure B12. `catalog()`

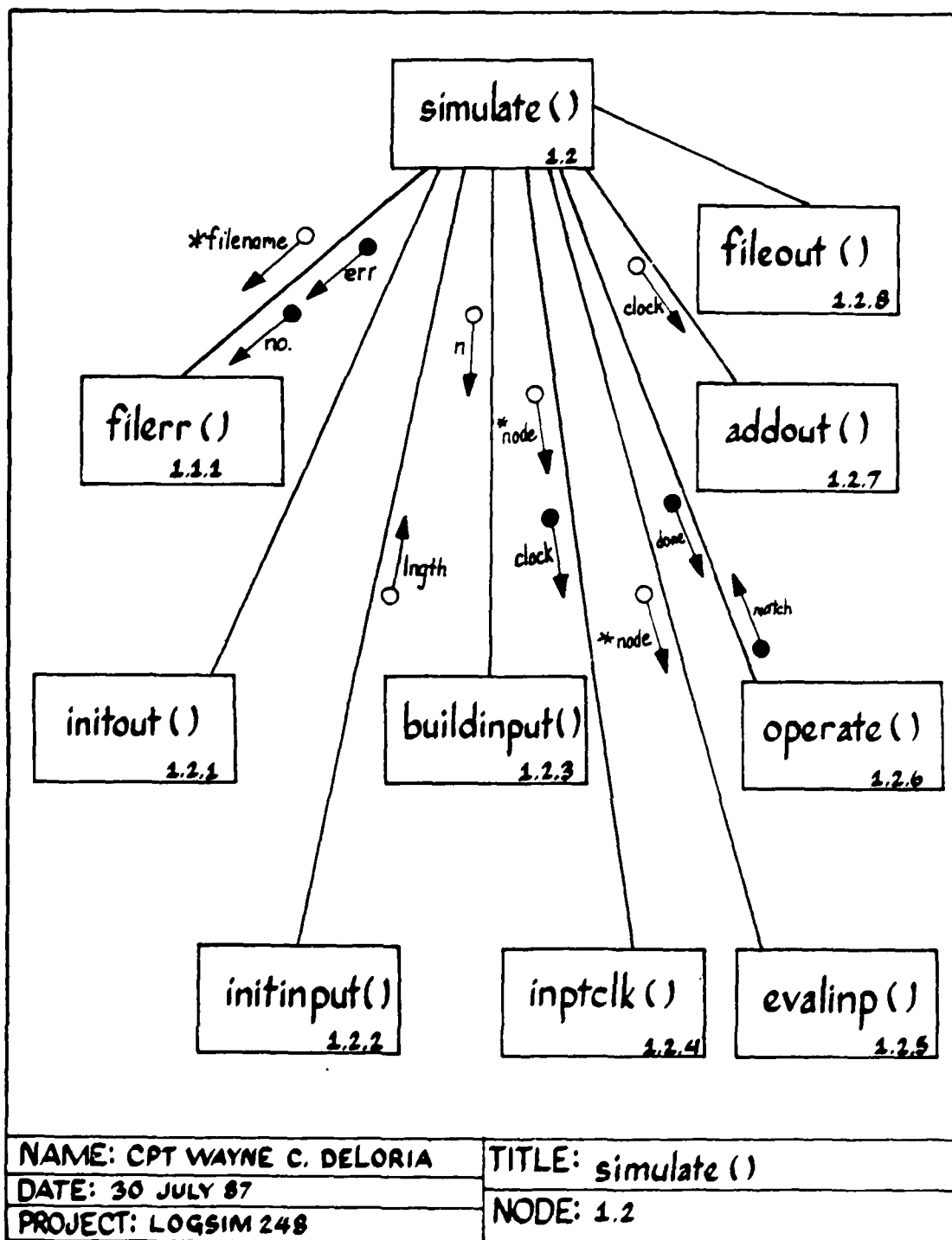


Figure B13. simulate()

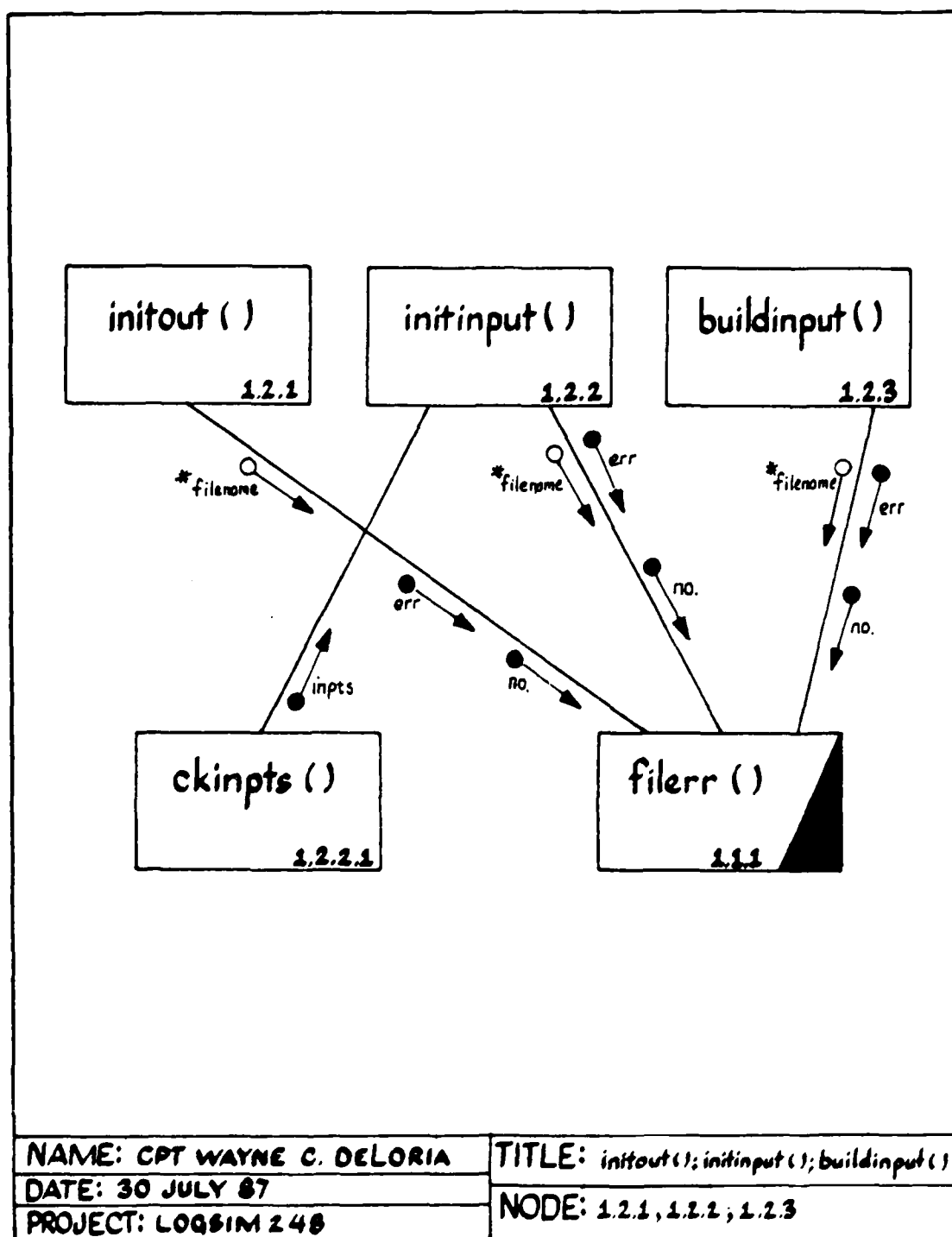


Figure B14. *initout()*, *initinput()*, *buildinput()*

NO-A188 823

A DIGITAL LOGIC SIMULATOR WITH CONCURRENT PROGRAMMING
CONSIDERATIONS(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING

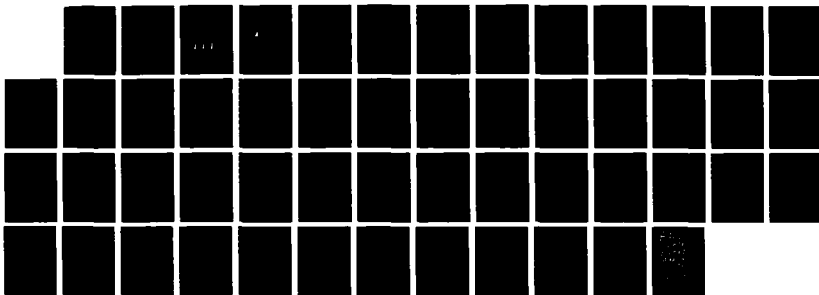
2/2

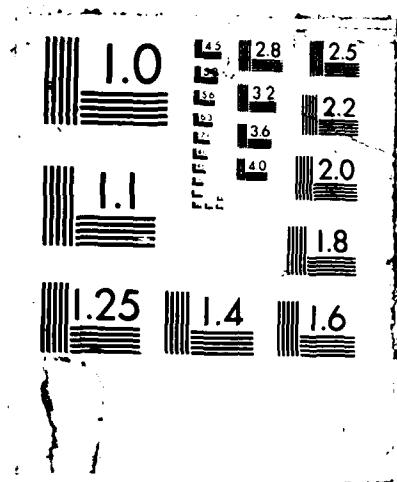
UNCLASSIFIED

W C DELORIA DEC 87 AFIT/GCS/ENG/87D-10

F/G 12/1

NL







B-15

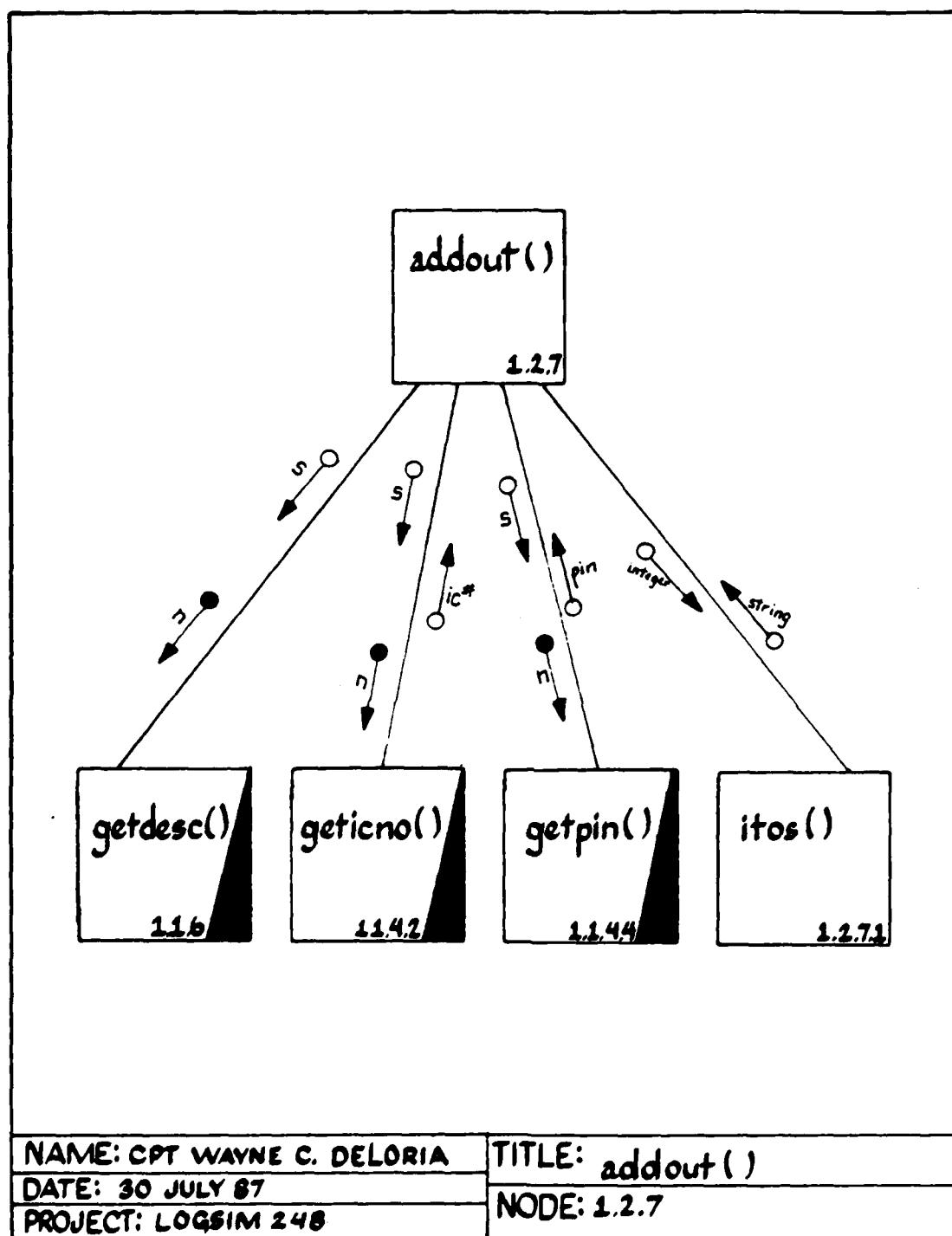


Figure B16. `addout()`

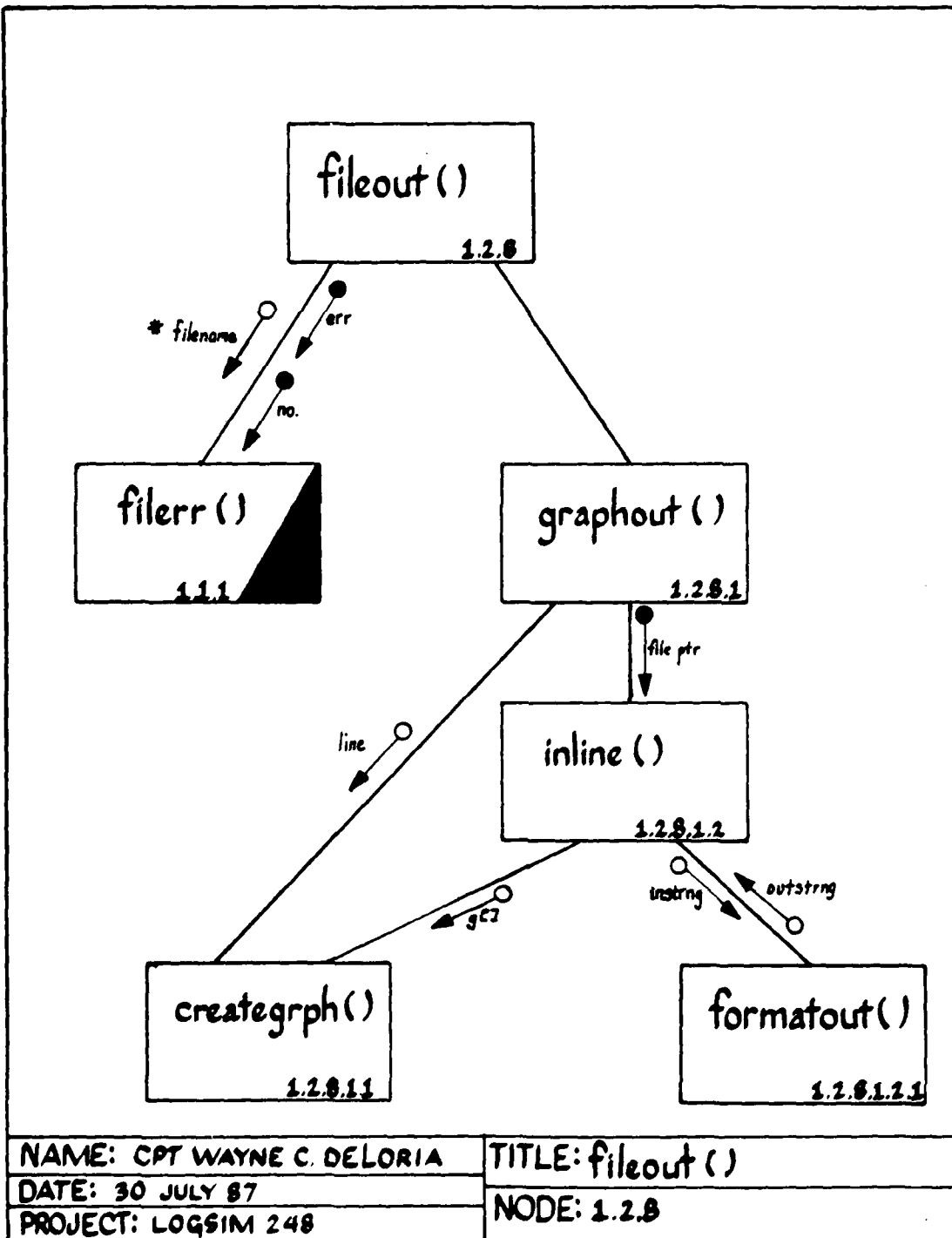


Figure B17. fileout()

Appendix C: Statistical Information

LOGSIM248 vs. LOGSIM V5.5 -- Sample Requirements for BCD.CKT

NUMBER RUNS NEEDED FOR SPEED COMPARISON

0.95
confidence level

0.1
error tolerance within +/- seconds

```
Read 50 items
Read 50 items
```

LOGSIM248 Data

```
4.174399      0.02459135
average      standard deviation
```

0.1700558
NUMBER RUNS NEEDED

LOGSIM V5.5 Data

6.227600	0.3524987
average	standard deviation

34.9415
NUMBER RUNS NEEDED

Paired T-test for BCD.CKT

```
1 LOGSIM RUN TIME DATA 1
14:55 SUNDAY, OCTOBER 25, 1987
```

VARIABLE	N	MEAN	STD ERROR OF MEAN
LOGSIM V5.5	50	6.21140000	0.04485533
LOGSIM248	50	4.17480000	0.00341951

[illegible]

VARIABLE	N	MEAN	STD ERROR OF MEAN	T	PR> T
DIFF	50	2.036600000	0.04408490	46.20	0.0001

Run Times for BCD.CKT

LOGSIM V5.5 Leading Edge	LOGSIM248 Leading Edge	LOGSIM248 Zenith 248	LOGSIM248 Intel iPSC
6.71	4.210	1.370	2.160
6.13	4.210	1.430	2.160
6.07	4.210	1.380	2.160
6.61	4.180	1.430	2.200
6.17	4.180	1.490	2.220
6.55	4.210	1.430	2.180
6.40	4.190	1.430	2.180
6.02	4.200	1.430	2.180
6.11	4.200	1.430	2.160
5.98	4.190	1.430	2.160
5.86	4.180	1.420	2.180
5.85	4.190	1.420	2.180
5.92	4.190	1.430	2.180
6.66	4.200	1.430	2.180
6.86	4.220	1.430	2.180
5.85	4.210	1.430	2.160
6.46	4.190	1.430	2.160
6.66	4.210	1.480	2.160
6.61	4.210	1.430	2.160
6.23	4.210	1.480	2.160
5.96	4.150	1.420	2.200
6.46	4.170	1.480	2.180
6.82	4.160	1.480	2.220
5.62	4.150	1.480	2.160
6.40	4.130	1.480	2.160
6.03	4.140	1.490	2.180
6.30	4.170	1.480	2.160
6.19	4.160	1.490	2.180
6.15	4.140	1.480	2.160
6.05	4.170	1.480	2.220
5.71	4.190	1.420	2.160
6.11	4.170	1.480	2.180
6.12	4.150	1.430	2.180
6.13	4.140	1.540	2.180
5.74	4.140	1.490	2.160
6.34	4.170	1.480	2.160
6.83	4.160	1.490	2.200
6.22	4.150	1.490	2.160
6.29	4.170	1.480	2.180
6.17	4.150	1.480	2.180
5.96	4.160	1.420	2.180
5.61	4.170	1.480	2.220
5.99	4.160	1.530	2.180
6.39	4.150	1.490	3.000
6.52	4.180	1.480	2.180
6.09	4.170	1.540	2.160
5.87	4.150	1.490	2.160

Run Times for BCD.CKT (cont.)

6.24	4.170	1.480	2.160
6.02	4.150	1.430	2.260
6.53	4.140	1.480	2.180
-----	-----	-----	-----
mean 6.2114	4.1744	1.4598	2.1940

LOGSIM248 vs. LOGSIM V5.5 -- Sample Requirements for
ADDER.CKT

NUMBER RUNS NEEDED FOR SPEED COMPARISON

0.95
confidence level

0.1
error tolerance within +/- seconds

```
Read 70 items
Read 70 items
```

LOGSIM248 Data

6.710714	0.03600813
average	standard deviation

0.3605694
NUMBER RUNS NEEDED

LOGSIM V5.5 Data

15.38814	0.495871
average	standard deviation

68.3793
NUMBER RUNS NEEDED

Paired T-test for ADDER.CKT

```
1 LOGSIM RUN TIME DATA 1
14:58 SUNDAY, OCTOBER 25, 1987
```

VARIABLE	N	MEAN	STD ERROR OF MEAN
LOGSIM V5.5	70	15.38814286	0.05926789
LOGSIM248	70	6.71071429	0.00430379

```

1          LOGSIM RUN TIME DATA          2
          14:58 SUNDAY, OCTOBER 25, 1987

```

VARIABLE	N	MEAN	STD ERROR OF MEAN	T	PR> T
DIFF	70	8.67742857	0.05721547	151.66	0.0001

Run Times for ADDER.CKT

LOGSIM V5.5 Leading Edge	LOGSIM248 Leading Edge	LOGSIM248 Zenith 248	LOGSIM248 Intel iPSC
16.43	6.750	2.090	4.160
16.28	6.790	2.200	4.140
16.07	6.700	2.030	4.120
15.50	6.780	2.090	4.160
15.88	6.740	2.090	4.140
16.72	6.740	2.030	4.160
16.42	6.720	2.090	4.140
15.45	6.700	2.030	4.160
15.51	6.720	2.090	4.120
15.63	6.760	2.140	4.140
15.47	6.780	2.090	4.140
16.31	6.710	2.080	4.100
15.37	6.750	2.030	4.140
15.91	6.730	2.090	4.120
15.54	6.740	2.140	4.120
15.13	6.750	2.150	4.120
15.35	6.730	2.090	4.120
16.05	6.770	2.030	4.120
15.22	6.730	2.090	4.120
16.00	6.730	2.090	4.120
15.25	6.780	2.080	4.120
15.48	6.740	2.090	4.140
15.68	6.730	2.090	4.160
15.52	6.760	2.030	4.120
16.45	6.700	2.030	4.120
15.89	6.750	2.090	4.140
15.60	6.750	2.030	4.100
15.92	6.720	2.090	4.120
15.06	6.720	2.090	4.120
15.90	6.700	2.030	4.140
15.59	6.710	2.150	4.120
15.81	6.720	2.080	4.120
14.92	6.720	2.090	4.160
15.81	6.730	2.090	4.120
15.15	6.730	2.140	4.180
15.22	6.740	2.090	4.140
15.09	6.730	2.080	4.140
15.26	6.760	2.140	4.120
15.19	6.710	2.090	4.140
15.22	6.770	2.090	4.140
14.93	6.680	2.030	4.140
15.17	6.670	2.090	4.120
15.94	6.680	2.090	4.140
14.96	6.680	2.090	4.140
14.96	6.690	2.090	4.140
15.45	6.670	2.030	4.120
15.01	6.680	2.090	4.140

Run Times for ADDER.CKT (cont.)

14.83	6.670	2.090	4.220
15.06	6.670	2.080	4.180
14.96	6.670	2.090	4.340
14.95	6.670	2.090	4.160
14.99	6.670	2.090	4.140
14.94	6.680	2.090	4.180
14.76	6.680	2.080	4.160
14.98	6.670	2.080	4.120
14.76	6.680	2.090	4.120
14.69	6.680	2.090	4.120
14.87	6.670	2.090	4.120
14.82	6.670	2.090	4.140
15.01	6.670	2.090	4.180
15.89	6.680	2.090	4.180
14.97	6.680	2.090	4.140
14.99	6.670	2.080	4.160
15.19	6.680	2.090	4.140
15.29	6.680	2.140	4.160
14.88	6.680	2.140	4.140
14.86	6.670	2.140	4.160
15.21	6.670	2.140	5.120
14.74	6.670	2.150	4.120
14.86	6.680	2.140	4.140
-----	-----	-----	-----
mean 15.3881	6.7107	2.0900	4.1557

LOGSIM248 vs. LOGSIM V5.5 -- Sample Requirements for
BRC3S.CKT

NUMBER RUNS NEEDED FOR SPEED COMPARISON

0.95
confidence level

0.1
error tolerance within +/- seconds

```
Read 65 items
Read 69 items
```

LOGSIM V5.5 Data

```
3.165692      0.02839286
average      standard deviation
```

0.2246619
NUMBER RUNS NEEDED

LOGSIM248 Data

6.380869	0.4872877
average	standard deviation

66.059
NUMBER RUNS NEEDED

Paired T-test for BRC3S.CKT

```
1 LOGSIM RUN TIME DATA 1
15:01 SUNDAY, OCTOBER 25, 1987
```

VARIABLE	N	MEAN	STD ERROR OF MEAN
LOGSIM V5.5	69	6.38086957	0.05866254
LOGSIM248	69	3.16463768	0.00338984

```
LOGSIM RUN TIME DATA
```

1		2
	15:01 SUNDAY, OCTOBER 25, 1987	

VARIABLE	N	MEAN	STD ERROR OF MEAN	T	PR> T
DIFF	69	3.21623188	0.05740001	56.03	0.0001

Run Times for BRC3S.CKT

LOGSIM V5.5 Leading Edge	LOGSIM248 Leading Edge	LOGSIM248 Zenith 248	LOGSIM248 Intel iPSC
6.98	3.150	1.810	1.900
6.45	3.150	1.750	1.920
5.89	3.170	1.760	1.920
6.24	3.190	1.810	1.920
6.42	3.170	1.820	1.920
6.97	3.180	1.810	1.940
6.86	3.190	1.810	1.920
7.02	3.180	1.810	1.940
6.93	3.180	1.760	1.920
6.85	3.180	1.820	1.940
6.42	3.210	1.810	1.940
7.06	3.190	1.820	1.920
6.35	3.160	1.810	1.900
7.88	3.190	1.810	1.920
6.42	3.170	1.750	1.920
5.79	3.200	1.810	1.940
5.59	3.190	1.820	2.160
6.81	3.200	1.810	1.920
7.00	3.200	1.810	1.940
6.36	3.190	1.750	1.920
6.37	3.200	1.760	1.940
6.86	3.200	1.820	1.920
6.62	3.190	1.820	1.920
6.32	3.140	1.810	1.900
6.46	3.200	1.860	1.920
7.28	3.190	1.810	1.920
6.65	3.200	1.810	1.920
6.69	3.210	1.760	1.920
6.34	3.200	1.810	1.920
6.98	3.190	1.760	1.920
7.16	3.180	1.820	1.940
6.22	3.190	1.810	2.000
6.56	3.190	1.810	1.920
6.84	3.200	1.810	1.900
6.38	3.220	1.870	1.920
6.06	3.120	1.820	1.920
5.83	3.120	1.750	1.920
6.24	3.140	1.870	1.920
6.32	3.130	1.920	1.920
5.95	3.130	1.870	1.940
7.36	3.140	1.810	1.920
6.18	3.120	1.760	1.980
6.51	3.140	1.810	1.940
6.31	3.140	1.810	1.900
6.34	3.120	1.810	1.920
6.02	3.120	1.760	1.920
6.18	3.140	1.810	2.000

Run Times for BRC3S.CKT (cont.)

6.20	3.140	1.810	1.920
6.63	3.140	1.810	1.920
6.35	3.160	1.810	1.900
6.62	3.120	1.820	1.940
5.96	3.150	1.870	1.920
6.90	3.140	1.870	1.920
6.01	3.150	1.870	1.900
6.27	3.190	1.810	1.920
5.58	3.160	1.810	1.920
5.78	3.140	1.810	1.940
5.61	3.150	1.760	1.920
6.08	3.140	1.760	1.920
5.84	3.140	1.810	1.920
5.76	3.160	1.750	1.920
6.27	3.140	1.750	1.940
5.91	3.160	1.760	1.920
6.47	3.150	1.870	1.920
5.82	3.140	1.810	1.920
5.89	3.160	1.810	1.920
5.58	3.120	1.810	1.920
5.71	3.150	1.810	1.900
5.72	3.160	1.810	1.920

mean 6.3809	3.1646	1.8072	1.9281

LOGSIM248 vs. LOGSIM V5.5 -- Sample Requirements for
DECODER.CKT

NUMBER RUNS NEEDED FOR SPEED COMPARISON

0.95
confidence level

0.1
error tolerance within +/- seconds

```
Read 112 items
Read 112 items
```

LOGSIM248 Data

8.791125	0.02086686
average	standard deviation

0.1206719
NUMBER RUNS NEEDED

LOGSIM V5.5 Data

21.60821	0.566321
average	standard deviation

88.2772
NUMBER RUNS NEEDED

Paired T-test for DECODER.CKT

```
1 LOGSIM RUN TIME DATA 1
18:46 SUNDAY, OCTOBER 25, 1987
```

VARIABLE	N	MEAN	STD ERROR OF MEAN
LOGSIM V5.5	112	21.60821429	0.05351229
LOGSIM248	112	8.78107143	0.00230535

```

1          LOGSIM RUN TIME DATA          2
          18:46 SUNDAY, OCTOBER 25, 1987

```

VARIABLE	N	MEAN	STD ERROR OF MEAN	T	PR> T
DIFF	112	12.82714286	0.05383677	238.26	0.0001

Run Times for DECODER.CKT

LOGSIM V5.5 Leading Edge	LOGSIM248 Leading Edge	LOGSIM248 Zenith 248	LOGSIM248 Intel iPSC
22.17	8.800	2.800	7.600
21.67	8.750	2.800	7.600
22.52	8.770	2.810	7.600
21.79	8.750	2.860	7.580
21.65	8.750	2.800	7.560
21.31	8.750	2.800	7.560
21.52	8.750	2.800	7.640
21.77	8.770	2.800	7.600
21.54	8.770	2.800	7.600
21.65	8.750	2.800	7.560
21.45	8.770	2.810	7.560
21.97	8.770	2.800	7.640
21.52	8.750	2.800	7.620
21.51	8.750	2.750	7.580
21.52	8.750	2.800	7.560
22.10	8.780	2.800	7.560
21.44	8.750	2.850	7.580
21.26	8.750	2.910	7.580
21.02	8.750	2.860	7.560
22.91	8.750	2.850	7.560
21.71	8.750	2.860	7.560
22.13	8.750	2.920	7.640
22.39	8.770	2.860	7.560
21.45	8.770	2.860	7.560
21.62	8.750	2.910	7.600
21.37	8.760	2.800	7.580
21.67	8.750	2.860	7.600
22.91	8.750	2.800	7.560
22.79	8.760	2.860	7.580
21.44	8.760	2.860	7.560
21.91	8.750	2.850	7.680
21.73	8.750	2.850	7.580
22.68	8.750	2.860	7.560
21.48	8.750	2.910	7.600
22.51	8.800	2.860	7.600
22.71	8.780	2.910	7.600
21.18	8.790	2.850	7.580
21.54	8.780	2.850	7.560
21.41	8.800	2.860	7.560
21.41	8.790	2.860	7.560
22.79	8.770	2.800	7.580
21.88	8.790	2.910	7.580
22.42	8.830	2.860	7.580
21.75	8.780	2.920	7.560
21.34	8.780	2.910	7.600
22.34	8.810	2.910	7.580
21.21	8.780	2.800	7.560

Run Times for DECODER.CKT (cont.)

21.23	8.770	2.850	7.560
22.91	8.770	2.800	7.580
22.79	8.830	2.860	7.560
21.44	8.810	2.850	7.600
21.91	8.800	2.800	7.580
21.73	8.820	2.850	7.560
22.68	8.780	2.850	7.780
21.48	8.830	2.850	7.560
22.51	8.770	2.860	7.580
22.71	8.810	2.910	7.580
21.18	8.780	2.800	7.580
21.54	8.800	2.800	7.620
21.41	8.780	2.850	7.560
21.41	8.790	2.810	7.560
22.79	8.770	2.800	7.600
21.88	8.770	2.740	7.600
22.42	8.770	2.860	7.560
21.75	8.770	2.800	7.600
21.34	8.760	2.800	7.580
22.34	8.780	2.800	7.560
21.21	8.770	2.800	7.600
21.23	8.830	2.810	7.640
21.35	8.820	2.800	7.600
21.79	8.780	2.850	7.600
21.26	8.780	2.800	7.600
21.48	8.810	2.800	7.580
21.01	8.830	2.800	7.640
20.91	8.780	2.800	7.560
20.96	8.830	2.800	7.560
20.89	8.830	2.850	7.560
22.57	8.810	2.740	7.620
21.14	8.800	2.800	7.580
21.20	8.780	2.800	7.560
21.29	8.760	2.850	7.580
21.12	8.800	2.810	7.580
21.01	8.780	2.800	7.640
21.00	8.780	2.740	7.580
21.21	8.810	2.800	9.100
21.43	8.780	2.800	7.680
21.06	8.780	2.810	7.640
21.50	8.830	2.800	7.600
22.21	8.820	2.800	7.580
21.15	8.790	2.860	7.580
21.01	8.790	2.850	7.580
20.91	8.780	2.850	7.680
21.00	8.760	2.860	7.560
21.25	8.760	2.860	7.640
21.06	8.790	2.800	7.600
20.98	8.780	2.800	7.580
21.45	8.780	2.800	7.580

Run Times for DECODER.CKT (cont.)

21.16	8.810	2.800	7.600
20.99	8.780	2.800	7.560
21.43	8.810	2.800	7.560
21.18	8.830	2.800	7.580
21.24	8.780	2.800	7.580
21.26	8.760	2.800	7.580
21.16	8.780	2.800	7.600
20.96	8.800	2.800	7.620
20.98	8.780	2.750	7.680
21.09	8.770	2.800	7.640
20.96	8.770	2.800	7.560
21.28	8.800	2.740	7.580
21.13	8.780	2.800	7.600
-----	-----	-----	-----
mean 21.6179	8.7805	2.8257	7.6036

Appendix D: Test Case Schematics and
Graphic interface Images

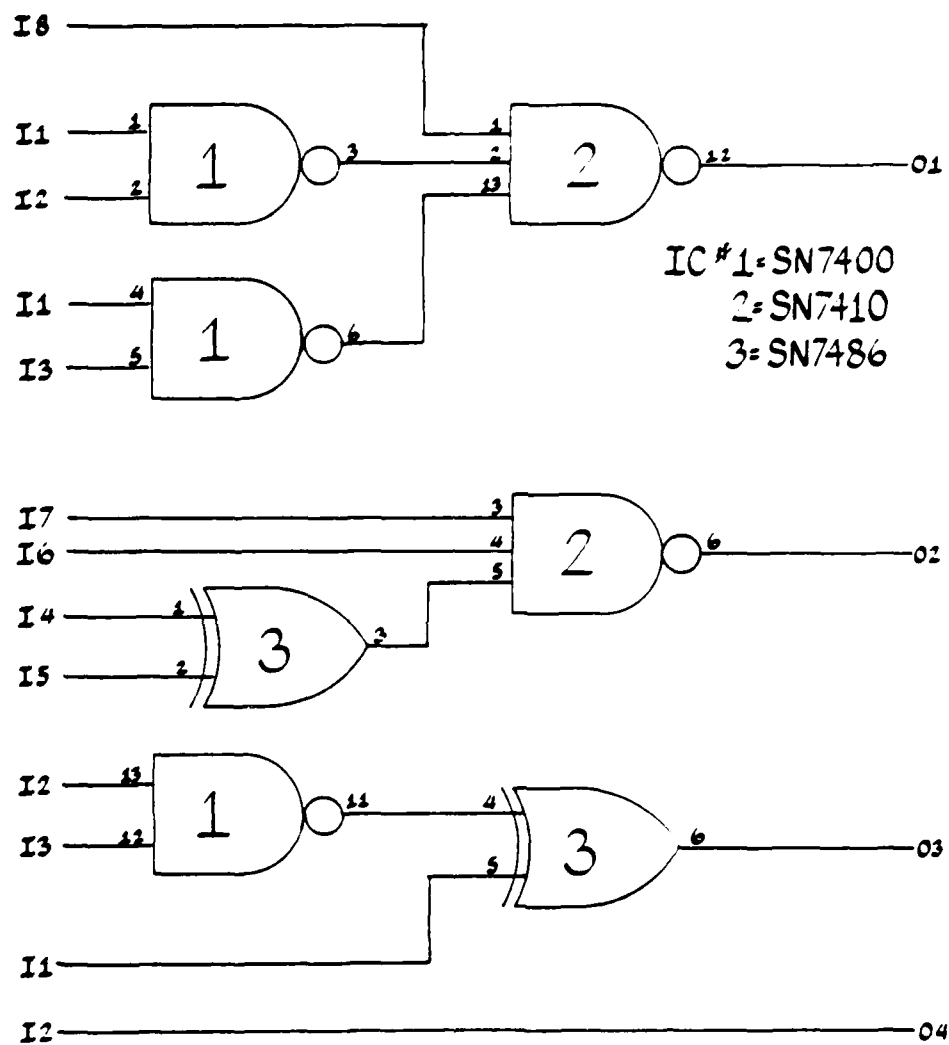


Figure D1. Schematic Diagram for BCD.CET

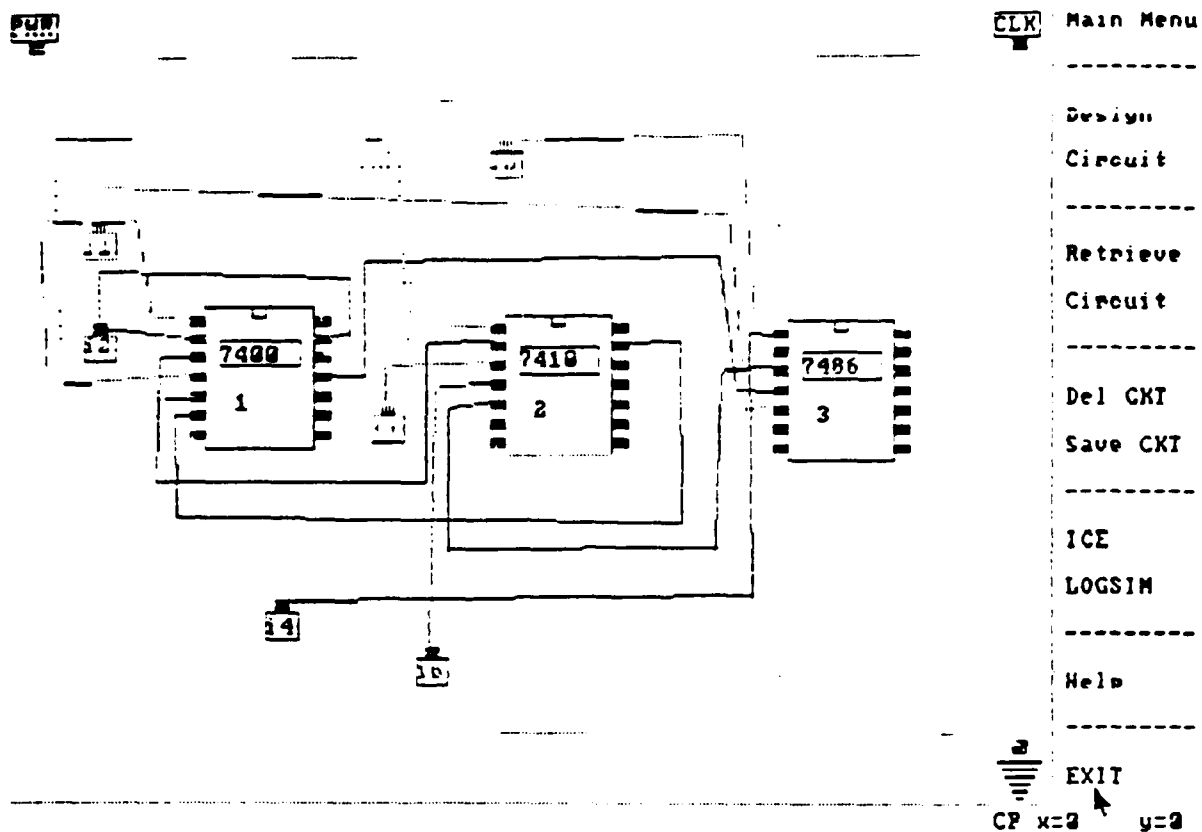


Figure D3. Graphic Circuit Image for BCD.CKT

```

I1: 01010101010101010101
I2: 00110011001100110011
I3: 00001111000011110000
I4: 00000000111111110000
I5: 10101010101010101010
I6: 11001100110011001100
I7: 11110000111100001111
I8: 11111111000000001111

```

Figure D2. Inputs for BCD.CKT

Input data streams:

Input #881	Port i1	:
Input #882	Port i2	:
Input #883	Port i3	:
Input #884	Port i4	:
Input #885	Port i5	:
Input #886	Port i6	:
Input #887	Port i7	:
Input #888	Port i8	:

Output file contents:

IC # 2 (SM 7418) PIN #12	:
IC # 2 (SM 7418) PIN # 6	:
IC # 3 (SM 7495) PIN # 6	:
Input # 2	:

Figure D5. Waveform Output for BCD.CKT

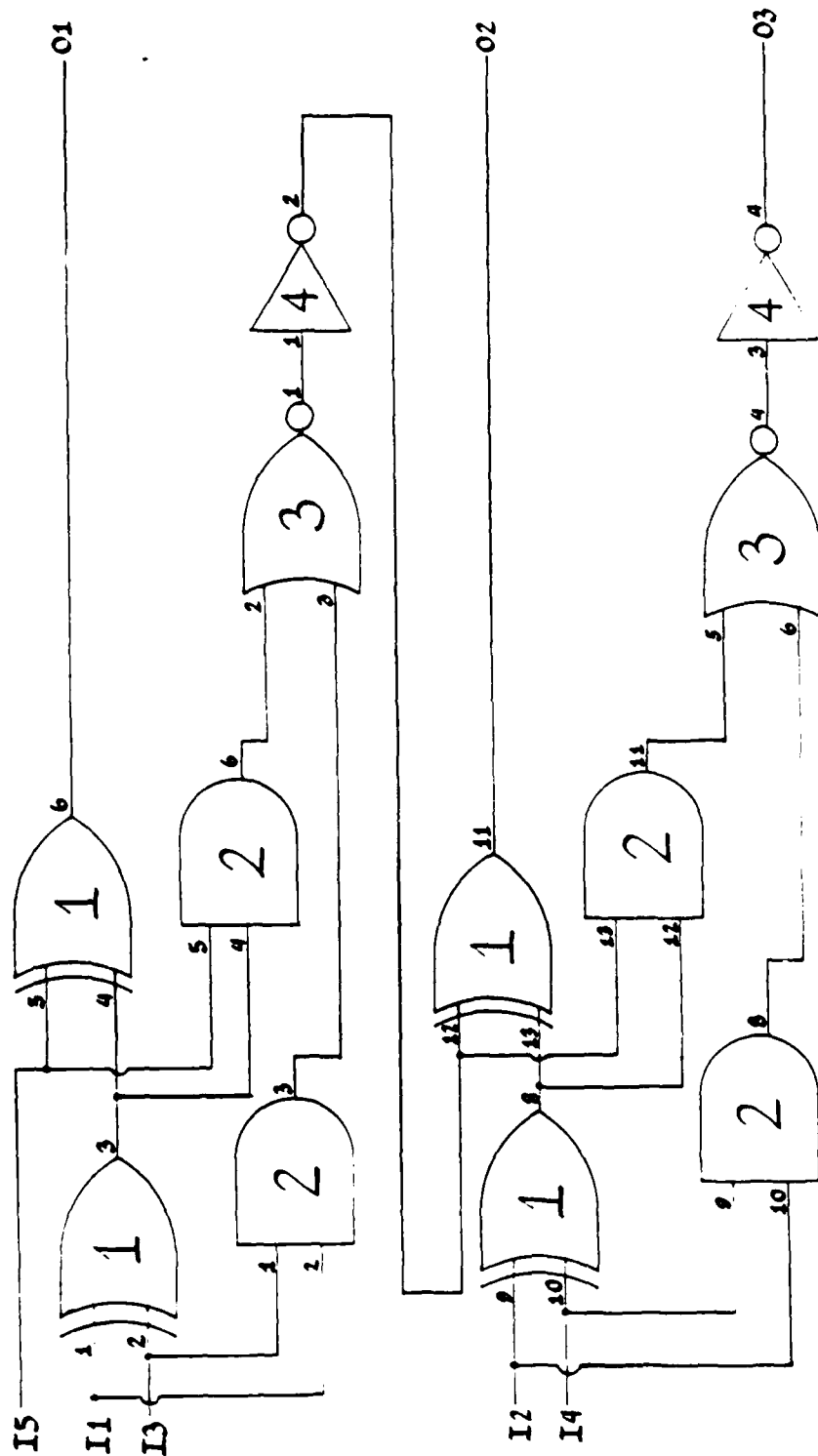
Input data files:

i1 = 001	*****:010101010101010101
i2 = 002	*****:00110011001100110011
i3 = 003	*****:00001111000011110000
i4 = 004	*****:00000000111111110000
i5 = 005	*****:10101010101010101010
i6 = 006	*****:11001100110011001100
i7 = 007	*****:11110000111100001111
i8 = 008	*****:11111111000000001111

Output file contents:

IC # 2 (SM 7418) PIN #12	:00010101111111110001
IC # 2 (SM 7418) PIN # 6	:01111111011111110111
IC # 3 (SM 7486) PIN # 6	:10101001101010011010
Input # 2	:00110011001100110011

Figure D4. Binary Output Image for BCD.CKT



IC # 1. SN7486
 2. SN7408
 3. SN7402
 4. SN7404

Figure D6. Schematic Diagram for ADDER.CKT

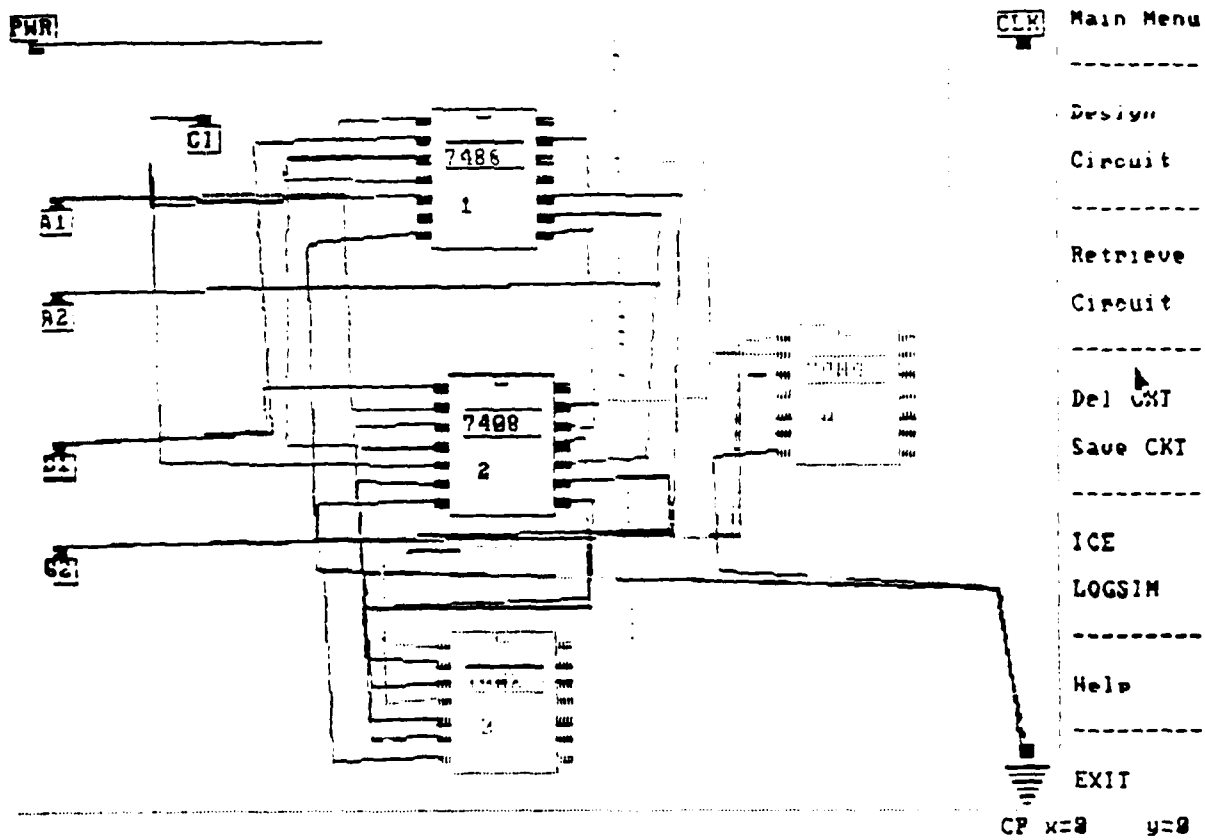


Figure D7. Graphic Circuit Image for ADDER.CKT

```

I1: 01010101010101010101010101010101
I2: 00110011001100110011001100110011
I3: 00001111000011110000111100001111
I4: 00000000111111110000000011111111
I5: 00000000000000001111111111111111

```

Figure D8. Inputs for ADDER.CKT

Input data files:

```
A1 = 001 *****:010101010101010101010101010101
A2 = 002 *****:00110011001100110011001100110011
B1 = 003 *****:00001111000011110000111100001111
B2 = 004 *****:00000000111111110000000011111111
C1 = 005 *****:00000000000000000111111111111111
```

Output file contents:

```
IC # 1 (SN 7486) PIN # 6 :01011010010110101010010110100101
IC # 1 (SN 7486) PIN #11 :00110110110010010110110010010011
IC # 4 (SN 7484) PIN # 4 :00000001001101110001001101111111
```

Figure D9. Binary Output Image for ADDER.CKT

Input data streams:

```
Input #001 Port A1 : 1111111111111111
Input #002 Port A2 : 1111111111111111
Input #003 Port B1 : 1111111111111111
Input #004 Port B2 : 1111111111111111
Input #005 Port C1 : 1111111111111111
```

Output file contents:

```
IC # 1 (SN 7486) PIN # 6 : 1111111111111111
IC # 1 (SN 7486) PIN #11 : 1111111111111111
IC # 4 (SN 7484) PIN # 4 : 1111111111111111
```

Figure D10. Waveform Output Image for ADDER.Ckt

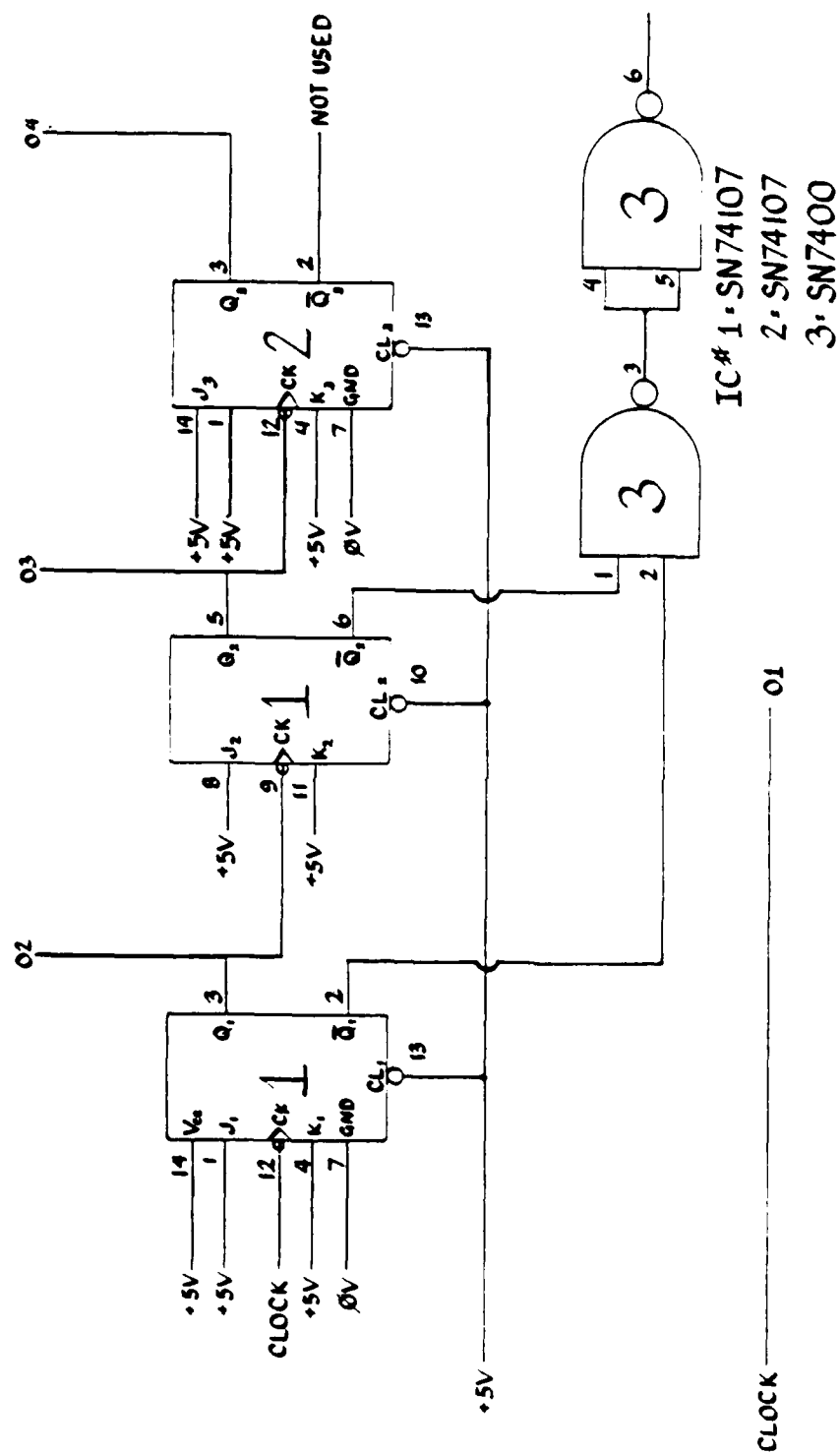


Figure D11. Schematic Diagram for BRC3S.CKT

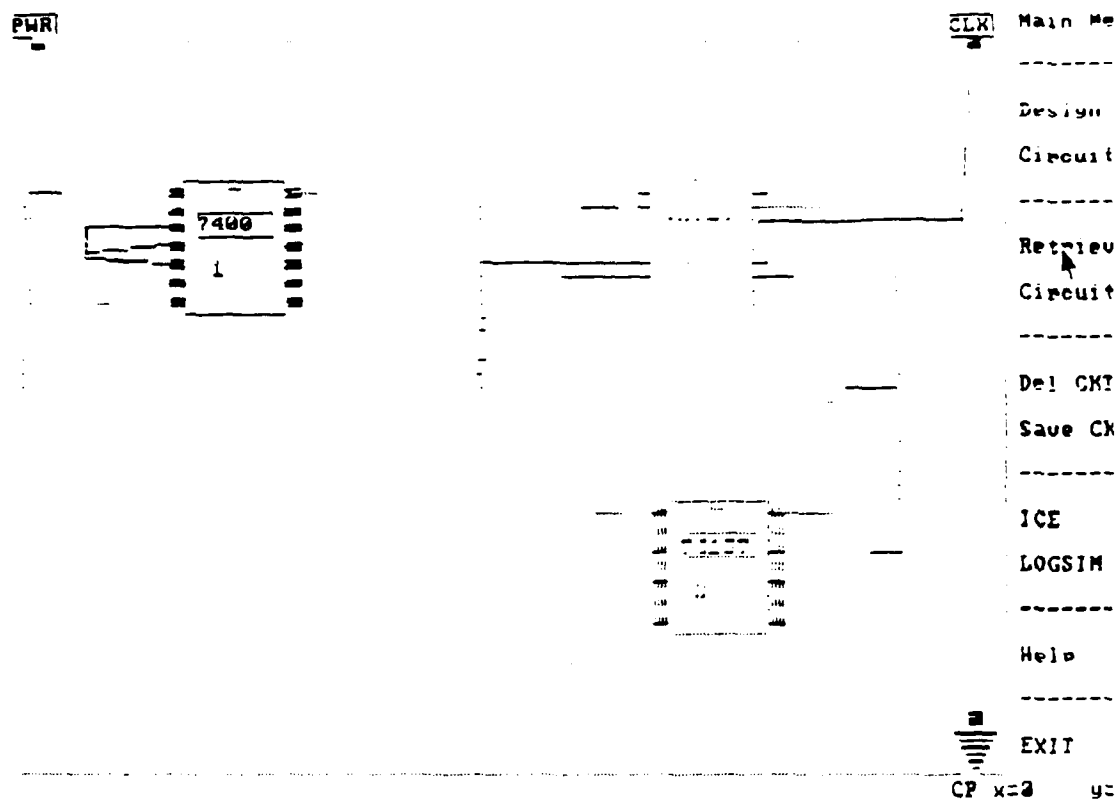


Figure D12. Graphic Circuit Image for BRC3S.CKT

Output file contents:

Clock	:010101010101010101
IC #002 (SN074107) PIN #03	:11001100110011001100
IC #002 (SN074107) PIN #05	:00111100001111000011
IC #003 (SN074107) PIN #03	:00000011111111000000
IC #001 (SN007400) PIN #06	:00000011000000110000

Figure D13. Binary Output for BRC3S.CKT

Input data streams:

No input streams supplied.

Output file contents:






Clock	: 
IC #002 (SN074107) PIN #03	: 
IC #002 (SN074107) PIN #05	: 
IC #003 (SN074107) PIN #03	: 
IC #001 (SN007400) PIN #06	: 

Figure D14. Waveform Output for BRC3S.CKT

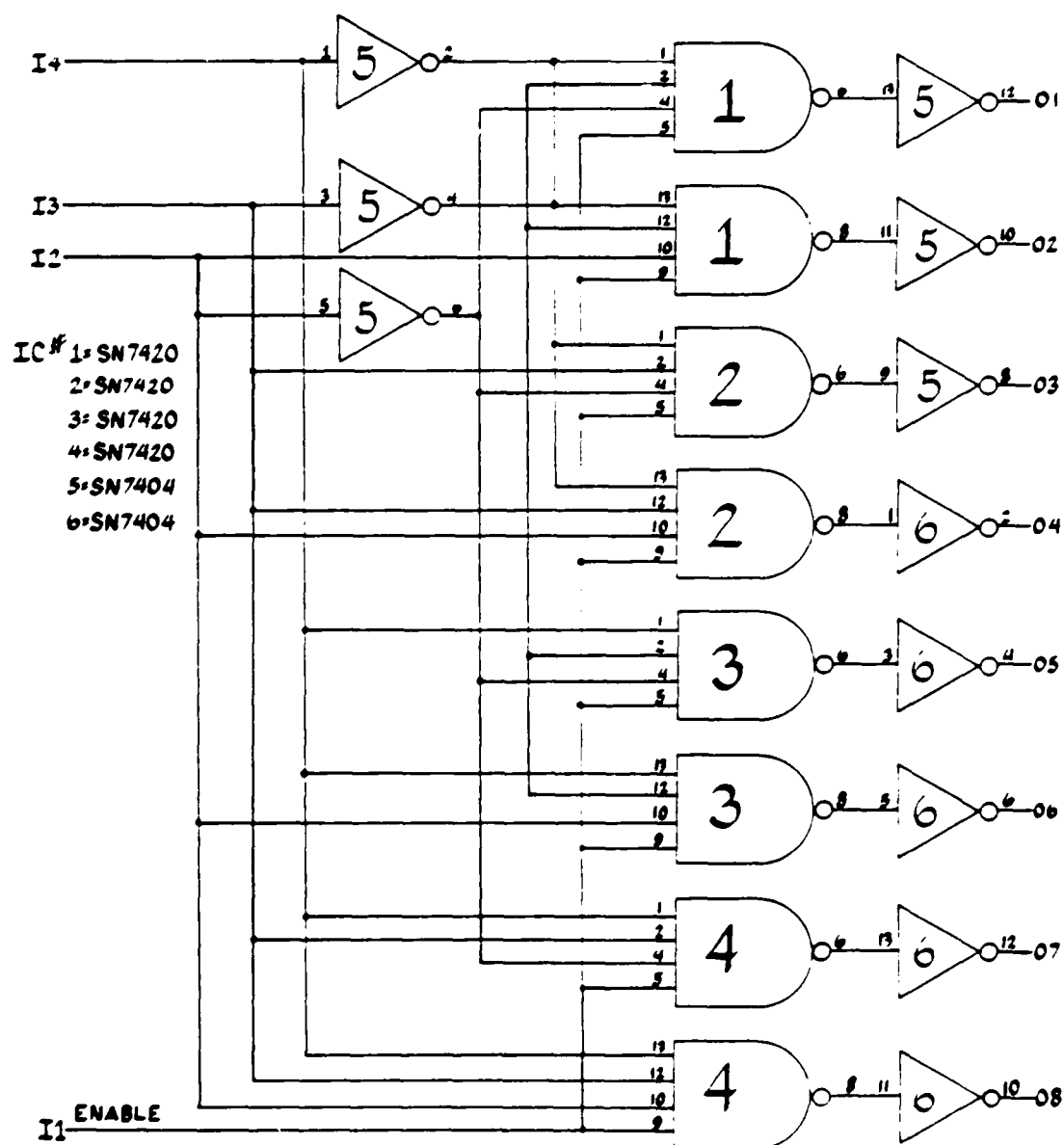


Figure D15. Schematic Diagram for DECODER.CKT

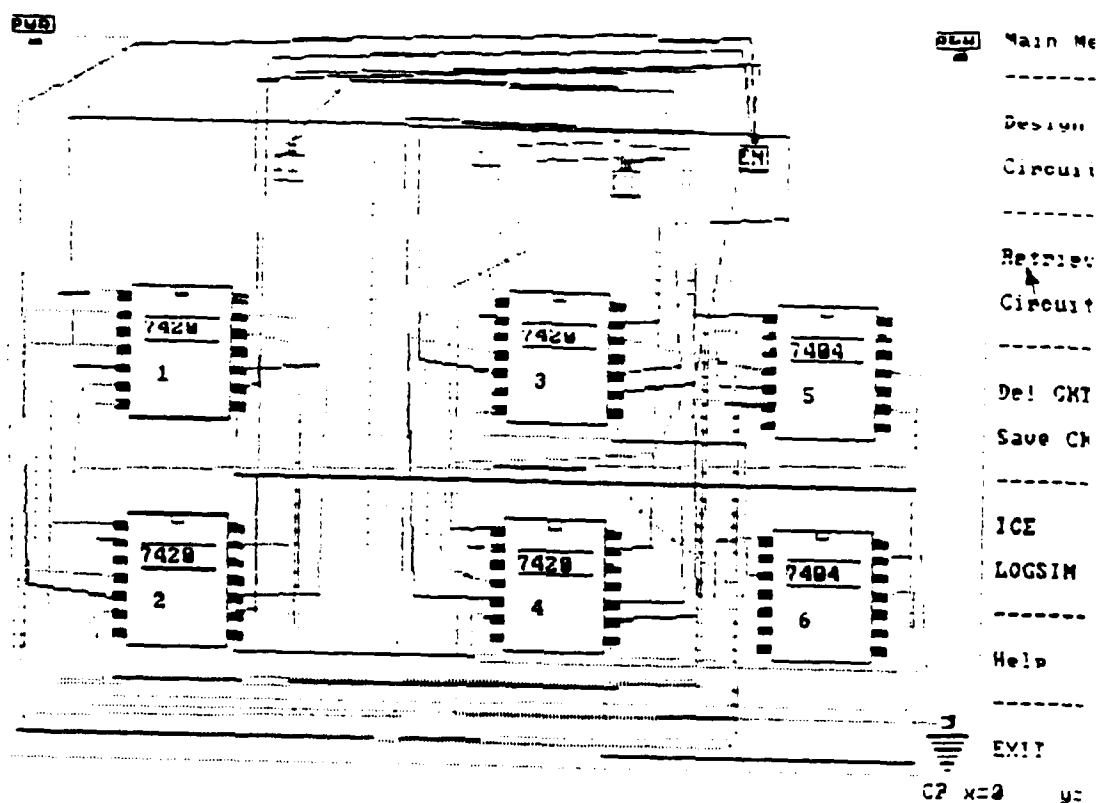


Figure D16. Graphic Circuit Image for DECODER.CKT

```

I4: 1111111100000000111111110000000011111111
I5: 01010101010101010101010101010101010101
I6: 0011001100110011001100110011001100110011
I7: 0000111100001111000011110000111100001111

```

Figure D17. Inputs for DECODER.CKT

```

EN = 004 ***** 111111110000000111111111000000011111111
11 = 005 ***** 01010101010101010101010101010101010101010101
12 = 006 ***** 00110011001100110011001100110011001100110011
13 = 007 ***** 0000111100001111000011110000111100001111

```

[illegible]

Figure D18. Binary Output for DECODER.CKT

Input data streams:

```

Input #004 Port EN : _____
Input #005 Port I1 : | | | | | | | | | | | | | | | | | | | | | |
Input #006 Port I2 : | | | | | | | | | | | | | | | | | | | | | |
Input #007 Port I3 : | | | | | | | | | | | | | | | | | | | | | |

```

Output file contents:

IC #	5	(SN	7484)	PIN #	12
IC #	5	(SN	7484)	PIN #	18
IC #	5	(SN	7484)	PIN #	8
IC #	6	(SN	7484)	PIN #	2
IC #	6	(SN	7484)	PIN #	4
IC #	6	(SN	7484)	PIN #	6
IC #	6	(SN	7484)	PIN #	12
IC #	6	(SN	7484)	PIN #	18

Figure D19. Waveform Output for DECODER.CKT

Appendix E: LOGSIM248 Manual for Independent Simulation

LOGSIM248 MANUAL FOR INDEPENDENT SIMULATION

E.1 Overview

LOGSIM248 is a new implementation of LOGSIM, version 5.5, which is a digital design logic simulation program developed by students of the Air Force Institute of Technology. All source code for this new implementation is written in the C programming language and developed using the Borland International TURBO C Integrated Development Environment.

LOGSIM248 is used to simulate sequential or combinational digital circuits comprised of TTL (Transistor to Transistor Logic) Integrated Circuit (IC) packages. Presently, the IC library within LOGSIM248 consists of 32 separate packages from the 7200 family of integrated circuits.

This simulation program may be run as an independent simulator or as an integral part of the larger digital design tool, IDIET (Integrated Digital Engineering Tool). Presently, no user interface is built into LOGSIM248 to enable interactive circuit entry. All data used by the simulator is supplied by four ASCII files. Three of these files are required, while the fourth is optional. LOGSIM248 results are written to two additional ASCII files and are available for user analysis.

The four input files must be available to LOGSIM248 prior to invocation of the simulator for proper independent execution. Some sort of file editor capable of creating ASCII files will be required by the user for creation of these files. Word processors which can convert formatted files to ASCII will work perfectly, provided no control characters (characters used for printer manipulation) are imbedded in the converted files.

The following sections of this manual present LOGSIM248 operation, materials required to run LOGSIM248, the input/output data file interface, file format and construction, and possible error conditions resulting from incorrect operation. Examples are presented as needed and are based on those test cases used for LOGSIM248 simulator development.

E.2 What Is LOGSIM248?

LOGSIM248 is a digital logic simulator used to simulate digital circuit designs which use TTL IC packages as major components. As the term "logic" implies, this simulation tool performs no timing simulation -- it is primarily designed to confirm expected output and circuit design by supplying the user with binary output streams based solely upon input data. Data is input in one or more binary streams corresponding to user defined input ports. This data is then

operated upon one bit (or set of bits for multiple inputs) at a time to produce output in the form of binary or waveform data streams.

All input data is supplied through the input file interface, and all output presented through similar output files. Under ideal conditions, these input files are created through the integrated digital design tool, IDIET. This tool combines a graphic user interface, and a connectivity checker, as well as, LOGSIM248. The graphic interface allows the user to draw the circuit on a computer monitor screen using both menu and mouse directives, after which the expert system may be invoked to check connections prior to simulation. Further explanation of this integrated design tool is beyond the scope of this manual. For a more detailed explanation of the functions and capabilities of IDIET, refer to "A Graphic User Interface for Digital Circuit Design: User's Manual", by Charles Adams.

It is not necessary to own an executable version of all components of IDIET to simulate circuits with LOGSIM248. All that is required is knowledge of the LOGSIM248 file interface, a text editor, and an executable version of the simulation program.

E.3 Materials Required

LOGSIM248 runs on all IBM PC, XT, and AT compatibles utilizing the MS/DOS operating system, therefore, it is

necessary for the user to have one of these of computers at his disposal. Of course, an executable version of LOGSIM248 is also required. This executable version may be obtained upon request by writing to:

Air Force Institute of Technology
School of Engineering
Department of Engineering and Computer Science
Wright-Patterson AFB, Ohio 45433

and enclosing one 5 1/4 inch diskette.

LOGSIM248 may be invoked from either a floppy disk or hard disk provided all input files are located within the same directory as the executable version of the simulator. Upon completion of the simulation, all output files will also be located in this directory, so it is important to insure plenty of storage room for this purpose.

Finally, the user will require the use of a text editor or word processor capable of producing ASCII files. This capability is required to produce the input files used during LOGSIM248 execution. All files produced must be pure ASCII files with no extraneous control characters imbedded in the text. If word processors are used they must be able to conform to this requirement or LOGSIM248 will not function properly.

E.4 LOGSIM248 File Interface

As mentioned above, LOGSIM248 uses four files as input. The data from these files is used to; (1) construct the circuit, (2) identify the output monitor points, (3) provide input data to the simulator, and (4) construct the files containing output data. All output data is contained in two separate files. One of these contains the output displayed in binary streams (ones and zeros), while the other displays each output stream graphically, in square-wave form. All input and output data is obtained from ASCII files containing no imbedded control characters or executable machine code. These files can be viewed or created through the use of any text editor or word processor with ASCII file production capability. When creating these files it is imperative that all data is entered as presented in the following sections. Any errors in these input files will either cause the simulator to produce erroneous results or cause the program to abort.

LOGSIM248 input/output file information is presented in Table E-1. This table shows the file name, file type (input or output), file entry format (delimited by []), and examples of file entries. File format entry codes are shown in Table E-2.

E.4.1 Circuit Configuration Input File - TEMP.CKT

The first of the input files, and perhaps most important, is TEMP.CKT. This file contains all input information required to build the circuit. All IC's and connections between these IC's are specified within this file. Additionally, connections between user defined input ports, power, ground, and clock must be contained in this file.

All entries to TEMP.CKT are composed of two fields separated by a full colon and followed by a new-line character (carriage return). Each of the two fields describe both ends of a particular connection within the circuit being designed. Each field consists of four sub-fields as pictured in Table E-1.

The first sub-field contains a capital letter immediately followed by a three digit number. This four character field describes where that end of the connection comes from. If the letter used is a 'T' then that end of the connection originated at a TTL IC package. The letter 'I' means an input, 'C' is the clock, 'P' is power, and 'G' is ground. The three digit number identifies the particular input port or TTL IC being referenced by this sub-field. For (P)ower, (G)round, and (C)lock this three digit number is inconsequential and may be anything. The number need not consist of three digits, but all position in the field must be filled with either a digit or a space. Padding with zeros






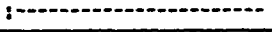



Table E-1. Data File Formats (see Table E-2. for format codes)	
<i>temp.ckt</i>	-- circuit input file - format [dnnnttttttpp comment:dnnnttttttpp comment] - example: P000 power +5Vdc :T001 740014 power Vcc
<i>temp.dis</i>	-- output monitor file - format: [dnnnttttttpp comment:] - examples: T001 740003 output pin #3 : I003 input #3 :
<i>temp.in</i>	-- input data stream file - format: [nnn:iiiiiiiiiii...] - example: 001:1010101010101010101010101010
<i>temp.out</i>	-- binary output file - format: [IC #nnn (SNnnnnn) PIN #nn :ooo...] [Input nnn :ooo...] [Clock :ooo...] [Power :ooo...] [Ground :ooo...] - examples: IC # 1 (SN 7400) PIN #13 :01101110011110000110011 Input 003 :10011000011100110001101 Clock :01010101010101010101010 Power :111111111111111111111111 Ground :000000000000000000000000
<i>temp.wav</i>	-- waveform output file - input stream description format: [Input #nnn Port cc :iii...] - output stream description format: [IC #nnn (SNnnnnn) PIN #nn :mww...] [Input nnn :mww...] [Clock :mww...] [Power :mww...] [Ground :mww...] - example: Input file contents: Input #001 Port A1 :  ----- Output data streams: IC # 1 (SN 7400) PIN #13 :  IC # 13 (SN74181) PIN #13 :  Input 003 :  Clock :  Power :  Ground : 

Table E-2. File Format Codes	
codes:	d = connector descriptor (1 character) T : TTL IC I : input port P : power (+5Vdc) G : ground (0Vdc) C : clock n = IC or input number (3 digits) t = IC type (6 digits) optional for input, power, ground, clock p = pin number (2 digits) optional for input, power, ground, clock comment = optional 20 character comment field i = input value, logical value 1 or 0 o = output value, binary logical value 1 or 0 w = output value, waveform logical value 1 =  logical value 0 = 

is a good practice for insuring proper field width. When padding, it is good practice to left justify the number; ex. T001, I 2, G000, etc..

The second sub-field consists of a six digit number describing the type of TTL IC being used for this connection. Because some IC type numbers are only four or five digits long (i.e., 7400, 74107) the same rules of padding hold here as above. It is important here to insure that the number entered is indeed in the IC library used by the simulator. A list of the 32 IC's presently contained in the library is provided as Annex A to this manual. If the end of the connection in question is connected to the clock, power, or ground, these six positions may be left blank or filled with any character.

Following the IC type indicator is a two digit number representing the particular pin on the IC being used for this connection. The same padding rules can be used here as above. These two positions may also be left blank in the case of power, ground, or clock. If the number entered here exceeds the number of pins normally found on IC packages of the type specified in the IC type sub-field, the simulator will not perform properly.

The next 20 positions of each field are not used by the simulator and may be used for comments by the designer. These positions have been included for future use or circuit specific details not contained in the other sub-fields.

An example of this file is contained in Annex B of this manual. It should be noted that all connections to the power (Vcc) and ground (GND) pins are listed in this example file. Although these are not required by the simulator (the simulator assumes power and ground connections for Vcc and GND pins only) it is good practice to include them.

E.4.2 Output Specification File - TEMP.DIS

In order to view expected output, it is necessary to specify those places in the circuit where this output will originate. These output monitor points are specified in the file TEMP.DIS. All entries in this file consist of one field terminate with a full colon. Each entry in this file is exactly like one field of the entries in TEMP.CKT. Only the left field is used and all sub-fields are the same as those described in the previous section. An example of this file is also contained in Annex B.

E.4.3 Input Data File - TEMP.IN

TEMP.IN contains all input data from each input port required for proper circuit simulation. Although some circuit designs may take all input data from the power supply, clock, or ground (electrical low), this file still must exist. If no input ports have been specific in the

circuit then no data need be present in this file. An empty file in this circumstance is needed for proper functioning of the circuit.

The entries in this file are comprised of two fields separate by a full colon. The first field contains a three digit number specifying an input port for the data stream which follows in the second field. The input port number must be left justified in the three digit field and padded with zeros. The input data stream for this input port must immediately follow the colon and may be as long as 80 characters. This data field may only contain zeros and ones which represent electrical high and low. Any other characters will not be correctly interpreted by the simulator and will produce erroneous results. An example of this file is also contained in Annex B.

E.4.4 Input Port Label File - TEMP.IND

This file contains the labels used to identify input ports. These labels are two character identifiers which will be used in the output waveform file. This is an entirely optional file. The simulator tests for the presence of this file and uses it only if it exists. However, if it does exist it must have correct data in it. The absence of data or an incorrect number of input port labels may cause simulation run time errors.

Entries in this file consist of two characters followed by any amount of information (up to 80 characters total) and a new-line character (carriage return). Only the first two characters of the line are read by the simulator. The rest of the line may be used for any purpose the designer wishes. The example in Annex B was created by the integrated digital design tool, IDIET, and contains additional information required by the graphic user interface component of that tool. This information is not mandatory.

These are the only files required for effective operation of LOGSIM248. A complete example consisting of a schematic circuit diagram, the three input files, and the two output files is contained in Annex B.

E.5 Invoking the Simulator

All that is required to invoke the simulator is a copy of the executable file, `logsim.exe`. Those input files described above should be created and placed in the same directory as `logsim.exe`. If these files are not in the same directory, the simulator will not be able to find them and the output files will contain an error message alerting the user.

At the computer prompt the user need only type:

```
A>logsim
```

followed by a carriage return. LOGSIM248 will simulate the circuit specified by the input files and return the system prompt when finished. After the prompt has reappeared on the screen, the output files, TEMP.OUT and TEMP.WAV, are available for viewing.

E.6 Summary

LOGSIM248, a digital logic simulator for simulating digital circuits comprised of TTL IC packages, was designed for ideal performance when incorporated as a component of the integrated digital design tool, IDIET. However, the simulator may still be utilized in the absence of this tool. This short manual has explained the operation and creation of interface data files for this simulator and presents examples in the following Annexes. All IC packages contained in Annex A may be used to construct a myriad of sequential and combinational digital circuits. These circuits can be efficiently simulated using this manual, a test editor, and a copy of the executable version of LOGSIM248.

ANNEX A: LOGSIM248 TTL IC Library

- 7400 - Quad 2-Input positive NAND Gate.
- 7402 - Quad 2-Input positive NOR Gate.
- 7404 - Hex Inverters.
- 7408 - Quad 2-Input positive AND Gate.
- 7410 - Triple 3-Input positive NAND Gate.
- 7420 - Dual 4-Input positive NAND Gate.
- 7425 - Dual 4-Input positive NOR Gate with strobe.
- 7427 - Triple 3-Input positive NOR Gate.
- 7430 - Single 8-Input positive NAND Gate.
- 7442 - Single BCD to DECIMAL (4 line to 10 line Decoder).
- 7483 - Single 4-Bit Binary Full adder with fast carry.
- 7486 - Quad 2-Input Exclusive-OR Gate.
- 7489 - Single 64-Bit Read/Write Memory.
- 7493 - Single 4-Bit binary counter.
- 7495 - Single 4-Bit Shift Register.
- 74107 - Dual J-K Flip Flop with clear.
- 74109 - Dual J-K positive edge-triggered Flip Flop with preset and clear.
- 74116 - Dual 4-Bit Latches.
- 74135 - Quad Exclusive-OR/NOR Gates.
- 74151 - Single 1-of-8 Data Selector/multiplexer.
- 74153 - Dual 4-Line to 1-Line Data Selector/multiplexer.
- 74157 - Quad 2 to 1-Line Data Sel/mult (Non-Inverted Data Outputs).
- 74163 - Synchronous 4-Bit Counter (Binary, synchronous clear).
- 74175 - Quad D-Type Flip Flop.
- 74181 - Arithmetic Logic Units/Function Generators.
- 74183 - Dual Carry Save Full Adders.
- 74193 - Synchronous Up/Down Dual Clock Counter (Binary with clear).
- 74194 - Single 4-Bit Bidirectional Universal Shift Register.
- 74274 - Single 4-Bit by 4-Bit Binary Multiplier.
- 74279 - Quad (Inv)S - (Inv)R Latch.
- 74284 - Single 4-Bit by 4-Bit Parallel Binary Multiplier used with '285'.
- 74285 - Single 4-Bit by 4-Bit Parallel Binary multiplier used with '284'.
- 74298 - Quad 2-Input Multiplexer with storage.
- 74378 - Hex D-Type Flip Flop.

Figure E-1. LOGSIM248 TTL IC Library

ANNEX B: Example Input/Output File Structure

```

P 0 POWER 0 1*****:T 3 748614*****
P 0 POWER 0 2*****:T 2 741014*****
P 0 POWER 0 3*****:T 1 740014*****
T 1 7400 7 4*****:G 0GROUND 0*****
T 2 7410 7 5*****:G 0GROUND 0*****
T 3 7486 7 6*****:G 0GROUND 0*****
T 1 7400 3 7*****:T 2 7410 2*****
T 2 7400 3 8*****:T 2 741013*****
T 3 7486 3 9*****:T 2 7410 5*****
T 1 740011 10*****:T 3 7486 4*****
T 1 740011 11*****:T 1 7400 1*****
I 1 740011 12*****:T 3 7486 5*****
I 1 740011 13*****:T 1 7400 2*****
I 1 740011 14*****:T 1 740013*****
I 2 740011 15*****:T 1 740012*****
I 2 740011 16*****:T 1 7400 5*****
I 3 740011 17*****:T 3 7486 1*****
I 4 740011 18*****:T 3 7486 2*****
I 5 740011 19*****:T 2 7410 4*****
I 6 740011 20*****:T 2 7410 3*****
I 7 740011 21*****:T 2 7410 1*****
I 8 740011 22*****:T 2 7410 1*****

```

Figure E-3. Circuit Configuration Input File - TEMP.CKT

```

T 2 741012*****:
T 2 7410 6*****:
T 3 7486 6*****:
I 2 i200*****:

```

Figure E-4. Output Monitor File - TEMP.DIS

```

001:010101010101010101010101
002:001100110011001100110011
003:00001111000011110000
004:00000000111111110000
005:101010101010101010101010
006:110011001100110011001100
007:11110000111100001111
008:11111111000000001111

```

Figure E-5. Input Data File - TEMP.IN

```

Input data files:
i1 = 001 *****:010101010101010101010101
i2 = 002 *****:001100110011001100110011
i3 = 003 *****:00001111000011110000
i4 = 004 *****:00000000111111110000
i5 = 005 *****:101010101010101010101010
i6 = 006 *****:110011001100110011001100
i7 = 007 *****:11110000111100001111
i8 = 008 *****:11111111000000001111

```

Figure E-6. Input Port Label File - TEMP.IND

Output file contents:

IC #	2	(SN	7410)	PIN #	12	:	00010101111111110001
IC #	2	(SN	7410)	PIN #	6	:	01111111101111110111
IC #	3	(SN	7486)	PIN #	6	:	10101001101010011010
Input #	2					:	00110011001100110011

Figure E-7. Binary Output File - TEMP.OUT

Input data streams:

Input #001	Port 11	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #002	Port 12	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #003	Port 13	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #004	Port 14	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #005	Port 15	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #006	Port 16	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #007	Port 17	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #008	Port 18	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■

Output file contents:

IC #	2	(SN	7410)	PIN #	12	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
IC #	2	(SN	7410)	PIN #	6	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
IC #	3	(SN	7486)	PIN #	6	:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■
Input #	2					:	■ ■ ■ ■ ■ ■ ■ ■ ■ ■

Figure E-8. Waveform Output File - TEMP.CKT

VITA

Captain Wayne C. DeLoria was born on 27 January 1954 in Chicopee, Massachusetts. He graduated from high school in 1972 and enlisted into the U.S. Army in March 1974. He served for six years as a Non-Morse Communications Analyst completing two overseas assignments. After attaining the rank of Sergeant, Captain DeLoria was accepted to Officer Candidate School from which he graduated on 29 February 1980. He then served as a Project Officer and Headquarters Company Commander for the Electronic Material Readiness Activity, Vint Hill Farm Station, Virginia. In 1984 he entered James Madison University through the full time Degree Completion Program. After graduating in December 1986 with a Bachelor of Science Degree in Computer Science, Captain DeLoria went on to attend The Air Force Institute of Technology, School of Engineering.

Permanent address : 36 Hillside Avenue

Chicopee, Massachusetts 01020

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/87D-10			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG		7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583				7b. ADDRESS (City, State, and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)				10. SOURCE OF FUNDING NUMBERS	
				PROGRAM ELEMENT NO.	PROJECT NO.
11. TITLE (Include Security Classification) See Box 19					
12. PERSONAL AUTHOR(S) Wayne C. DeLoria, B.S., Captain, USA					
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1987 December	
15. PAGE COUNT 205					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Digital Simulation, Parallel Processing		
FIELD	GROUP	SUB-GROUP			
12	05				
12	06				
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Title: A DIGITAL LOGIC SIMULATOR WITH CONCURRENT PROGRAMMING CONSIDERATIONS (UNCLASSIFIED) Thesis Chairman: Nathaniel J. Davis IV, Captain, PhD, USA ECE Assistant Professor of Electrical and Computer Engineering					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL CPT Nathaniel J. Davis IV			22b. TELEPHONE (Include Area Code) (513) 255-3576		22c. OFFICE SYMBOL AFIT/ENG

Approved for public release: 1AW AFR 100-17.
 31 Dec 87
 100-17
 100-17
 100-17
 100-17

UNCLASSIFIED

Block 19. (cont.)

The digital logic simulator, LOGSIM248, a re-engineered version of LOGSIM, version 5.5, has been implemented as a component of the digital design environment, IDIET (Integrated Digital Engineering Tool). This new design expands the capabilities of the older version by improving run time performance, maintainability, and compatibility. Written in the C programming language, LOGSIM248 boasts looser coupling between functional modules while exhibiting greater functional cohesion within these modules. As an integral part of IDIET, the simulator overcomes difficulties created by the complicated user interface of earlier versions.

With greater run time performance as a goal, this new simulator was studied and adapted to produce a concurrent implementation. Here, several roadblocks were encountered which essentially showed this algorithm and data structure implementation to be difficult to "parallelize" at best. Due to communication constraints on the host computer, data structures used to simulate circuits caused large delays due to the requirement to disassemble and re-assemble them at the various processing nodes. This program handicap coupled with communication transmission delays between processors resulted in time complexity problems.

Essentially a software engineering project, the re-design of LOGSIM, version 5.5, was necessitated by various shortcomings associated with the older version. The new implementation conforms to the proposed ANSI standard for the C programming language by utilizing only standard library functions and source code which complies with the original Kernighan and Ritchie model. This re-hosting has improved system portability allowing LOGSIM248 to run on all MS/DOS micro-computers available to the designer.

UNCLASSIFIED

END

DATE

FILMED

MARCH

1988

DTIC