1988

A timing simulator

Kathryn D. Heintz

A TIMING SIMULATOR

by

Kathryn D. Heintz

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

May 19, 1988

Approved by:

Nan C. Schaller	5-19-88
Nan Schaller, Chair	date
George A. Brown	May 19, 1988
George Brown	date
Rayno Miemi	May 19, 1988
Rayno Miemi	date
Peter G. Anderson	19 May 88
Peter G. Anderson	date

Title of Thesis: A Timing Simulator

I, Kathryn D. Heintz , prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

21 Del Verde Road
Rochester, New York 14624
Phone 247-2436

July 6, 1988

ABSTRACT

A timing simulator, called TIMSIM, has been developed which performs gate level simulation of simple digital logic circuits. TIMSIM has a library of twelve standard TTL gate elements and memory elements. These elements incorporate various features including single outputs, multiple outputs, non-symmetric inputs, and memory states. TIMSIM uses a rise-fall delay model and three values to represent a signal's logic level.

CR Categories: B.6.3, B.7.2

Key Words and Phrases: timing simulation, gate level simulation, event-driven algorithm, design verification, propagation delay

TABLE OF CONTENTS

Chapter 1 Introduction..... 1
1.Ø Overview of Problem 1
1.1 Objectives of Project 1
1.2 Relationship to Other Software..... 2
1.3 Design Overview..... 3
1.4 Outline of Thesis..... 5

Chapter 2 Historical Overview..... 6
2.Ø Introduction..... 6
2.1 The Simulation Approach - Issues and Directions... 6
2.2 Logic Simulation Overview..... 8
2.2.1 History of Timing Simulation..... 9
2.2.2 Evolution of Algorithms To Do Timing Simulation 11
2.2.3 Projects..... 12
2.3 State of the Art in Timing Simulation..... 14
2.3.1 Multi-level Simulation..... 14
2.3.2 Design Verification..... 14
2.3.3 Languages..... 15
2.3.4 Special Purpose Multiprocessors..... 16
2.3.5 Data Flow Approach..... 16
2.3.6 Logic Simulation Machines..... 16
2.3.7 Microprocessor Logic Simulators..... 17
2.4 Summary..... 18

Chapter 3 Project Development..... 19
3.1 Introduction..... 19
3.1.1 Algorithms, Languages and Elements..... 19
3.1.2 Types of Delay Models..... 2Ø
3.1.3 Multi-valued Logic..... 22
3.2 Essential Features of the Simulator..... 24

Chapter 4 Implementation..... 26
4.Ø Overview..... 26
4.1 Inputs and Outputs..... 26
Fig. 4.1 A Circuit and its Description..... 27
Fig. 4.2 Data Flow Diagram..... 28
4.2 Files..... 3Ø
Fig. 4.3 File Structure..... 3Ø
4.3 Modules in the Parser..... 32
Fig. 4.4 Structure Chart of the Parser..... 32
4.4 Modules in the Simulator..... 34
Fig. 4.5 Structure Chart of the Simulator..... 34
4.5 Important Data Structures..... 36
4.5.1 Library..... 36
Fig. 4.6 Library Data Structure..... 36
Fig. 4.7 Timing Delays..... 37
4.5.2 Circuit and Table..... 37
Fig. 4.8 Circuit and Table Data Structures..... 38

4.5.3 Event Scheduling Mechanism..... 39
Fig. 4.9 Timewheel Data Structure..... 39
4.6 Event Scheduling Process..... 39
Fig. 4.10 Event Scheduling Flow Chart..... 40

Chapter 5 Testing..... 41
5.0 Testing Strategy..... 41
5.1 Testing the Parser..... 41
5.2 Testing the Simulator..... 43
5.3 Test Comparisons with Other Simulators..... 44

Chapter 6 Conclusions..... 45
6.1 Summary of the Project..... 45
6.2 Problems Encountered and Resolved..... 46
6.2.1 Multi-valued Logic..... 46
Fig. 6.1..... 47
6.2.2 Initialization or Simulation Startup..... 49
6.2.3 Event Scheduling Analysis..... 50
6.3 Further Work..... 51

Appendices

A. Users' Manual
B. Sample Runs
C. Analysis of the Three-Valued Truth Table for the 7474 D-type Flip-flop

CHAPTER ONE

INTRODUCTION

1.0 Overview of Problem

Hardware simulation is often a useful tool in an academic environment because of the expense involved in hardware implementation. This paper discusses the development of a simulation tool, a timing simulator, which could be used by students in introductory digital logic courses, for design verification of simple circuits.

1.1 Objectives of the Project

As students are introduced to logic circuits, they are required to design simple combinational and sequential circuits using AND gates, OR gates, flip-flops, inverters, etc. The interaction of the various circuit components with respect to timing is a critical design consideration. Students need to understand how a change in an input will affect the output and, in addition, how the propagation delays associated with each of the gates affect the operation of the circuit. (Propagation delay is the time it takes for the output of a gate to change after the input has changed states.) Interactively changing the values of the

inputs and observing the effects on the outputs enhances the understanding of the circuit. In some designs, unexpected and undesirable hazards or race conditions may develop which require analysis and redesign. Hazards and races are "temporary, unwanted output signals which may occur in logic circuits due to different delays in different signal paths." (Schwarz 1987)

The objective of this project has been to investigate the issues involved in developing a logic level timing simulator and to design and build the simulator. This process involved choosing and modifying a simulation algorithm to do event list simulation, understanding the requirements of simulating memory chips and developing appropriate input and output strategies. Also considered were various levels of complexity in logic simulation and the different types of delays - propagation, transition and transmission delays - which are inherent in a logic circuit. The implications of three-valued logic were studied.

1.2 Relationship to Other Software

Many simulators have been developed to aid the circuit designer, most of a more complex nature than the one discussed here. The relative simplicity of this simulator is in keeping with a student user's level of understanding of circuit design. This simulator has been designed to supplement the VLSI design tools made available by the

University of Washington Consortium and installed on RIT systems. These design tools include a circuit simulation program called RNL and a network description program called NETLIST.

This simulator, which has been named **TIMSIM**, uses the same format that NETLIST uses to create a circuit description file which expresses the input circuit to be simulated. The simulation is done in a series of steps with a report issued at the end of each simulation step. This step procedure is similar to RNL. The commands used to control the simulation and the output formats are different from those of RNL. (For further information, see NETLIST and RNL documentation.)

1.3 Design Overview

TIMSIM consists of two programs, a parser and a simulator. Each can be run as a separate program.

The parser program reads the circuit description input, and decodes and expands it. The input to the parser is a circuit description file created by the user which uses a Lisp-like syntax to list each element with its associated input and output nodes. The output of the parser is an intermediate file which is an expanded version of the input circuit description.

The simulator program reads the intermediate file and loads the data structures which will support the simulation.

The user then interactively manipulates the circuit simulation by setting the values of the inputs and letting these signals propagate through the circuit. The output of the simulator is a summary of which nodes changed, the values and the time of the change. The user has a choice of three formats for the output as well as a choice of screen or file output.

The simulation responds to changes in input values initiated by the user and incorporates propagation delays specific to each individual gate. These delay times are randomized within prescribed limits in order to approximate a real world situation where there is some variation in delays.

There are several data structures used in the simulator. A library data structure contains all the information required for each of the TTL elements in the simulator. The library data structure includes the timing delays, functionality information, the number of inputs and outputs, etc. A table stores information about each node, its present value, its former value, and its relationship to the other elements. A circuit data structure includes each element and its connections to other elements. The future event list, or time-wheel, is a circular array of pointers to linked lists which contains the schedule of future events. An event is a change in the value of a node.

The simulator will handle combinational circuits using ANDs, ORs, inverters and also sequential logic including flip-flops and latches.

1.4 Outline of Thesis

Chapter 2 gives an overview of the field of circuit simulation and the development of logic level simulators in particular. Chapter 3 is a discussion of some of the issues investigated and some of the assumptions made in the development of the simulator. The program organization and the details of the implementation of the simulator are discussed in Chapter 4. The functional description of the programs, structure charts, and data structure diagrams are also included. Chapter 5 discusses the testing strategies used for each section of the program and the test results. Chapter 6 summarizes the paper and discusses possible extensions.

Three appendices are included. Appendix A is a complete User's Manual developed to make the simulator as user-friendly as possible with examples and sample runs. Appendix B has several runs of the program using some of the test circuits. A detailed analysis of the ternary (3-valued) truth table for the 7474 d-type flip-flop is shown in Appendix C. The source code is not included with the thesis. A listing is available upon request.

CHAPTER TWO

HISTORICAL OVERVIEW

2.0 Introduction

The beginning of this chapter (2.1) will include a discussion of general topics in simulation: current issues and future directions. The major portion of the chapter (2.2-2.3) will be a historical overview of the simulation of digital logic circuits for the purpose of design verification. Much of the literature concerns simulators with features other than timing. These include behavioral, fault, functional, and mixed level simulators. Therefore this chapter will discuss simulators, many of which are more complicated than the one proposed for this thesis. The term circuit simulation will be used in a generic sense.

2.1 The Simulation Approach - Issues and Directions

Simulation is the process of representing some portion of the real world in a systematic way, for the purpose of studying the behavior of a system, in terms of evaluation, prediction or optimization of performance. It is often desirable and usually cheaper to be able to predict a system's behavior, to experiment with various designs and to evaluate expected results before the real world system is implemented.

Event list management is one of the important software issues in simulation. An event, in circuit simulation, represents a change in the state of a signal, from Ø to 1 or 1 to Ø. The activity in a circuit can be modelled by a linked list of such events (referred to as an event list), and inserting an event on this list is called event scheduling. Event lists can also be implemented by binary trees or arrays of linear lists. Event list management involves an attempt to optimize speed and "robustness" (speed maintained over a wide class of problems), as well as flexibility and ease of use (Vaucher and Duval 1975).

Another area of concern in simulation is languages. Some simulation specialists prefer widely known general purpose languages with special simulation features while others support the development of new special purpose simulation languages (Henriksen 1984). Both approaches are moving toward developing concurrent features for taking advantage of parallel processing hardware.

Modeling development environments, which include an integrated set of simulation software tools, are evolving as simulation problems become larger and more complex. Modeling development environments provide a setting where models can be developed from early problem definition phases, through the experimental design phase, to final analysis of results (Nance 1983).

2.2 Logic Simulation Overview

Verification of the accuracy of a logic design before the device is actually built can save both time and money. In the early days of integrated circuit design, the approach to simulation was to build simple hardware prototypes for the purpose of testing circuit designs. Changing these prototypes to reflect design changes was very difficult, however, and a new approach was needed.

Software simulators, which are large and complex programs, were developed to model the circuits and to verify the design. These simulators had several advantages, among them ease of modification of the program and relatively low cost. Over time many improvements have been made to these simulators including modeling of different levels of circuit design, faster algorithms, more functional hardware description languages and faster hardware. They represent the primary method of simulation in use today.

Recently, as a response to the growing complexity of VLSI design, some research projects have been returning to a hardware logic simulation scheme, which is substantially different from the previous approaches. These new hardware simulators, sometimes called logic simulation engines, use special purpose CAD hardware and often use distributed and parallel processing. Processors specially designed for simulation reduce the time of the design process, and

hardware simulation takes advantage of the decreasing cost of computer hardware (Howard, Malm, Warren 1983).

Some modern simulation schemes involve a combination of hardware and software. Hardware components, such as microprocessor chips, are incorporated into the simulators to avoid the "time-consuming process of writing software models for complex components." (Mentor Graphics product literature) These hardware components also improve the accuracy of the simulation.

2.2.1 History of Timing Simulation

There are several different ways to model a circuit depending on the information needed. This section will deal with gate level simulation which generates information about timing characteristics in a circuit.

Early timing models used fixed propagation delays, called zero or unit delay models. Seshu's Sequential Analyzer, the first gate level simulator to be published, was of the unit delay type. It was satisfactory for regular logic, but difficult to use for an asynchronous circuit. The network to be simulated was compiled into machine language and executed (Hemming and Hemphill 1975).

An important improvement on this technique was the assignable delay simulator which allowed each device to have one or more propagation times. The Tegas2 developed by Szygenda provided a more accurate model view of system

operation and also handled race and hazard analysis and edge-triggered devices (Hemming and Hemphill 1975).

Toward the goal of more accurate timing, a third state, called undefined, was used to indicate that the logic at a node was not known. This undefined state was particularly useful for simulating the initial state of the circuit. A delay ambiguity problem arose with min and max differences in propagation delays (Bening 1979).

Among several attempts at better timing modeling was the use of a standard deviation (based on min and max propagation times) for delay time at each transition. A further refinement was to use different transition delay times for low to high and high to low transitions. There also came a realization that the loading effects (number of loads on the output) can change the propagation delay times (Bening 1979).

More recently additional states have been added to improve timing simulation of larger systems. Other factors, including physical environment and path delays, are known to affect timing accuracy. The increase in circuit size has necessitated increased capacity of simulators including larger memories, faster hardware and better algorithms (Bening 1979).

A continuing problem is the determination of the size of the input test set which will detect as many as possible

of the design problems without making the simulation program unreasonably long at current computer speeds.

2.2.2 Evolution of Algorithms to Do Timing Simulation

During the mid 1960's a method of simulating logic circuits using an event-scheduling technique was developed. In the intervening years, Ernst Ulrich and other researchers have published a number of papers which illustrate the evolution of this method. A "time-mapping" event scheduling technique in which the Ulrich timing-wheel is used as a future events queue was first proposed (Ulrich 1969). Simultaneously-occurring events were attached to the events queue as side branches.

Time-mapping schemes of Ulrich and others were combined with linear lists to accomplish non-integral event timing (Vaucher and Duval 1975). Ulrich then further developed the algorithm using a converging lists data structure which breaks up a long linear list into short lists and then converges these lists into a single list at a particular point which is always ahead of current time (Ulrich 1976).

Ulrich later summarized four table-lookup techniques which are used to speed up logic simulation by ratios of 15:1 (Ulrich 1980). These techniques include zoom-table, linked list traversal using a traverse-table, and code concatenation for simultaneous traversal of two linked lists. Other advantages to these methods are reductions in

program complexity and programming effort and reductions in volume of program coding.

Another recent publication concerns other techniques which make a gate level logic simulator faster (Miyoshi et.al. 1985). A number of single input logic gates are deleted from the simulation provided they meet certain criteria. Therefore, the number of events to be simulated is reduced and so is computer storage. This method involves an analysis of the gates to determine if they can be deleted.

2.2.3 Projects

A few projects in circuit simulation as well as some outstanding contributors to the field deserve comment. A long-standing project at Bell Labs involves the MOTIS - a Mos Timing Simulator (Chawla, Gummel, Kozak 1975). Many authors have referenced this work. In 1980 a mixed mode simulator (transistor level, logic gate level and functional level) was described (Nham and Bose 1980) and later the 2nd generation Motis Mixed Mode Simulator was developed (Chen et. al. 1984). This project involves a continuing effort in improving the accuracy, flexibility and efficiency in MOS circuit simulation.

One of the early circuit simulation projects, OLLS: On-Line Logical Simulation System, was in operation at MIT in 1971. It was a complete software package with which a

design engineer could interactively design and simulate digital systems using a light pen and programmed function keyboard. I/O was mainly through punched cards. The system allowed flexibility in choice of logic families. It was written in IBM 360 Assembly language and generated off-line plotter output (Howie and Tavan 1971).

F/Logic, an interactive fault and logic simulator was developed at Bell-Northern Research (Wilcox and Rombeek 1976). The simulator is table-driven and uses a timing stack which controls the simulation. Later implementations of F/Logic emphasized functional simulation as an improvement over the basic gate level simulation (Wilcox 1979).

Projects at MIT AI Laboratory use a somewhat different approach. RSIM, a logic level timing simulator uses a linear transistor model to predict the logic state of each node and estimate transition time if it changes state (Terman 1983). The simulation algorithm uses a Lisp-like command language to drive the simulation. Another MIT project called SIMMER also uses a Lisp programming environment and is used as a research tool for understanding the relation of structure to function (Lathrop and Kirk 1985).

2.3 State of the Art in Circuit Simulation

Recent developments in the area of circuit simulation result from requirements for more speed, accuracy. flexibility and ease of use. Integrated circuits have become so large and complex that new approaches are required to simulate them at a reasonable cost. These include software improvements such as partitioning and vectorizing and hardware developments such as concurrent processors and hardware simulators.

2.3.1 Multi-level Simulation

Multi-level simulation incorporates a hierarchical design methodology in which each phase of the design process is modeled in an integrated manner. The simulation tools are developed to support low level early design phases, then fault simulation, and finally the manufacturing test phases (Gonauser, Egger, Frantz 1984). DECSIM, developed at Digital Equipment Corporation, is a multi-level simulation system supporting transistor, gate, vector and behavioral levels (Kearney 1984). It also uses concurrent algorithms.

2.3.2 Design Verification

Formal design verification is becoming an important issue as circuit designs grow more complex. Formal verification techniques are still beyond the scope of

designers but informal validation of design correctness is a useful technique (Whelan 1984).

2.3.3 Languages

New hardware description languages are constantly being developed, many derived from existing languages. The objective in developing a new language is to approximate as closely as possible the circuits which are being described, and therefore make interaction with the simulator easier for the user (Kearney 1984). A contrasting viewpoint is that there are already a large number of hardware description languages in existence which are not widely used and therefore there is no need to develop new languages. The proliferation of different languages leads to a lack of standardization which hinders the development and dissemination of commercial computer-aided design tools (Leiberherr 1984).

Another very important language issue is that of concurrency. Effective parallel processing requires languages which will take full advantage of the parallelism. Consequently new concurrent languages are being developed and concurrent features are being added to conventional languages.

2.3.4 Special Purpose Multiprocessors

PRSIM is a multiprocessor algorithm for logic level timing simulation involving a high degree of parallelism (Arnold and Terman 1985). It uses a partitioning technique which increases the capacity and accuracy of the simulation. It was developed at MIT and is implemented there on the CONCERT multiprocessor. It is in the development stage on the BBN Butterfly supercomputer.

2.3.5 Data Flow Approach

A concurrent timing simulator called CEMU has been implemented on a virtual data flow machine at AT&T Bell Labs Ackland et. al. 1985). A data flow graph is generated from the input circuit description and a time-step subdivision is used for evaluation. The simulator is reported to run much faster than full circuit simulators. The project uses eight M68000 microprocessors and a VAX 11/780.

2.3.6 Logic Simulation Machines

At the Los Gatos Laboratory, the IBM Logic Simulation Machine (LSM) was developed (Howard, Malm, Warren 1983). It is an integral part of the design effort rather than a design verification check on a completed circuit. It consists of a set of 64 processors which operate in parallel to perform the actual simulation of a design. The software

is written in an interactive IBM version of Pascal which takes advantage of the speed and capacity of the machine. The input and output are graphic. The IBM Yorktown Simulation Engine is the second generation of the LSM.

2.3.7 Microprocessor Logic Simulators

Two simulators which are currently available for use on microcomputers are called Logicsim and Logic Designer/Logic Simulator.

Logicsim runs on an IBM-PC as well as other microcomputers with 64K of memory. It will simulate TTL 7400 series ICs and CMOS ICs. It has available a library of pre-defined network macros which the user may incorporate into a circuit design. Logicsim uses a compiled simulation scheme and offers various analysis tools and printer and plotter options for output.

Logic Designer/Logic Simulator runs on an Apple II+ or an IBM-PC. The Logic Designer enables the user to create his circuit design directly on the screen and it incorporates a number of routines to aid in the design process. Logic Designer/Logic Simulator has a limited number of gates and does not use preset, clear and clock signals, nor does it use delays in the circuit simulation. (Rubenstein 1983)

2.4 Summary

Circuits are increasing in size and complexity. This necessitates design tools which are faster and more accurate. Simulation has become an important part of the design process largely because of the cost-reduction in both time and money which can be realized. Simulation has been done both in hardware and software depending on the relative costs and efficiencies of each at the time. Multiprocessing and parallel processing will become more important in circuit simulation because of the speed required and the parallel nature of circuits.

CHAPTER THREE

PROJECT DEVELOPMENT

3.0 Introduction

In the process of developing this timing simulator, several topics were researched, including appropriate simulation algorithms, languages and the selection of a representative set of gates and integrated circuits (referred to as ICs). Issues such as multi-valued logic and various delay models were also studied and certain assumptions were determined to be necessary. This chapter will discuss these investigations and the resulting assumptions.

3.1.1 Algorithms, Language and Elements

Event-directed simulation was chosen because it is more efficient than compiler-driven simulation, as it simulates only the active portions of the circuit. Algorithms to do event-driven simulation were studied and various ideas were used to develop the simulation algorithm for this problem. (Breuer 1975, Ulrich 1980) A table data structure, for storing information about each node, was adapted from Breuer's work. A circuit data structure was developed to represent the connections between elements. A time-ordered

queue, to schedule the future events, is based on Ulrich's time-wheel.

C was chosen as the language in which to code this project for several reasons. Pointers and structures are heavily used and dynamic allocation is used for the manipulation of the future events list. C seemed most appropriate for this task.

A subset of TTL elements was chosen to be included in the simulator's library. Elements were chosen to provide a reasonable subset with which to design simple circuits. Information about each element includes propagation delays and functional data. Two values, usually typical and max, are stored for each low to high and high to low propagation delay. For each element there is a C function which determines the outputs for a given set of inputs.

3.1.2 Types of Delay Models

TIMSIM is designed to simulate the timing of a circuit with respect to propagation delays (low to high and high to low) within the elements. There are several types of delay models which can be used to simulate the timing in a circuit. This section compares the delay characteristics of TIMSIM with other delay models.

A transport delay simulator uses a rather simplistic concept and can be modeled as a unit delay or zero delay

simulator or with variable delays assigned to the elements. It will not function as an accurate timing simulator.

An ambiguity delay simulator takes into account the ranges of each propagation delay, ie., minimum and maximum possible timing for each delay. Built into the simulator then is a region of ambiguity for each signal change. During these times of ambiguity it is not clear if the signal is a Ø or a 1. These regions of ambiguity will propagate through the circuit often leading to an overly pessimistic analysis (d'Abreu 1985). The TIMSIM simulator uses only one value for each propagation delay and does not contain these regions of ambiguity.

TIMSIM uses a value for the delay which is determined by the random generation of a number between the typical value and the maximum value of the delay. The typical and max values are used rather than min and max because often a minimum is not supplied by the chip manufacturer. The random value is used to model the real world situation in which two identical elements would not have exactly the same delay at every signal change.

An inertial delay simulator incorporates the idea that a signal needs to remain unchanged at the input of an element for some specified length of time (particular to that element) before it is "recognized" by the element. If the pulse is long enough it may cause a change in the output signal. This inertial delay concept is not used in TIMSIM.

The decisions of whether to incorporate inertial delay or ambiguity delay were made by the necessity to control the complexity of the simulation development. It was also decided to ignore transition time, the time it takes for a signal to go from Ø to 1, but instead, consider the transition to be instantaneous. These assumptions were made with the realization that certain real world conditions might be missed in the modeling. For example, a spike which might happen in a real circuit may not be generated by this simulator because the ambiguity delays which might cause a spike to occur are not allowed. Furthermore, if spikes do occur in a simulation run, the simulator may propagate them through the circuit because the inertial delays present in a real circuit are not there to filter them out. Therefore, erroneous situations may not be recognized by this simulator because of its limitations.

3.1.3 Multi-valued Logic

The issue of multi-valued logic or multi-valued signals is discussed in several papers on logic simulators (d'Abreu 1985, Szygenda 1975, RNL, Breuer 1975). These sources suggest the addition of a third logic value expressed as 'X' or 'u' to represent unknown, undefined, indeterminate or 'don't care' states. There are several different interpretations for this third logic value and some authors have also suggested the need for several more logic values

to represent falling and rising values (d'Abreu 1985) and potential spikes, hazards or races (Szygenda 1975).

Some papers (RNL, d'Abreu 1985) use X (unknown) to represent the value of the signal between Ø and 1, i.e. between a valid low and a valid high. Others (Lewin 1977) use the unknown state to represent a 'don't care' situation. To others, (Szygenda 1975), X means indeterminate, e.g. the value of the output of a clocked R-S flip flop with inputs of 1 and 1. TIMSIM uses a -1 to represent the indeterminate state. That state is used when a node is either Ø or 1 but it is impossible to determine which it is.

Several sources indicate the importance of using unknown values at the point of "startup" of the simulator. Setting every signal to an unknown value, then resetting prime inputs to a known level (Ø or 1), and propagating the signals through the circuit is a common circuit initialization procedure (RNL, d'Abreu 1985). If, after this initialization, there are still nodes which are unknown, there is a problem. Unfortunately, there are several possible sources of the problem. They include the circuit design, the initialization routine itself or the simulator's incapacity to deal with the complexity of the circuit (Breuer 1975).

TIMSIM uses an initialization sequence of this type to cause the circuit to stabilize before simulation begins. All nodes are set to unknown, then the prime inputs are set

at Ø and a zero delay simulation initialization is run on the circuit. If any nodes remain at the unknown state at this point, these nodes are reported to the user who can then elect to do another initialization run. This additional initialization procedure will set the outputs of certain clocked ICs, which have known values at the inputs but which do not reach a known state in the zero delay simulation initialization.

3.2 Essential Features of the Simulator

A library of twelve standard TTL gates and ICs forms the basis of TIMSIM. These components are of two types - simple gates with single outputs and symmetric inputs (ANDS, ORS and XORS) and the more complex ICs, such as a JK flip flop and a latch. These two groups have different requirements with respect to data structures and algorithms.

The information in the library includes arrays which contain the propagation delays for each TTL component and C functions which compute the value of each output, given the inputs. It is necessary to store four delay times (typical and max for low to high and for high to low transitions) for each distinct combination of input and output. In the case of the simple gates (AND, OR, etc.), the inputs are symmetric and one four element array is sufficient for each. For an IC such as the 7475 latch, the propagation delays are different depending on which input changed and therefore

more delays need to be stored. The algorithm used to simulate the multi-output, non-symmetric input components needs to keep track of which input caused a change to which output and then to retrieve the correct propagation delay for that change.

An IC such as the TTL 7474 d-type flip flop, which uses the former state of its input lines as well as the present state, requires that the simulator store both the previous value and the present value. Components with multiple outputs require keeping track of the order of the output nodes. Two data structures are used to store the connections between the components in the circuit - one which represents the circuit and one which stores information about the nodes.

A significant aspect of the simulation algorithm is the resolution of overlapping delay scheduling. For example, suppose node **n** is scheduled to change and then, because of a change in another input to node **n**, another event is scheduled for node **n** before the previous one is executed. It must be determined what value node **n** should take. This involves a schedule/deschedule algorithm which reads the future event list and schedules, reschedules or cancels an event.

CHAPTER FOUR

IMPLEMENTATION

4.Ø Overview

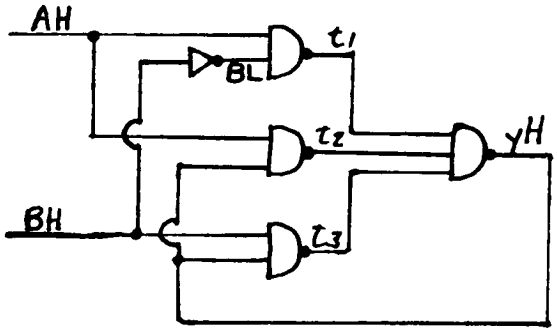
The implementation of the simulator can be described in several ways. Descriptions of the inputs and outputs are found in section 4.1. File descriptions of the source code in section 4.2 indicate the organization of the programs. Structure charts with accompanying descriptions for the modules are located in sections 4.3 and 4.4. Section 4.5 contains detailed descriptions of the data structures and section 4.6 describes the event scheduling and processing procedure.

4.1 Inputs and Outputs

The input to the parser is a circuit description file, e.g. **fname**, made up of a series of statements containing key words and their arguments, enclosed in parentheses. These statements specify TTL elements and their associated nodes.

As is illustrated in Fig. 4.1, all the nodes in the circuit must be listed first. Each element is followed by its output nodes, then its input nodes. The connections in the circuit are thereby established by the named nodes. The symbol ; identifies a comment line.

There are additional features which can be included in the circuit description which will give the user flexibility in creating a circuit. The key words associated with these features are **include, repeat,** and **macro. Include** allows another file which is stored in the user's directory, to be included in the circuit being developed. **Repeat** allows a repetitive sequence to be specified and the parser will expand it as part of the circuit. **Macro** introduces a macro definition which the parser will store and expand at the appropriate place in the circuit.

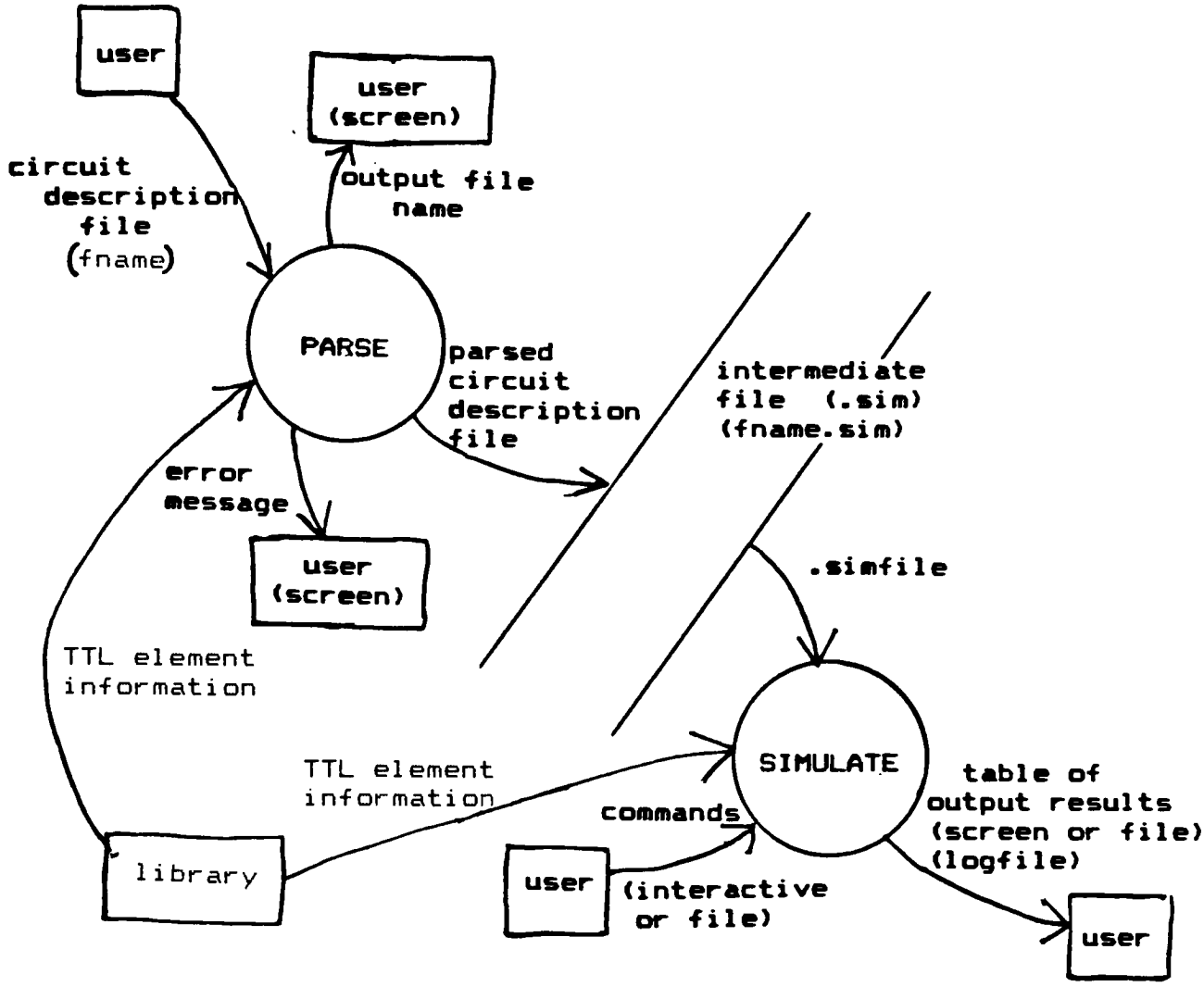



```
;file je2
(node AH BH BL t1 t2 t3 yH)
(inv BL BH)
(nand2 t1 AH BL)
(nand2 t2 AH yH)
(nand2 t3 BH yH)
(nand3 yH t1 t2 t3)
```

Fig. 4.1

Fig. 4.2 is a data flow diagram of TIMSIM. The main output of the parser is a file called **fname.sim** (see Appendix A), which is stored in the user's directory after successful parsing of the input file. This file contains the expanded circuit description in readable form. The user may verify the circuit connections with this file.

If the circuit description file was parsed correctly a message appears on the screen, giving the name of the parsed file. If the input file was not syntactically correct an error message is output and the program halts.



Data Flow Diagram

Fig. 4.2

One input to the simulator is the **fname.sim** file, output from the parser. The simulator reads this file and loads the information into the internal table and circuit data structures.

The other input to the simulator is a series of commands that define the parameters of the simulation.

These commands can be run from a batch command file or entered interactively. If interactive, the user will be prompted for selections. The command file which needs to be included as a command line option follows a precise format for choosing the simulation options. The options available to the user are the length of a simulation step, the format of the output, and the selection of nodes for the output report. These options remain constant through an entire simulation.

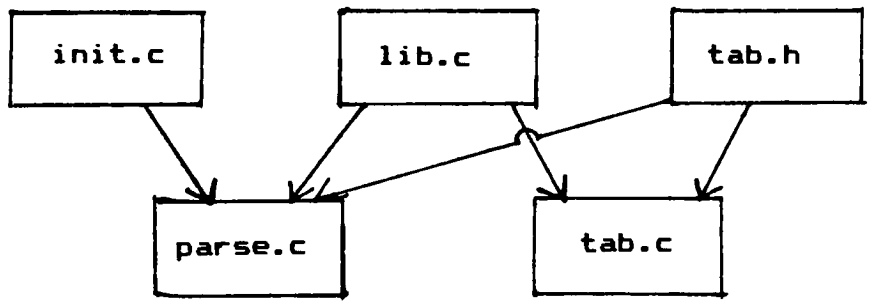
There are three possible output formats. One is in the form of a table which reports all activity (changes in node values) during the simulation step and the times of those changes. Another output format is graphical with node values reported at regular intervals. A third output format reports the value of the nodes at the end of the simulation step as well as the time of the last state change.

Default for the output report is to the screen. There is also a log file command line option. Exercising this option causes the results of each simulation step (in whichever format was chosen) to be placed in a file in the user's directory. The file is called **logfile.**

To begin a simulation step, the user sets the value of any or all nodes (not only primary inputs) and begins the simulation. The simulator determines whether there are any changes to the values of output nodes affected

by the changes in input values. If a potential change is recognized, the propagation delay (low to high or high to low) for the particular element is determined and the event, the change in value of the output node, is put on the future events list. This process continues for the period of time chosen as the simulation step length.

4.2 Files



File Structure

Figure 4.3

The content of each of the files is described below.

lib.c - The lib.c file contains the library of TTL elements which are available to the simulator. The library includes a table of propagation delays for each element, with each delay being represented by a typical and a max delay time for low to high and for high to low. The library also contains subroutines which express the functionality of each element.

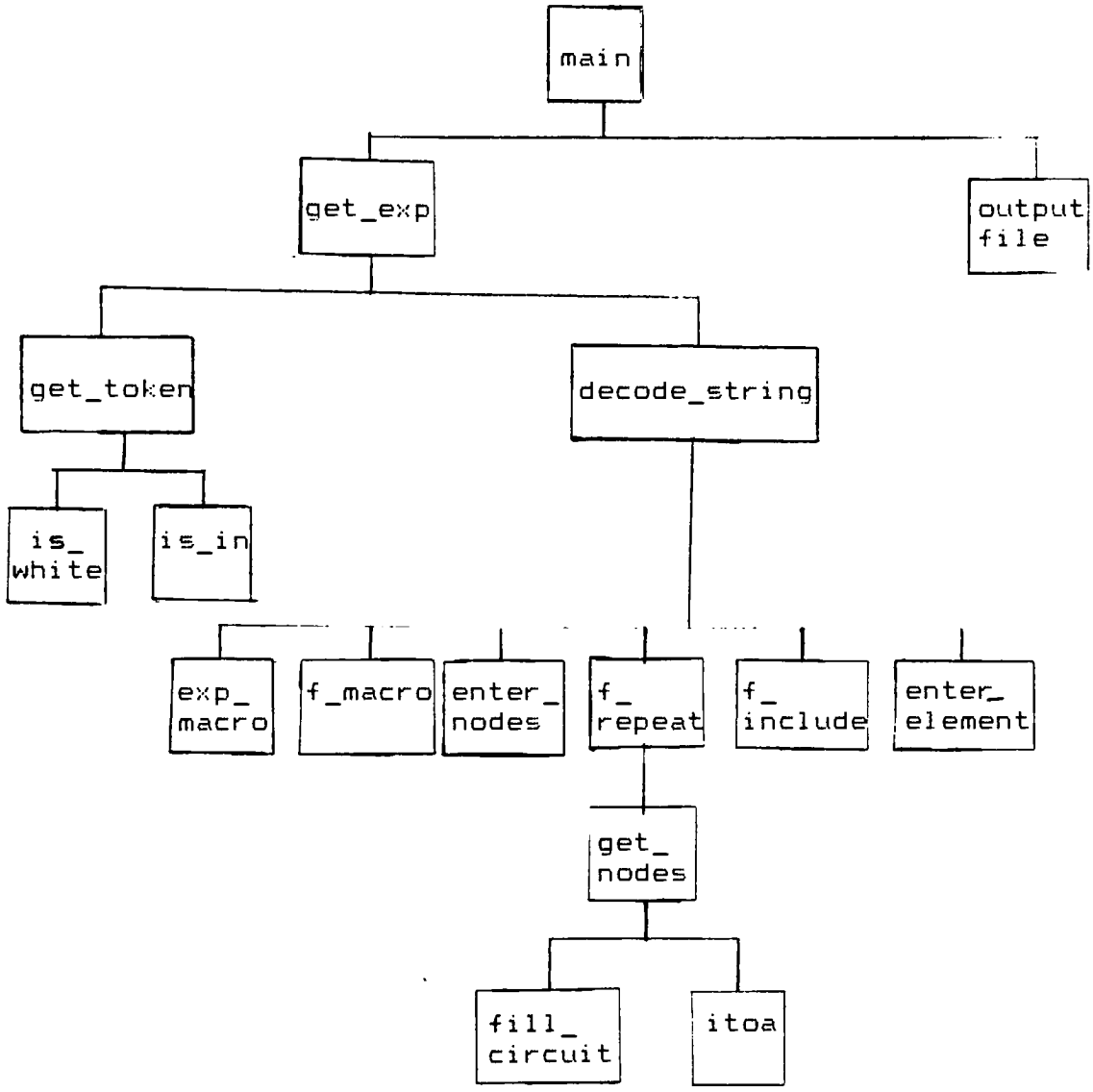
tab.h - The tab.h file contains all the supporting data structures for the simulator. These include the main table of nodes which drives the simulation, the circuit structure, the time-wheel and the record array. The table is an array of structures, each representing a node and its related information - value, previous value, scheduled time to change and locations in the circuit. The circuit is an array of structures, each representing an element in the circuit. It stores the input nodes and output nodes for each element. The time-wheel is a circular array of pointers to linked lists which are used to store the pending events in the simulation. The record array keeps track of all the events so they can be reported.

parse.c - The parse.c file contains the executable code for the parser with subroutines for decoding each of the key words in the circuit description file. It reads the circuit description file, expands the macro and repeat constructs and outputs a file containing the expanded circuit.

init.c - The init.c file contains the error messages for the parser and also the mapping of elements and key words into integers.

tab.c - The tab.c file contains the executable code for the simulator. It does an initialization routine, reads in commands from the user, then repeatedly reads the future events list, executes the changes, then schedules more changes.

4.3 Modules in the Parser



Structure Chart of the Parser

Figure 4.4

The following are brief descriptions of the major subroutines in the parser.

main - opens and closes the input file; names the output file

get_exp - drives the parser

get_token - reads the next character; determines its type

decode_string - case statement for the key words in input

exp_macro - expands macros and puts them in the circuit

f_macro - stores a macro description for later processing

enter_nodes - loads nodes into the table

f_repeat - decodes repeat loops in the input file

get_nodes - deciphers the node list in a repeat loop

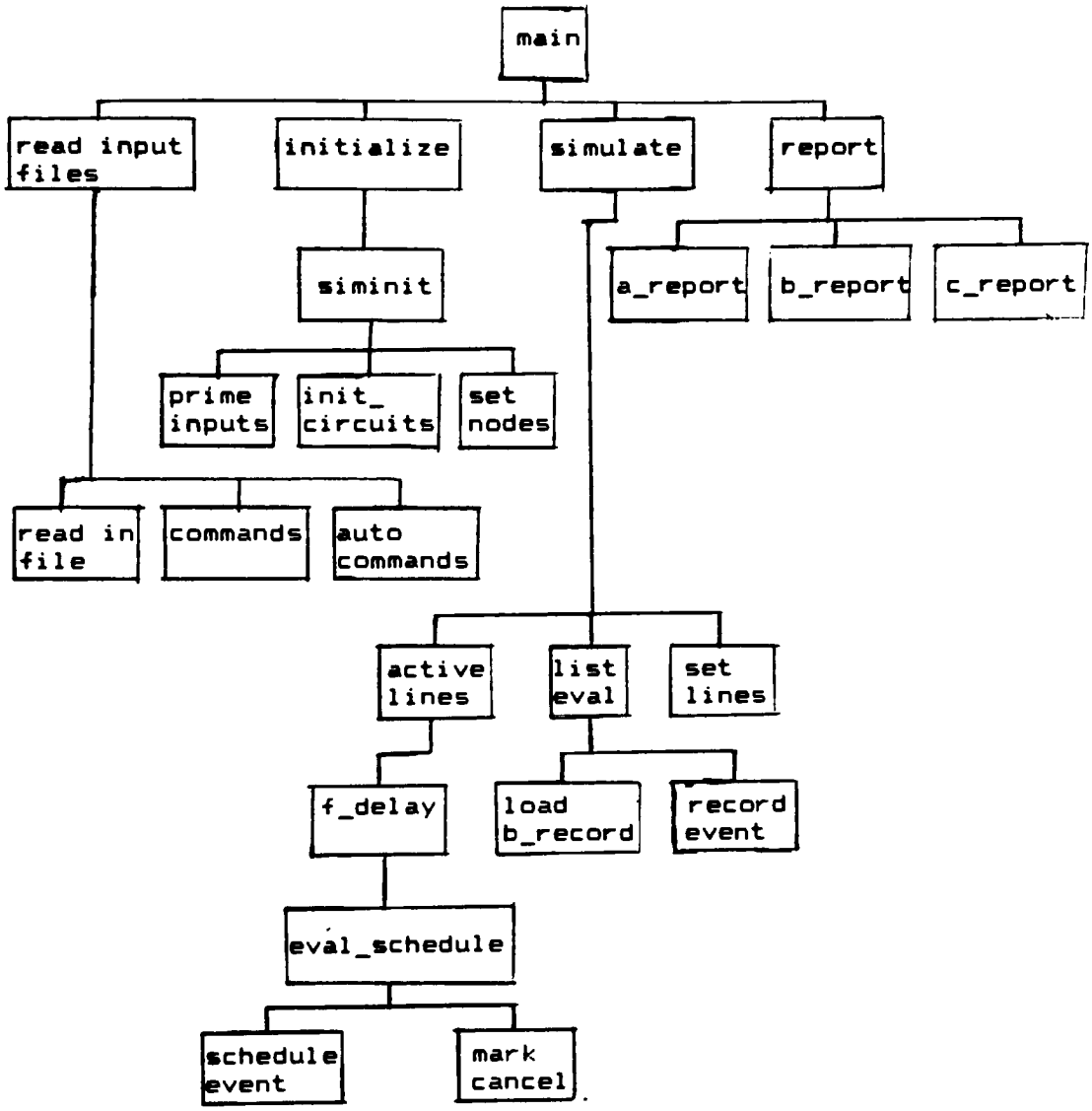
fill_circuit - loads the circuit data structure with indexed nodes

f_include - handles any include lines in the input file

enter_element - puts an element with its nodes into the circuit data structure

output_file - writes the output file.sim

4.4 Modules in the Simulator



Structure Chart of the Simulator

Figure 4.5

The following are brief descriptions of the major subroutines in the simulator.

main - opens files; initializes; reads input; drives simulation

set_lines - resets nodes after each simulation step

autocommands - interprets commands from a command line file

commands - reads and interprets the interactive commands

list_eval - traverses the time-wheel and processes events

schedule_event - puts an event on the future events list

record_event - maintains the record of events

loadb_record - keeps track of events for one output format

a_report - prints out output format **a**

b_report - prints out output format **b**

c_report - prints out one output format **c**

active_lines - traces the signal from present node to the fanouts

evalschedule - determines schedule or deschedule of events

mark_cancel - marks an event for descheduling

prime_inputs - sets the primary inputs in the initialization

init_circuit - initializes the circuit

setnodes - interprets the user commands for the initialization of unknown nodes

sim_init - coordinates the initialization schemes

readinfile - reads the input file

4.5 Important Data Structures

4.5.1 Library

The library contains an array of structures, each structure representing a TTL component, a gate or an IC, henceforth referred to as an element, in the simulator. See Fig. 4.6.

element name
number of inputs
number of outputs
timing array indicator
symmetric or non-symmetric inputs
pointer to function

Library Data Structure

Fig. 4.6

The library also contains arrays of propagation delays. Different requirements for the different TTL elements necessitated several arrays for handling delay times for various combinations of inputs and outputs. The basic information stored for each combination of inputs and outputs is shown in Fig. 4.7.

low to high		high to low	
typical	maximum	typical	maximum

Timing Delays

Fig.4.7

Each element's structure contains a pointer to the C function which determines the functionality of the element, i.e., the output value determined by the input values.

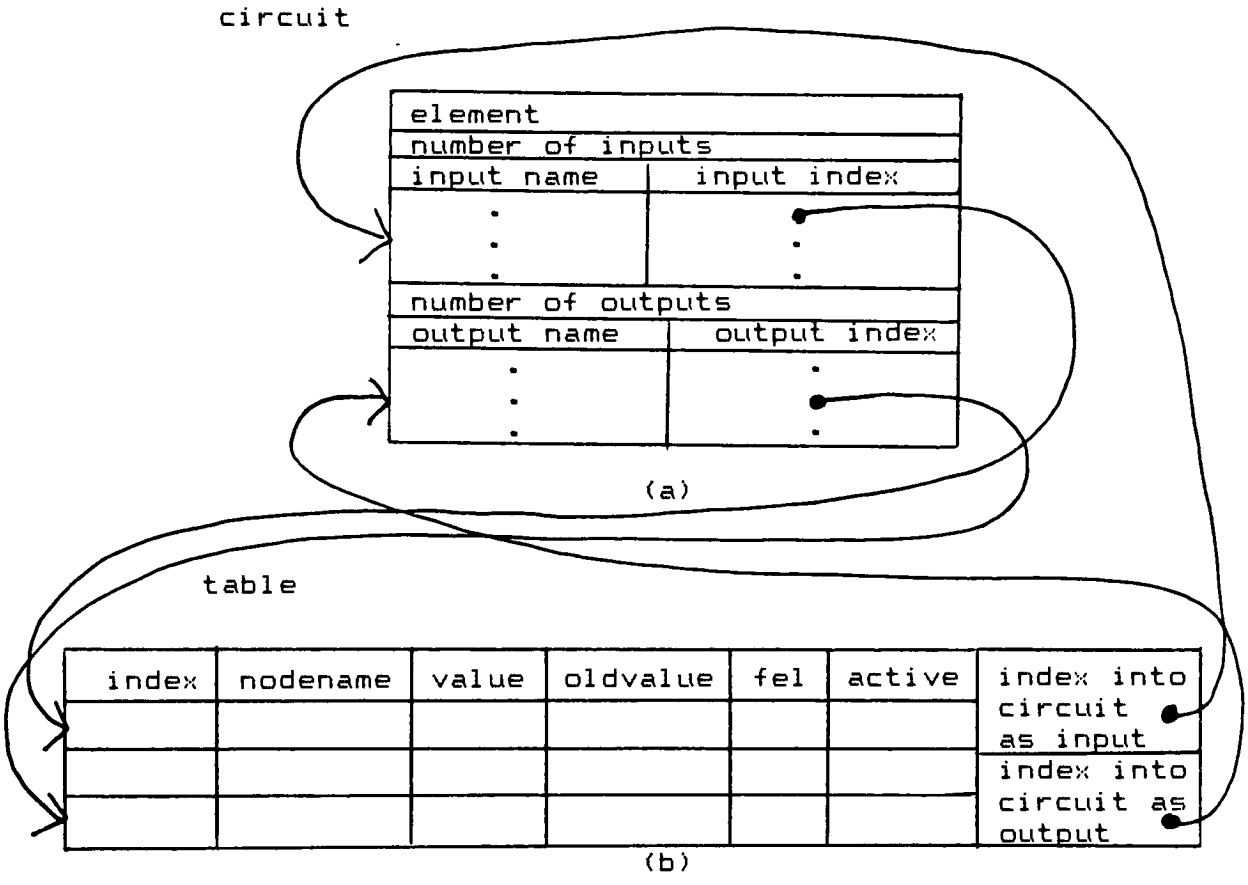
4.5.2 Circuit and Table

The circuit and table data structures each have several fields which store the information about the circuit to be simulated. These fields are loaded with data when the simulator reads the input .sim file. The circuit data structure values remain fixed during the simulation. Some of the fields in the table vary during a simulation run.

The circuit data structure is an array of structures, each representing an element in the input circuit. Each structure contains the fields shown in Fig. 4.8a. The circuit data structure contains table indices of the nodes which are inputs and outputs in the simulated circuit.

The table data structure is illustrated in Fig. 4.8b. The table is an array of structures, each representing a node in the circuit. Fields of each node include the name, the current value, the previous value (old value), and the

time at which a node is scheduled to change (fel). The active field indicates that the node value has been set by the user. Two lists of pointers to the node occurrences in the circuit are also included.



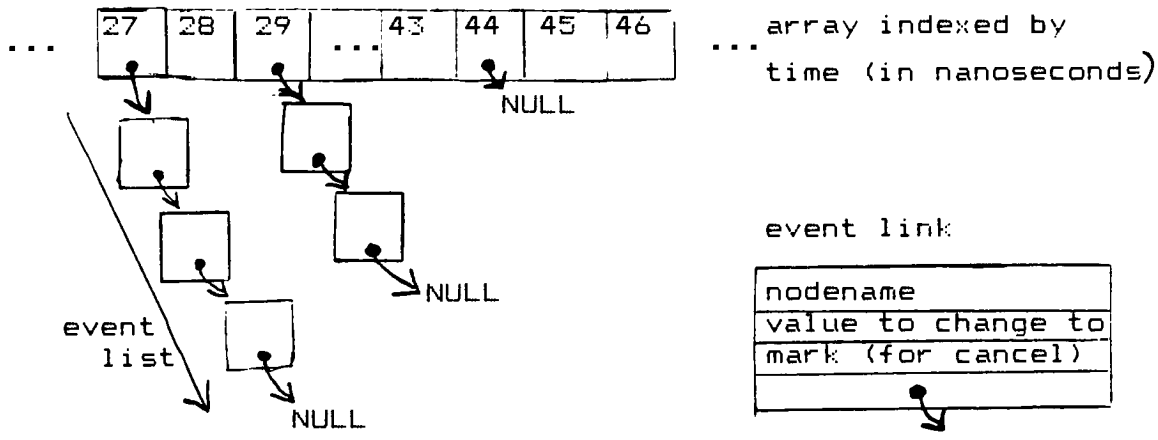
Circuit and Table Data Structures

Fig. 4.8

4.5.3 Event Scheduling Mechanism

The time-wheel is a circular array of pointers to linked lists. Each link in a list represents an event which has been scheduled for the time indicated by the index of the pointer.

The links are dynamically allocated when an event is scheduled and freed when the list is empty. The indices of the array represent time in nanoseconds. When the simulation reaches the end of the array, it wraps around and reuses the array of pointers.



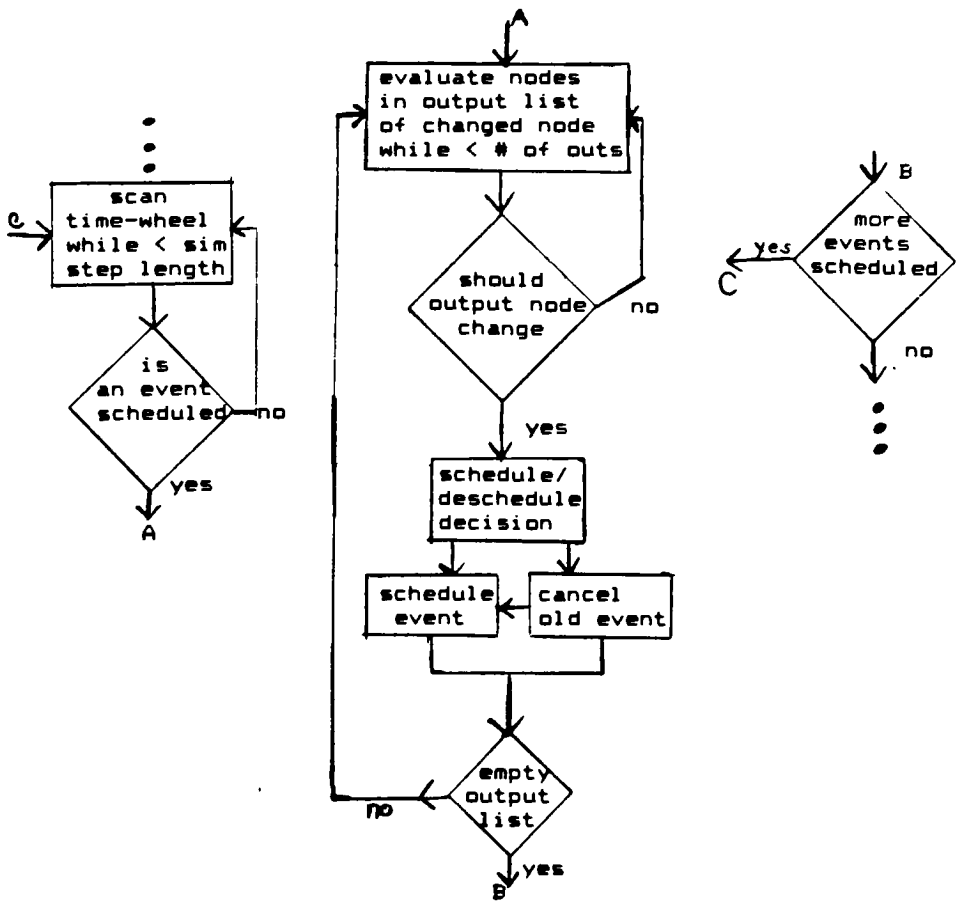
Time-wheel Data Structure

Fig. 4.9

4.6 Event Scheduling Process

Fig. 4.10 is a flow chart of the steps in processing an event and then scheduling resultant events. When the algorithm finds an event scheduled on an event list, it must

process the event. First it changes the old value of the node to equal the current value and then changes the current value as the event link indicates. The algorithm then determines all the nodes whose values could be affected by this current change. These nodes are the fanout list of n. The algorithm investigates each of these nodes to determine if its value should be scheduled to change. Each new event is then scheduled by allocating a link on the linked list which emanates from the array at the appropriate time.



Event Scheduling Flow Chart

Fig. 4.10

CHAPTER FIVE

TESTING

5.0 Testing Strategy

Each part of TIMSIM was extensively tested during the development phase. Input appropriate to the level of the program was used at each stage of development. The objective of the testing strategy was to cause the program to fail. Representative circuits were developed to test various features of the parser and simulator. Appendix B contains sample runs of the simulator with output from several different test circuits.

5.1 Testing the Parser

The parser depends on precise, specific and accurate circuit description input. The input format is specified in the User's Manual. Deviations from these input formats will cause an error message to be printed and the program to be aborted. (Appendix B of the User's Manual has a complete list of the error messages.)

The parser was developed and tested module by module. Testing was done for each expected key word and its associated input. For example, when the word **node** is recognized, a list of the nodes in the circuit is expected.

The code then processes this list of nodes, storing each in a table data structure.

Each feature of the parser was tested, as the code was written, with several correct circuit description files containing that feature. Valid combinations of the key words were tested also. The best technique for checking the correct operation of the parser was to compare the output .sim file with the original circuit. The connections between nodes can be traced in the .sim file and can verify that the parser is correctly interpreting the input circuit.

As the final phase of testing the parser code, the technique of error guessing was used to develop error traps. When the parser had been developed to the point where it would parse correctly valid circuit description files, an attempt was made to determine the common types of user input errors which could be expected in the circuit description file. These anticipated errors were placed into circuit description files which were used to develop error traps and explicit error messages. When an error is encountered, the program is aborted, and the error message is printed on the screen. Each error message contains a line number indicating the approximate location of the error in the input file.

Examples of errors in the user input, which can be anticipated and tested for, are violations of the input format such as incorrect use of ( )'s, misspelled key words,

missing lines and incorrect use of punctuation. There is no attempt to interpret the user's intention or to "second guess" the input.

5.2 Testing the Simulator

Each simulator feature was tested during its development with a variety of input circuits. These circuits were designed to specifically check that particular feature. Various combinations of circuits and features were also tested.

The simulator reads the input file and puts values into the internal table and circuit data structures. During development, the values in these data structures were printed out and verified to be certain that the simulator was processing the right circuit.

A group of test circuits containing certain key features were used to test new modules and code changes as they developed. Different circuits were used which incorporated the new features. The timing delays of the test circuits were calculated by hand so that the simulation results could be verified. Test circuits were chosen to incorporate various combinations of elements, feedback or potential hazard situations.

Each new element, gate or IC, was tested as it was added to the library. In the case of the more complex ICs, three levels of circuit complexity were tested. These

included the IC as a circuit itself, a small circuit containing a few elements, then a larger circuit with extra features.

Features of the simulator which were extensively tested were various combinations of choices made by the user (output formats, length of simulation steps), the schedule/deschedule algorithm and the initialization procedures. If the parser fails and writes a .sim file which is incorrect (with improper connections between nodes), the simulator will be unable to run the simulation and will produce a segmentation fault core dump.

5.3 Test Comparisons with Other Simulators

Appendix B contains sample runs of the simulator. The test circuits were run on the RNL Simulator as well as TIMSIM with comparable results. The results were also hand calculated using typical delays to establish a frame of reference for the results. TIMSIM uses randomized delay times between typical and max values and RNL uses fixed values. The comparisons between the two are therefore only approximate, but consistently similar. Nodes change in the same manner at similar intervals.

CHAPTER SIX

CONCLUSION

6.1 Summary of the Project

The TIMSIM simulator is a reasonably accurate, easy-to-use gate level timing simulator for simple digital logic circuits. The logic circuits can be constructed from a specific subset of 12 standard TTL gates and ICs. The circuits which can be simulated by TIMSIM are limited in size to 30 elements. In a two-part process, TIMSIM first accepts a circuit description file and outputs a parsed version of this file. In the second part, it performs a series of simulation steps on that circuit. At the beginning of each simulation step the user sets the value of nodes. The program then reports the activity (change of node value, at what time) in the circuit during the simulation step. The user determines the length of the simulation step and specifies which nodes are to be reported.

TIMSIM allows the user flexibility in describing the circuit with the use of macros and repeat loops. The user can choose from three output formats providing different ways of viewing the results.

The simulator would be useful as one component in a "tool kit" of design aids for digital logic circuits. Its

use is limited to those circuits containing the standard TTL elements contained in its library. In an academic setting, where the objective is a general understanding of the effect of propagation delays on the design of a digital logic circuit, TIMSIM could be a useful learning aid.

6.2 Problems Encountered and Resolved

Several problems were encountered in the process of developing the simulator. Three levels of logic necessitated an extensive analysis of the functionality of some of the ICs, particularly those with a memory component. Additional initialization procedures were necessary to enable the simulator to handle certain types of ICs. An algorithm to analyze the event-scheduling function was developed to address some of the limitations of the simulator with respect to setup and hold times.

6.2.1 Multi-valued Logic

The most challenging analysis aspect of this project has been the issue of multi-valued logic. It was determined at the start to use high (1), low (Ø), and a third value (u for unknown). There are two purposes for this unknown state. Prior to powerup, the values of all nodes in the circuit are unknown (u). During the initialization phase of the simulation, the prime inputs are set and a Ø-delay

simulation is run so that the output nodes of each element assume values consistent with their inputs. The second purpose of u is to model that situation in which, during a simulation run, the inputs of an IC assume values such that the IC's outputs are unknown. An example is: if the inputs $\overline{\text{preset}}$ and $\overline{\text{clear}}$ of a 7474 d-type flip-flop are both low (trying to preset and clear at the same time) the outputs are unknown. The u is **not** used as an indeterminate state between Ø and 1 or as a "don't care" state.

Published truth tables for gates and ICs use only two values, high and low. Ternary truth tables (3-valued logic) are considerably more complicated. As an example, consider a 2-input nand gate. There are 9 possible input combinations (as compared to 4 for 2-level logic). These 9 quickly reduce to 6 because of the symmetric nature of the inputs and to 4 because of the presence of Ø in the inputs. See Fig. 6.1.

A	B	Z
Ø	Ø	1 ←
Ø	1	1 ←
~~1~~	~~Ø~~	~~1~~
1	1	Ø
Ø	u	1 ←
~~u~~	~~Ø~~	~~1~~
1	u	u
~~u~~	~~1~~	~~u~~
u	u	u

Because of symmetry
Ø 1 is equivalent to 1 Ø
Ø u is equivalent to u Ø
1 u is equivalent to u 1

If either input is a Ø, the output is 1, regardless of the other input.

Fig. 6.1

On the other hand, the complexity of 3-valued logic becomes evident in developing the truth table for the 7474

d-type flip-flop. In this chip there are four non-symmetric inputs, $\overline{\text{preset}}$, $\overline{\text{clear}}$, clock and data. For 3-valued logic this involves 3^4 or 81 possible input combinations. However, in this case, the value of the inputs before the change (memory component) is also a consideration in determining the outputs. These memory states provide another 8Ø possible input combinations for each of the original 81 inputs, or a total of $64ØØ^+$ input combinations. There are some dominance relationships here which will reduce the complexity considerably. For example, changes in $\overline{\text{preset}}$ or $\overline{\text{clear}}$ affect the outputs immediately regardless of the value of clock and data. In addition a change in clock from Ø to 1 is the only significant clock change, but when this change occurs, the states of $\overline{\text{preset}}$ and $\overline{\text{clear}}$ have to be considered in order to determine the output values. A simplifying feature is that a change in data has no effect on the outputs, which reduces the total input possibilities. See Appendix C for an analysis of the ternary truth table for the 7474 d-type flip-flop.

As another example, the 74151, a data selector/ multiplexer, has a total of 12 non-symmetric inputs which means 3^{12} possible input combinations to consider. It is clear that this number can be reduced substantially. To do this requires a careful, deliberate analysis of the logic of the chip in order to discover the combinations of inputs

which will produce particular outputs. Certain patterns tend to emerge through this process of analysis.

6.2.2 Initialization or Simulation Startup

The purpose of a startup procedure is to allow the circuit to attain a known state at the beginning of the simulation so that the future changes can be clearly understood. The initialization strategy of setting all nodes to u, setting prime inputs to Ø and running a Ø-delay simulation to initialize all nodes in the circuit works well for all of the circuits tested except those which have memory elements. These ICs, which use the former values of the inputs or outputs to determine the new values of the outputs, will continue to contain nodes which have an unknown value after the standard circuit initialization.

A different strategy is necessary in those cases. The problem has been addressed by giving the user the opportunity to intervene and to assign a value to those nodes which are still unknown after the first initialization. In some situations it is possible, however, that an unknown value is an appropriate starting point for the simulation. These unknowns will, in many cases, continue to propagate through the circuit. Encountering this situation may provide a valuable learning experience for a novice circuit designer.

6.2.3 Event Scheduling Analysis

Sometimes a node may be scheduled to change more than once in a short period of time. That is, after an event for node **n** is inserted in the future events queue and before that change can be executed, another event occurs which may cause a change in node **n**. It becomes necessary to investigate any previously scheduled, but not yet executed changes in node **n** each time that node **n** is in the fanout list of a node which has changed. The potential scheduling of an event for a given node depends on what events have already been scheduled for that node.

The time of a scheduled event for a node is stored with the node. At the time of a potential event-scheduling for a node, a check is made to determine if the node is already scheduled to change. If it is, the type of change (low to high or high to low) is compared with the present proposed change. The times of the two future changes are also compared. Depending on the outcomes of these comparisons, the formerly scheduled event may be cancelled and/or the new event may be scheduled or no new action may be taken at all.

This algorithm attempts to model as closely as possible the changes in node values which would occur in the operation of a real circuit. Any node value change for however short a time is carried out. This is an aid to hazard detection in asynchronous circuits. It should be noted that spikes (very short duration changes in a node

value) may occur in the simulator which might be filtered out by the effects of setup and hold time in a real circuit.

6.3 Further Work

There are several additions or extensions which might be made to this project to make it a more accurate and versatile simulator.

Obviously, more TTL elements could be added to the library. As the simulator is written now, elements which are not included as primitives can be implemented (as macros) using the elements in the present library. However, the timing delays using macros in that way would not be the same as the propagation times through an element implemented as a primitive.

A possible extension would be to develop libraries of logic families other than standard TTL, such as low power Schottky or advanced Schottky TTL. Some of these logic families would require smaller time increments than the one nanosecond time unit currently being used. This is easily changed. With this extension, a command line option could be used to access the desired library. It should be noted that TTL is declining in usage for SSI applications and other logic families such as high performance Cmos are increasing.

Another extension would be to simulate inertial delay (setup and hold times) or ambiguous delay (using both min

and max delays). These would allow the possibility of experimenting with pessimistic timing or worst case analysis.

Another extension would be to add more features to the parser to allow more complicated circuit description input. Possibilities would be to allow a macro call inside a repeat loop.

The simulator could be revised to allow more interaction during a simulation run. The user might be allowed to specify an input vector to change a signal during a simulation step.

Other input and output features are also possible. A plot of the output might be a more meaningful way to understand the results of a simulation run.

APPENDIX A

USER'S MANUAL

TIMSIM USER'S MANUAL

System Name

TIMSIM

System Description

TIMSIM is a set of two programs which will enable the user to simulate the behavior of simple digital logic circuits which use a subset of standard TTL elements. It can be used to aid in the design of small combinational and sequential circuits by modeling the propagation delays of the elements and revealing hazard conditions in the design.

TIMSIM is intended to be used as an extension to the set of circuit simulation tools, one of which is called RNL, that run on Unix. TIMSIM uses a similar input scheme and incorporates some of the features of RNL but the command language and the output are different.

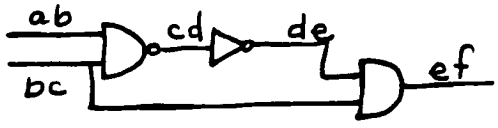
There are two programs in the TIMSIM simulator - CDF and ATS. The first program, CDF, requires as input a circuit description file. It outputs an intermediate file which is then used as input to the second program, ATS.

Part I - CDF

Creating the Circuit Description File

To simulate a circuit, it is necessary to first create an input file that describes a circuit. This file contains a series of statements specifying gates and other TTL **elements** connected by named **nodes.**

For example, consider the following circuit and its corresponding circuit description (in file **cdfile**):




```
;file cdfile
(node ab bc cd de ef)
(nand2 cd ab bc)
(inv de cd)
(and2 ef bc de)
```

(Lines which begin with ; are comment lines and are not read as part of the data.) Each statement has the form:

(command-name argument1 argument2 ... argumentn)

Each **element** is designated by listing the element name and its outputs and inputs as follows:

(nand2 cd b bc) ** Notice outputs are listed first.

(The available elements and their names are listed in Appendix 1.) Every circuit description begins with a list of all nodes identified in the circuit, as in line 2 of the example. Names of nodes should start with an alphabetic character and have a maximum length of 7.

To Run CDF

cdf cdfile

The response, after successful completion, is

Output file is cdfile.sim

The format of this file is explained in Appendix 4. If the input circuit description file is not syntactically correct, an error message will appear on the screen and the run will halt. Error messages are explained in Appendix 2.

Additional Features

Additional input features of CDF give the user flexibility in creating the circuit description input file for CDF. These allow the user to **include** another file in the circuit description, to write a **repeat** loop in the circuit description, and to define a **macro** for use in describing the circuit.

- (**include** filename)

This keyword allows another file using the standard format to be included in this file. The simulator will expand this construction to make the "included" circuit a part of the main circuit. The node which connects the "included" file to the main file must be declared in both files. "Includes" may be nested, up to three levels.

- (**repeat** loop-index start-value end-value
  (body of repeat loop)
)

example:

```
(repeat i 1 6
(inv in.(-i 1) in)
)
```

This facility allows the user to specify a repetitive situation, such as stringing together several inverters, with a repeat loop. The notation (-i 1) means (i - 1), i.e., the node before i.

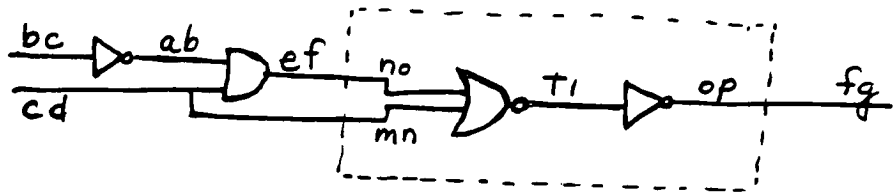
- A macro definition is written as:

```
(macro name (param1 ... paramn)
(local n1 n2 ...)
   body of the macro)
```

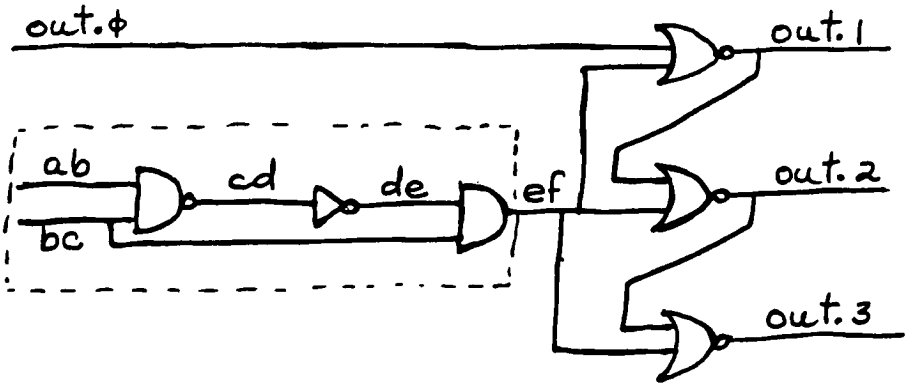
Names in the parameter list must not be the same as global nodes and local nodes also need to have unique names. See example next page. Macros will be expanded and included in the circuit at the indicated position.

Example test files:

```
;test file macf
;contains a macro def
(macro ore (no mn op)
(local t1)
(nor2 t1 no mn)
(inv op t1)
)
(node ab bc cd ef fg)
(inv ab bc)
(and2 ef ab cd)
(ore ef cd fg)
```




```
;testfile tf6
;combines include and repeat
(include cdfile)
(node ef out)
(repeat i 0 2
(nor2 out.(+i 1) out.i ef)
)
```



Part II - ATS

Input

There are two sets of input to ATS. The first input is the .sim file which was the output of CDF. The other input is a series of commands which will control the simulation.

This series of commands may be implemented either by responding interactively to the menu on the screen or by creating a file called **command** which contains the simulation directives. This command file is used only at the beginning of the simulation.

Overview of the simulation procedure

The simulation of the circuit activity is done via a series of simulation steps. At the beginning of each step any of the nodes may be set to values (Ø or 1) by the user. Typing **s** starts the simulator, which then runs for some simulated period of time (in nanoseconds) (again determined by the user). At the end of this time (called a simulation step) a report of the activity in the circuit is output (either to the screen or to the screen and to a log file). The user may then run another simulation step by responding to a prompt and resetting some nodes, then by typing **s** again. This can be done interactively, indefinitely, as the user chooses, in order to examine the behavior of the circuit for various combinations of inputs. The original choices for output format, step length and nodes reported, as set up in the first step, hold through the entire simulation. Typing **q** will cause the simulation session to terminate.

Interactive Session

The interactive method requires the user to respond to the choices on the screen. To run the simulation interactively, type

ats cdfile.sim

The following will appear on the screen:

To set the length of a simulation step, choose:
- a. 4Ø nanoseconds
- b. 1ØØ nanoseconds
- c. user selects a step length
- d. automatic halt (no more events scheduled)

Enter your choice - a, b, c, or d

To select the output format, choose:
- a. Report 1 - the time of each node state change
- b. Report 2 - graph of node values with respect to time
- c. Report 3 - state of all nodes at the end of a step

Enter your choice - a, b, or c

The following are the nodes in the circuit...

Type the names of the nodes to be reported, followed by a **$** or type **all** with no **$**.
Set node by typing the name of the node followed by a **1** or a **Ø**.
Type **s** at beginning of line to end input and start simulation.

With certain combinations of gates and flip-flops, the following message may appear:

The circuit contains the following nodes with unknown values:
xx
xx
Check your circuit to be sure memory elements are correctly set or cleared. Set the node by typing the name of the node followed by **1** or **Ø**.

In this case, the user may respond as indicated by setting the nodes which are unknown or by setting other nodes to achieve a particular state in the circuit. If unknown node values still remain, the following message may then appear:

The circuit still contains node(s) with unknown values.
Do you wish to continue with the simulation? y or n

Batch command file

A command file may be used for the first simulation step by putting the following into a file named **command**. (See interactive session for description of choices.) Only what appears in bold type is included in the file. Use one line for each command.

choice of step length **a** or **b** or **c** or **d**
choice of format **a** or **b** or **c**
which nodes to report eg. **ab ef $** (followed by $)
or **all** set nodes

simulate **s**

The following is an example command file for cdfile.sim

```
a                /* (output format a) */
a                /* (sim step length 40 ns) */
ab bc ef  $      /* (report on these nodes) */
s                /* (begin) */
```

(No comments or extraneous characters belong in this file.)

When using this file to run the simulation, type

ats cdfile.sim command

One other option is possible. The user may record the results of a simulation with a log file. The choice is made by typing **log** on the command line. The file created in the user's directory is called logfile.

In this case, to run ATS, type -

ats cdfile.sim command log (for batch)
ats cdfile.sim log (for interactive)

Output of ATS

The main output of the ATS program is the results of the simulation run. These results are reported, at the end of each simulation step, in one of three formats determined by the user.

Format **a** reports on each change of value of the node and the time (in nanoseconds) at which the change occurred.

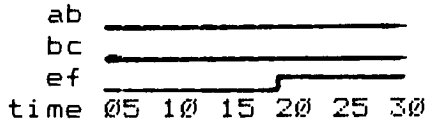
	TIME	NODE	OLD_VALUE	NEW_VALUE
*	Ø	ab	Ø	1
*	Ø	bc	Ø	1
	Ø	cd	Ø	1
	Ø	de	Ø	Ø
	Ø	ef	Ø	Ø
	7	cd	1	Ø
	24	de	Ø	1

after 4Ø nanoseconds

The listing of values of nodes at time Ø reflects the following initial conditions:

1. A Ø-delay simulation has been done with nodes set to unknown and prime inputs set to Ø to initialize the circuit.
2. The values of nodes set by the user are recorded (preceded by an *).

Format **b** is a simple graphical output which samples the simulation at regular intervals and reports the value of all nodes.



Format **c** gives the state of the selected nodes in the circuit at the end of the simulation step.

NODE	STATE	TIME_OF_LAST_CHANGE
ab	1	last change in ab at Ø
bc	1	last change in bc at Ø
cd	Ø	last change in cd at 7
de	1	last change in de at 24

after 4Ø nanoseconds.

A signal which is either Ø or 1, but it cannot be determined which, is indicated by a -1 in formats **a** and c. In format **b**, this unknown state is denoted by an X.

Error_Conditions_and_Limitations

1. A complete listing of the error messages is included in Appendix 2.
2. Syntax requirements and constraints on the system are also found in Appendix 2.
3. Use of values other than 1 or Ø by the user will give unpredictable results.
4. The simulator is designed for relatively small circuits, under 1ØØ elements. If the size of the circuit approaches this maximum, other parameters may be exceeded, causing unpredictable results.
5. It is important to note that TIMSIM is limited to a subset of standard TTL elements and it models only the propagation delays of these elements.
6. To avoid possible confusion with the single letter commands **s** and **q** in the command language, it is not advisable to name a node s or q.

Summary

The following steps summarize this document:

1. Create a circuit description e.g. cdfile.
2. Type **cdf cdfile** - this will create a file cdfile.sim.
3. The next step may be either interactive or batch mode, using a command file.
4. Type **ats cdfile.sim (command) (log)**
5. Set nodes - e.g. x 1
   y 1
6. Type **s** to begin simulation step.
7. After each simulation step, nodes may be reset, responding to a prompt.
8. Type **q** at any point when user input is expected to end the simulation.

APPENDIX 1

Standard TTL Elements Implemented in TIMSIM

TTL#	Type	TIMSIM Library Name
7400	2-input nand gate	nand2
7402	2-input nor gate	nor2
7404	hex inverter	inv
7408	2-input and gate1	and2
7410	3-input nand gate	nand3
7420	4-input nand gate	nand4
7427	3-input nor gate	nor3
7432	2-input or gate	or2
7442	4-10 BCD to decimal decoder/multiplexer	dec4_10
7474	d-type flip-flop	dff
7475	2-bit quad latch	latch
7486	2-input exclusive or gate	xor

USER'S MANUAL
APPENDIX 2

Part I - Error Messages

The program keeps track of the line numbers in the circuit description. Each error message will contain a line number. These line numbers should be considered to be approximate - i.e., the error is in the vicinity of that line number.

Line #Ø is used to report errors on the command line such as not giving a valid filename as the input file to the simulator, eg. typing **cdf simmple** instead of **cdf simple.**

ERROR MESSAGES

"wrong number of file names in command line"
"syntax error in circuit input"
"no circuit to simulate"
"no such command-name"
"could not open the file"
"too long name for node"
"incorrect node expression"
"left paren missing inside repeat loop"
"circuit element unknown to simulator"
"error in parens in circuit description file"
"no such node found"
"expected `(' "

Part II - Syntax Constraints and Requirements

- Number of elements in a circuit is limited to 3Ø.
- Number of nodes in a circuit is limited to 1ØØ.
- Names of nodes should start with an alphabetic character and have a maximum length of 7.
- Each node can be an input to no more than 1Ø elements.
- The maximum number of "included" files is 3.
- Macro names start with an alphabetic character and are limited to 8 characters.
- Parameter names are limited to 8 characters.
- Macros can have no more than 1Ø elements.
- Macros can have no more than 1Ø parameters.
- Macros can have no more than 1Ø local nodes.
- No more than 5 macros can be defined in one file.

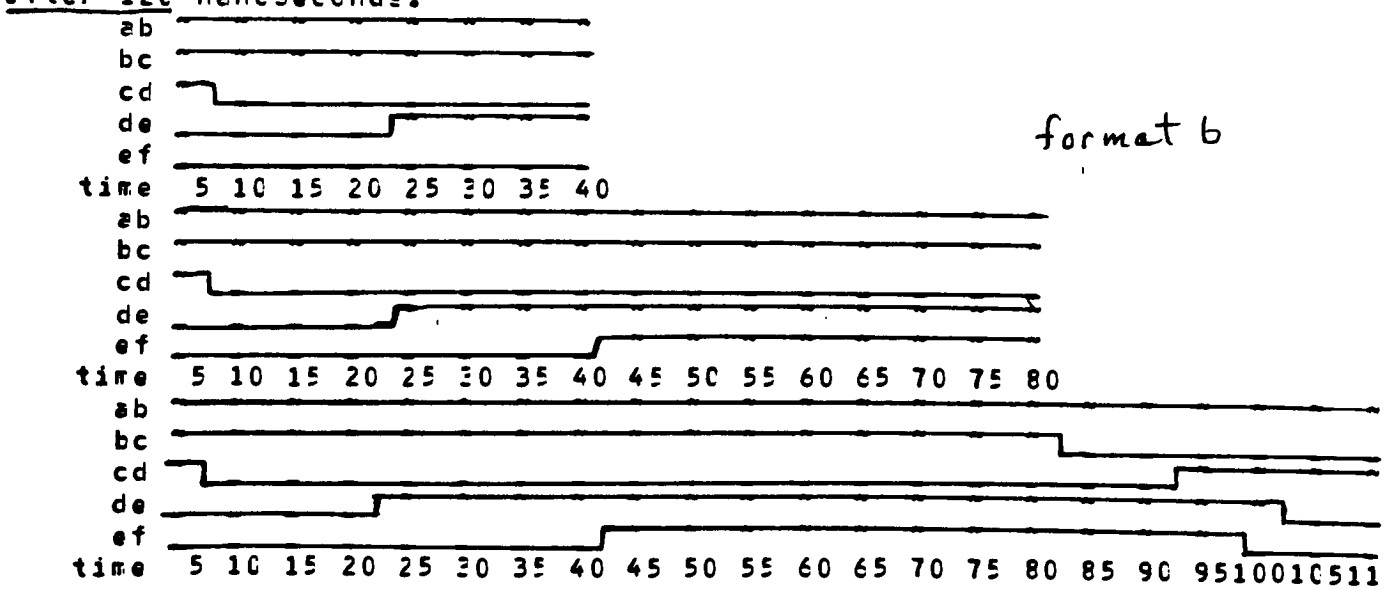
USER'S MANUAL
APPENDIX 3

This appendix contains two sample runs of the program illustrating various features.

Example 1 is the logfile results from testfile cdfile with a simulation step length of 40 nanoseconds and illustrating output formats a and b.

```
      TIME        NODE        OLDVALUE       NEW VALUE

  *      0          ab             0              1
  *      0          bc             0              1
         0          cd             0              1
         0          de             0              0
         0          ef             0              0
         7          cd             1              0
        24          de             0              1
after 40 nanoseconds.
      TIME        NODE        OLDVALUE       NEW VALUE

        44          ef             0              1
after 80 nanoseconds.
      TIME        NODE        OLDVALUE       NEW VALUE

  *     81          bc             1              0
        94          cd             0              1
        96          ef             1              0
       103          de             1              0
after 120 nanoseconds.
```

format a



format b

Example 2 is the scriptfile output of a simulation of testfile macf (see page 3) with the simulation step running as long as there are events scheduled. The output format is a.

```
To run a simulation step - choose from the following options...

To set the length of a simulation step, choose:
   a. 40 nanoseconds
   b. 100 nanoseconds
   c. user selects a step length
   d. automatic halt (no more events scheduled)

 Enter your choice - a, b, c, or d
c
step length is automatic.

To select the output format, choose:

a. Report 1 - the time of each node state change
b. Report 2 - graph of node values with respect to time

Enter your choice - a or b
a
Report a will be issued

The following are the nodes in the circuit...
    ab    bc    cd    ef    fg    t1

Type the names of the nodes to be reported, followed by a $
 or type  all  with no $.
all

Set node by typing the name of the node followed by 1 or 0.

Type s at beginning of line to end input and start simulation.
cd 1
s
```

```
    TIME      NODE      OLD VALUE      NEW VALUE

       0        ab            0              1
       0        bc            0              0
  *        0         cd              0                1
       0        ef            0              0
       0        fg            0              0
       0        t1            0              1
       9        t1            1              0
      19        ef            0              1
      30        fg            0              1
after 30 nanoseconds.

To reset nodes for another simulation step, type node, then value.
Type s at beginning of line to end input and start simulation.
Type q to end the simulation.
bc 1
s
     TIME      NODE      OLD VALUE      NEW VALUE

  *       31        bc            0              1
       42        ab            1              0
       60        ef            1              0
after 60 nanoseconds.


To reset nodes for another simulation step, type node, then value.
Type s at beginning of line to end input and start simulation.
Type q to end the simulation.
cd 0
s
     TIME      NODE      OLD VALUE      NEW VALUE

  *       61        cd            1              0
       80        t1            0              1
       94        fg            1              0
after 94 nanoseconds.


To reset nodes for another simulation step, type node, then value.
Type s at beginning of line to end input and start simulation.
Type q to end the simulation.
bc 0
cd 1
s
     TIME      NODE      OLD VALUE      NEW VALUE

  *       95        bc            1              0
  *       95        cd            0              1
      104        t1            1              0
      115        ab            0              1
      123        fg            0              1
      136        ef            0              1
after 136 nanoseconds.


To reset nodes for another simulation step, type node, then value.
Type s at beginning of line to end input and start simulation.
Type q to end the simulation.

                    simulation.
```

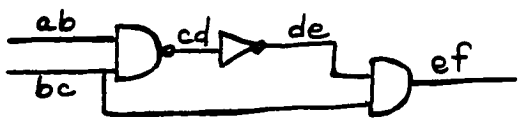
USER'S MANUAL
APPENDIX 4

The output of the first program in the simulator, CDF, is the filename.sim file. It is used as input to the ATS program. It may be necessary to check the .sim file to determine if the simulated circuit is exactly as you designed it, i.e. the hardware description is correct for the circuit design.

This appendix will explain the format of the .sim file so that the user can check it against the original circuit.

Each node is given an index number as it is read in the node list. Each element is given an index number as it is read in the circuit description file. (Note all indexes start with Ø).

Example 1:




```
;testfile cdfile
(node ab bc cd de ef)
(nand2 cd ab bc)
(inv de cd)
(and2 ef bc de)
```

cdfile.sim	explanation
(1) 3 5	(1)number of nodes; number of elements
(2) nand2	(2)name of element #Ø
(3) 2	(3)number of inputs
(4) ab Ø	(4)name of first input; its index
(5) bc 1	(5)name of second input; its index
(6) 1	(6)number of outputs for the element
(7) cd 2	(7)name of output #1; its index
(8) inv	(8)-(12)same information for element 1
(9) 1	
(1Ø) cd 2	
(11) 1	
(12) de 3	
(13) and2	(13)-(18)same as above for element 2
(14) 2	
(15) bc 1	
(16) de 3	
(17) 1	
(18) ef 4	

```
(19)   Ø   ab   1   Ø Ø   -7 -7
(2Ø)   1   bc   2   Ø 1   2 Ø   -7 -7
(21)   2   cd   1   1 Ø   Ø Ø
(22)   3   de   1   2 1   1 Ø
(23)   4   ef   Ø   2 Ø
```

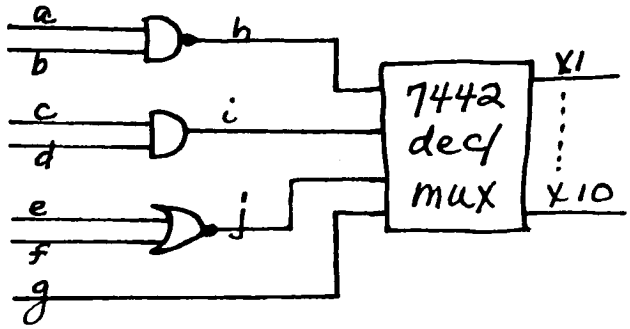
lines (19 - 23)

Column 1 - index of node
Column 2 - name of node
Column 3 - number of elements for which this node is an input (Notice node 4 is not an input to any element)
Columns 4, 5, 6, ... - Groups of two integers indicate first the element(s) to which this node is an input and second which input it is.
These input indicators are followed by output indicators, i.e., the element from which this node is an output.
-7 -7 indicates the node is a primary input.
(In the case of ef, a primary output, the 2 indicates the element from which it is an output and which output it is.)

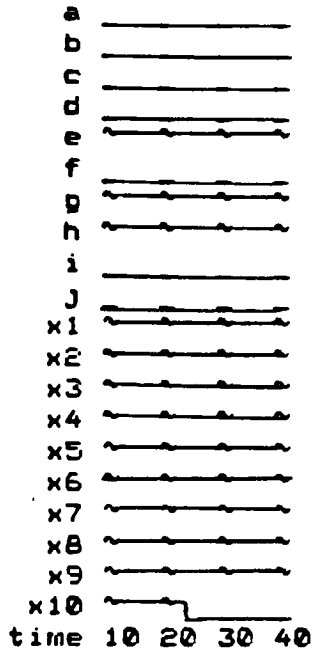
APPENDIX B

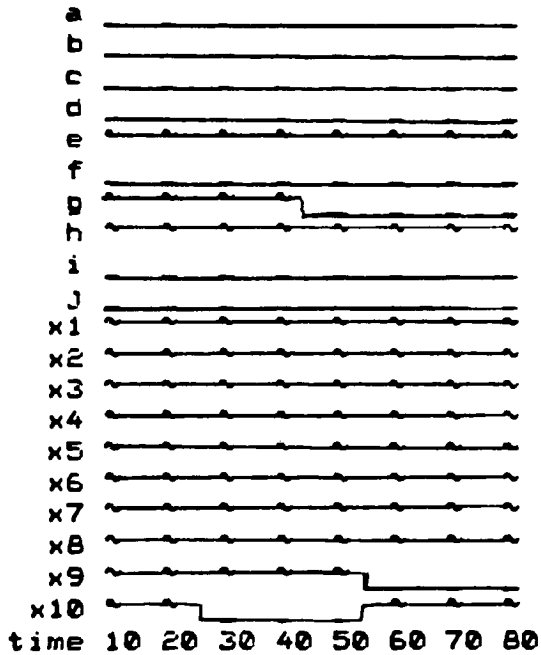
SAMPLE OUTPUT

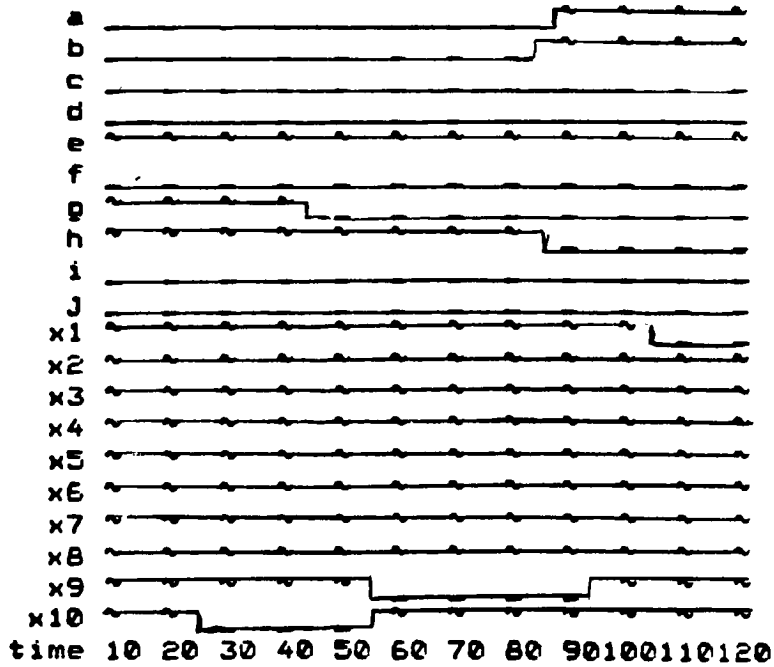
```
;testfile dec4
;tests a 7442 decoder multiplexer
(node a b c d e f g h i j x1 x2 x3 x4 x5 x6 x7 x8 x9 x10)
(nand2 h a b)
(and2 i c d)
(nor2 j e f)
(dec4_10 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 h i j g)
```



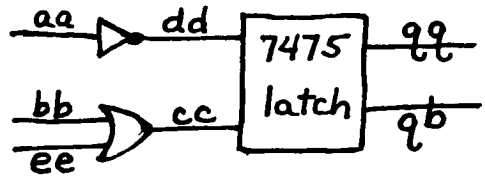

```
        TIME        NODE        OLDVALUE        NEW VALUE

          0           a             0               0
          0           b             0               0
          0           c             0               0
          0           d             0               0
   *      0           e             0               1
          0           f             0               0
   *      0           g             0               1
          0           h             0               1
          0           i             0               0
          0           j             0               1
          0          x1             0               1
          0          x2             0               1
          0          x3             0               1
          0          x4             0               1
          0          x5             0               1
          0          x6             0               1
          0          x7             0               1
          0          x8             0               1
          0          x9             0               1
          0         x10             0               1
          8           j             1               0
         22         x10             1               0
after 40 units of time.
        TIME        NODE        OLDVALUE        NEW VALUE

   *     41           g             1               0
         51         x10             0               1
         55          x9             1               0
after 80 units of time.
        TIME        NODE        OLDVALUE        NEW VALUE

   *     81           a             0               1
   *     81           b             0               1
         88           h             1               0
         98          x9             0               1
        102          x1             1               0
after 120 units of time.
```


a
b
c
d
e
f
g
h
i
J
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
time 10 20 30 40


a
b
c
d
e
f
g
h
i
J
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
time 10 20 30 40 50 60 70 80

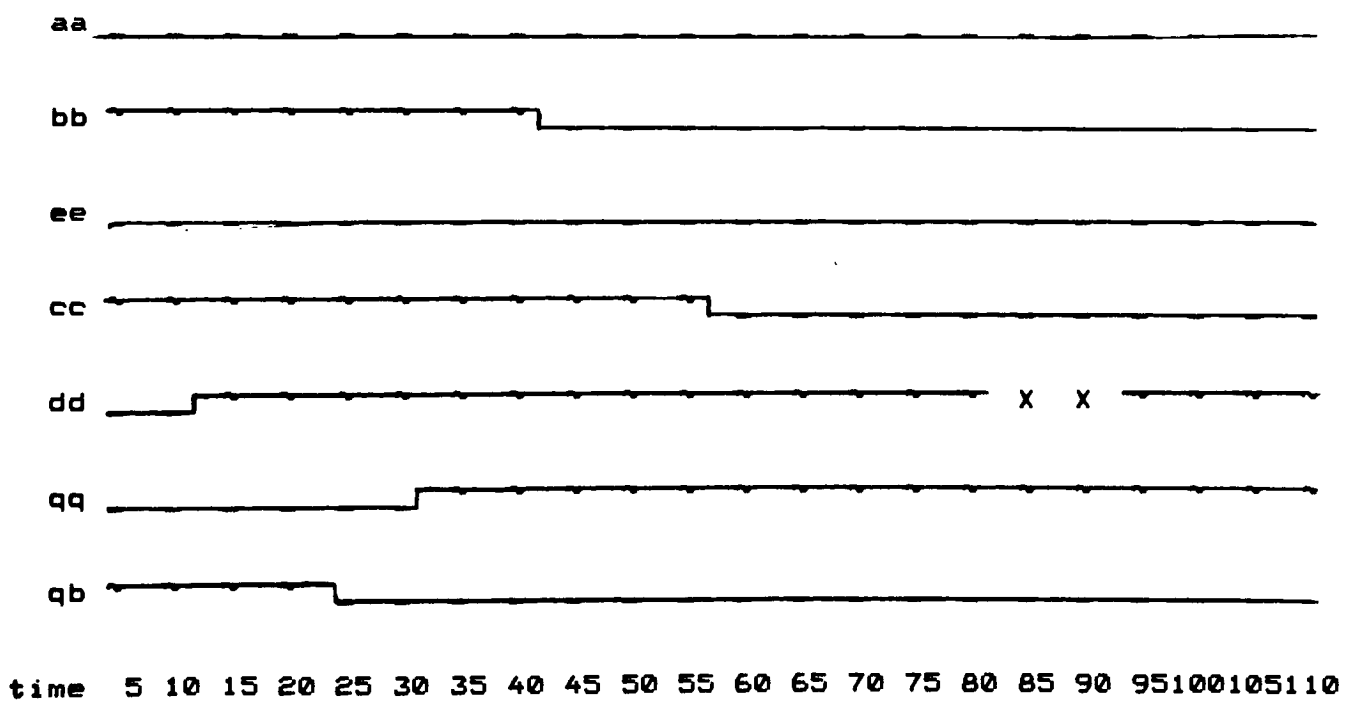

a
b
c
d
e
f
g
h
i
J
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
time 10 20 30 40 50 60 70 80 90100110120

```
;testfile latch2
;contains a 7475 latch
(node aa bb ee cc dd cc qb)
(inv cc aa)
(or2 cc bb ee)
(latch qc qb dd cc)
```

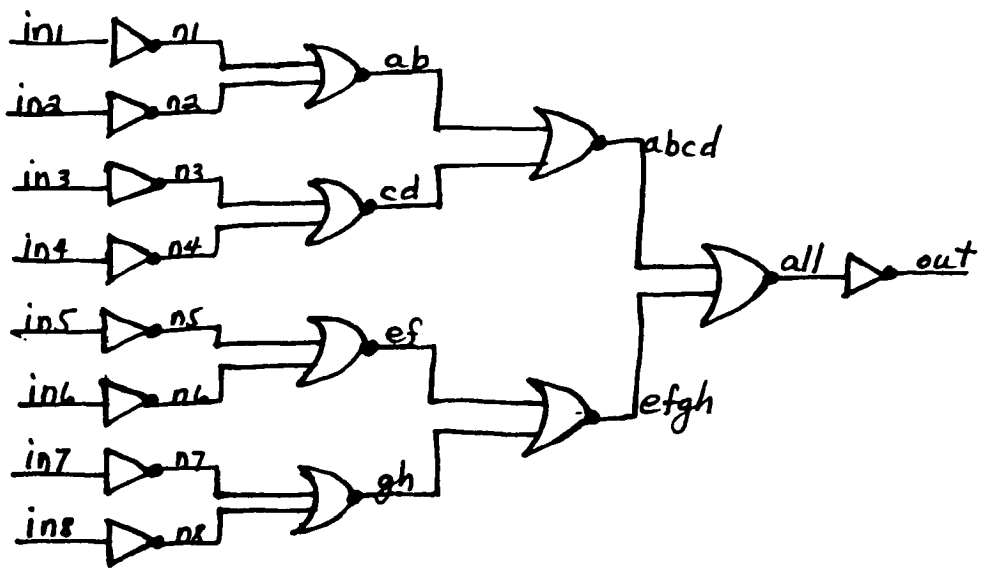



```
        TIME        NODE        OLDVALUE        NEW VALUE

   *        0          aa             0              1
            0          bb             0              0
            0          ee             0              0
            0          cc             0              0
            0          qc             0              1
   *        0          cc             0              1
   *        0          qb             0              0
           13          dd             1              0
after 39 nanoseconds.
        TIME        NODE        OLDVALUE        NEW VALUE

   *       40          bb             0              1
           54          cc             0              1
           66          qc             1              0
           72          qb             0              1
after 79 nanoseconds.
        TIME        NODE        OLDVALUE        NEW VALUE

   *       80          aa             1              0
   *       80          bb             1              0
           97          cc             1              0
           97          dc             0              1
after 119 nanoseconds.
        TIME        NODE        OLDVALUE        NEW VALUE

   *      120          aa             0              1
          130          dc             1              0
after 159 nanoseconds.
        TIME        NODE        OLDVALUE        NEW VALUE

   *      160          bb             0              1
          171          cc             0              1
after 199 nanoseconds.
```

```
The circuit contains the following nodes with unknown values:

qq
qb
Check your circuit to be sure memory elements
are correctly set or cleared.
Set node by typing the name of the node followed by 1 or 0.

Type s at beginning of line to end input and reinitialize.
aa 1
bb 1
s


To run a simulation step - choose from the following options...


To set the length of a simulation step, choose:
   a. 40 nanoseconds
   b. 100 nanoseconds
   c. user selects a step length
   d. automatic halt (no more events scheduled)

 Enter your choice - a, b, c, or d
```



```
;testfile je3
;uses a macro def
(macro or8 (out a b c d e f g h)
(local out all ab cd ef gh abcd efgh)
(nor2 ab a b)
(nor2 cd c d)
(nor2 ef e f)
(nor2 gh g h)
(nor2 abcd ab cd)
(nor2 efgh ef gh)
(nor2 all abcd efgh)
(inv out all)
)
(macro n1 n2 n3 n4 n5 n6 n7 n8 in1 in2 in3 in4 in5 in6 in7 in8 f1)
(inv n1 in1)
(inv n2 in2)
(inv n3 in3)
(inv n4 in4)
(inv n5 in5)
(inv n6 in6)
(inv n7 in7)
(inv n8 in8)
(or8 11 n1 n2 n3 n4 n5 n6 n7 n8)
```



```
      TIME        NODE        OLDVALUE      NEW VALUE

         0          n1             0             1
         0          n2             0             1
         0          n3             0             1
         0          n4             0             1
         0          n5             0             1
         0          n6             0             1
         0          n7             0             1
         0          n8             0             1
   *        0          in1             0             1
         0          in2             0             0
   *        0          in3             0             1
         0          in4             0             0
   *        0          in5             0             1
         0          in6             0             0
   *        0          in7             0             1
         0          in8             0             0
         0           f1             0             0
         0          out             0             1
         0          all             0             0
         0           ab             0             0
         0           cd             0             0
         0           ef             0             0
         0           gh             0             0
         0         abcd             0             1
         0         efgh             0             1
         9           n3             1             0
        11           n5             1             0
        13           n7             1             0
        13           n1             1             0
after 39 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

   *       40          in2             0             1
   *       40          in4             0             1
        49           n4             1             0
        53           n2             1             0
        68           cd             0             1
        74           ab             0             1
after 79 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

        81         abcd             1             0
after 119 nanoseconds.
```

	TIME	NODE	OLDVALUE	NEW VALUE
*	120	in1	1	0
*	120	in2	1	0
*	120	in3	1	0
*	120	in4	1	0
	138	n2	0	1
	139	n3	0	1
	140	n4	0	1
	141	n1	0	1
	150	ab	1	0
	153	cc	1	0

after 159 nanoseconds.

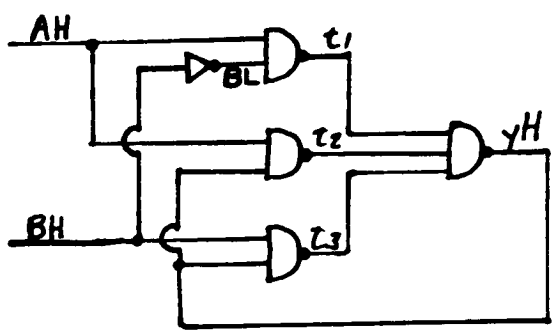
	TIME	NODE	OLDVALUE	NEW VALUE
*	160	in5	1	0
*	160	in6	0	0
*	160	in7	1	0
*	160	in8	0	0
	165	abcc	0	1
	170	n5	0	1
	181	n7	0	1

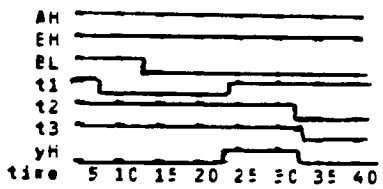
after 199 nanoseconds.

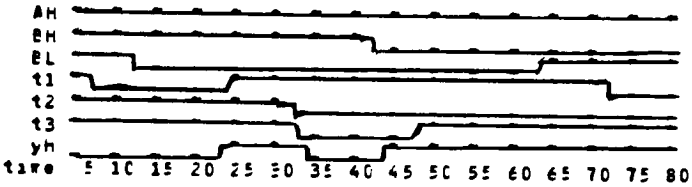
	TIME	NODE	OLDVALUE	NEW VALUE
*	200	in6	0	1
*	200	in8	0	1
	206	n3	1	0
	214	n6	1	0

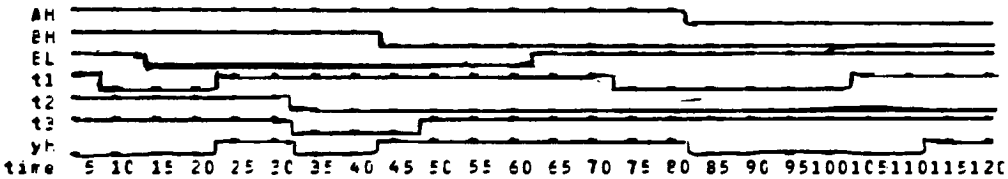
after 239 nanoseconds.

```
;testfile file je2
;circuit contains feedback
(node AH BH BL t1 t2 t3 yH)
(inv BL BH)
(nand2 t1 AH BL)
(nand2 t2 AH yH)
(nand2 t3 BH yH)
(nand3 yH t1 t2 t3)
```









	TIME	NODE	OLDVALUE	NEW VALUE
*	0	AH	0	1
	0	BH	0	0
	0	BL	0	1
	0	t1	0	1
	0	t2	0	1
	0	t3	0	1
	0	yH	0	0
	7	t1	1	0
	23	yH	0	1
	33	t2	1	0

after 40 nanoseconds.

	TIME	NODE	OLDVALUE	NEW VALUE
*	41	AH	1	0
	58	t2	0	1
	62	t1	0	1
	75	yH	1	0

after 80 nanoseconds.

	TIME	NODE	OLDVALUE	NEW VALUE
*	81	AH	0	1
	89	t1	1	0
	103	yH	0	1
	117	t2	1	0

after 120 nanoseconds.

	TIME	NODE	OLDVALUE	NEW VALUE
*	121	BH	0	1
	129	BL	1	0
	130	t2	1	0
	148	t1	0	1

after 160 nanoseconds.

	TIME	NODE	OLDVALUE	NEW VALUE
*	0	AH	0	1
*	0	BH	0	1
	0	BL	0	1
	0	t1	0	1
	0	t2	0	1
	0	t3	0	1
	0	yH	0	0
	7	t1	1	0
	11	BL	1	0
	23	yH	0	1
	24	t1	0	1
	32	yH	1	0
	32	t3	1	0
	33	t2	1	0
	45	yH	0	1
	46	t3	0	1
	51	t2	0	1
	55	t3	1	0
	58	t2	1	0
	61	yH	1	0
	76	yH	0	1
	77	t3	0	1
	78	t2	0	1
	85	t2	1	0
	88	t3	1	0
	89	yH	1	0
	98	yH	0	1

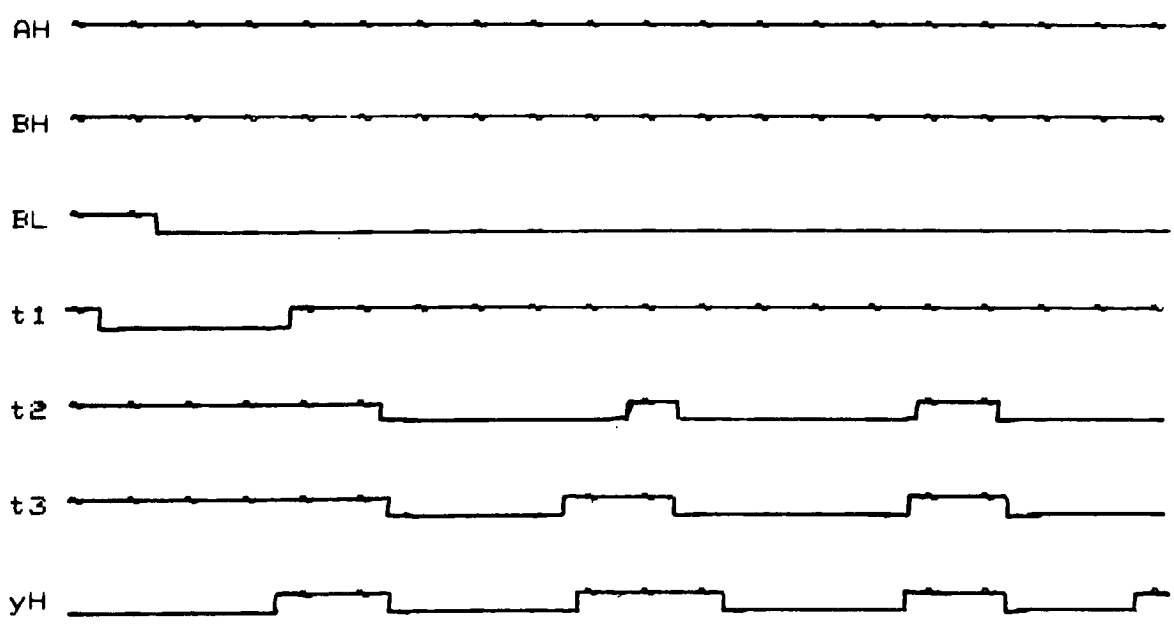
after 100 nanoseconds.

Number of events is 27
State of nodes at the end of the simulationsteo.

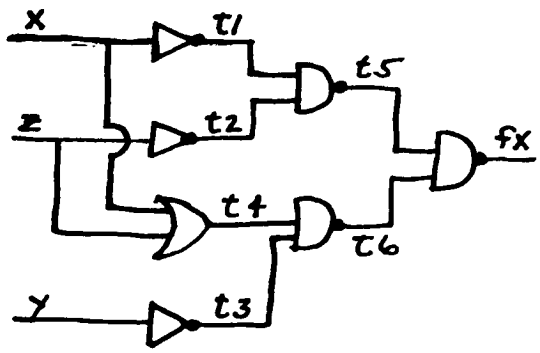
NODE	STATE	
AH	1	last change in AH at 0
BH	1	last change in BH at 0
BL	0	last change in BL at 11
t1	1	last change in t1 at 24
t2	0	last change in t2 at 85
t3	0	last change in t3 at 88
yH	1	last change in yH at 98

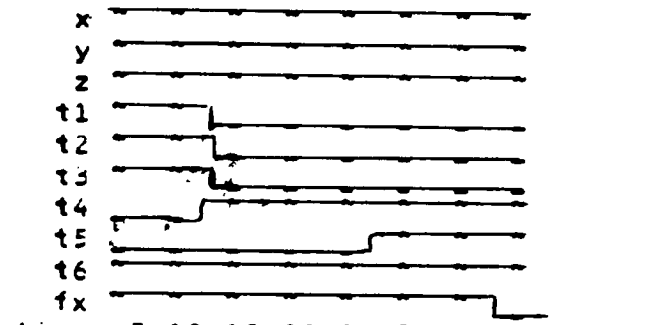
after 100 nanoseconds.

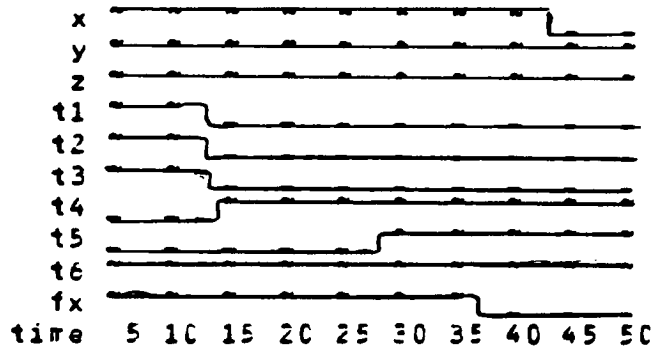
```
To run a simulation step - choose from the following options...

To set the length of a simulation step, choose:
   a. 40 nanoseconds
   b. 100 nanoseconds
   c. user selects a step length
   d. automatic halt (no more events scheduled)

 Enter your choice - a, b, c, or d
b
step length is 100

To select the output format, choose:

a. Report 1 - the time of each node state change
b. Report 2 - graph of node values with respect to time
c. Report 3 - the state of nodes at the end of the sim step

Enter your choice - a or b or c
b
Report b will be issued

The following are the nodes in the circuit...
    AH    BH    BL    t1    t2    t3    yH

Type the names of the nodes to be reported, followed by a $
 or type  all  with no $.
all

Set node by typing the name of the node followed by 1 or 0.

Type s at beginning of line to end input and start simulation.
AH 1
BH 1
```

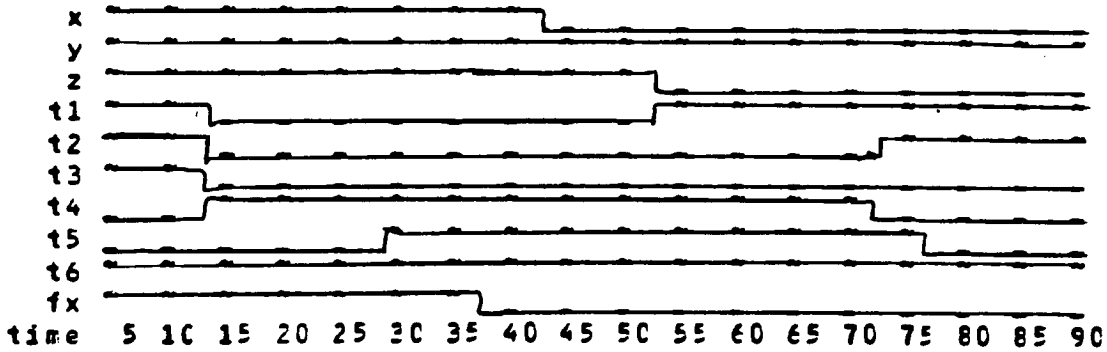


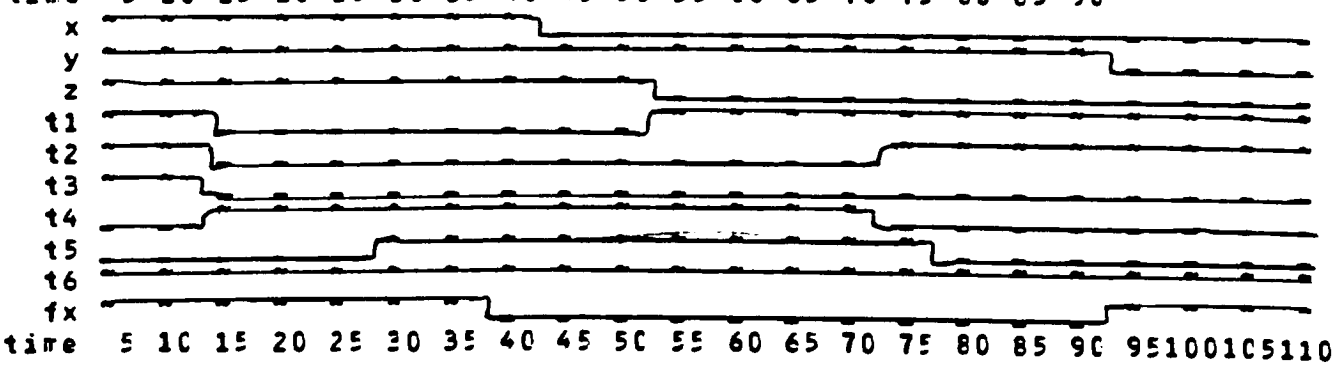
```
; testfile jel
;circuit contains a hazard
(node x y z t1 t2 t3 t4 t5 t6 fx)
(inv t1 x)
(inv t2 z)
(nand2 t5 t1 t2)
(inv t3 y)
(or2 t4 x z)
(nand2 t6 t3 t4)
(nand2 fx t5 t6)
```





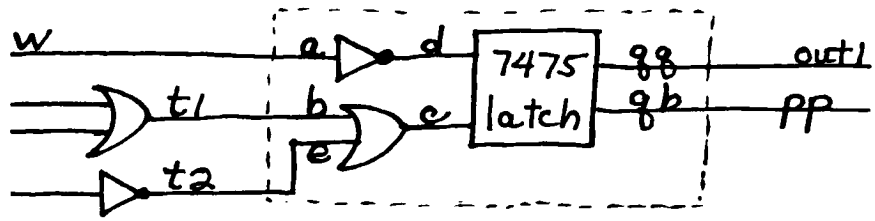


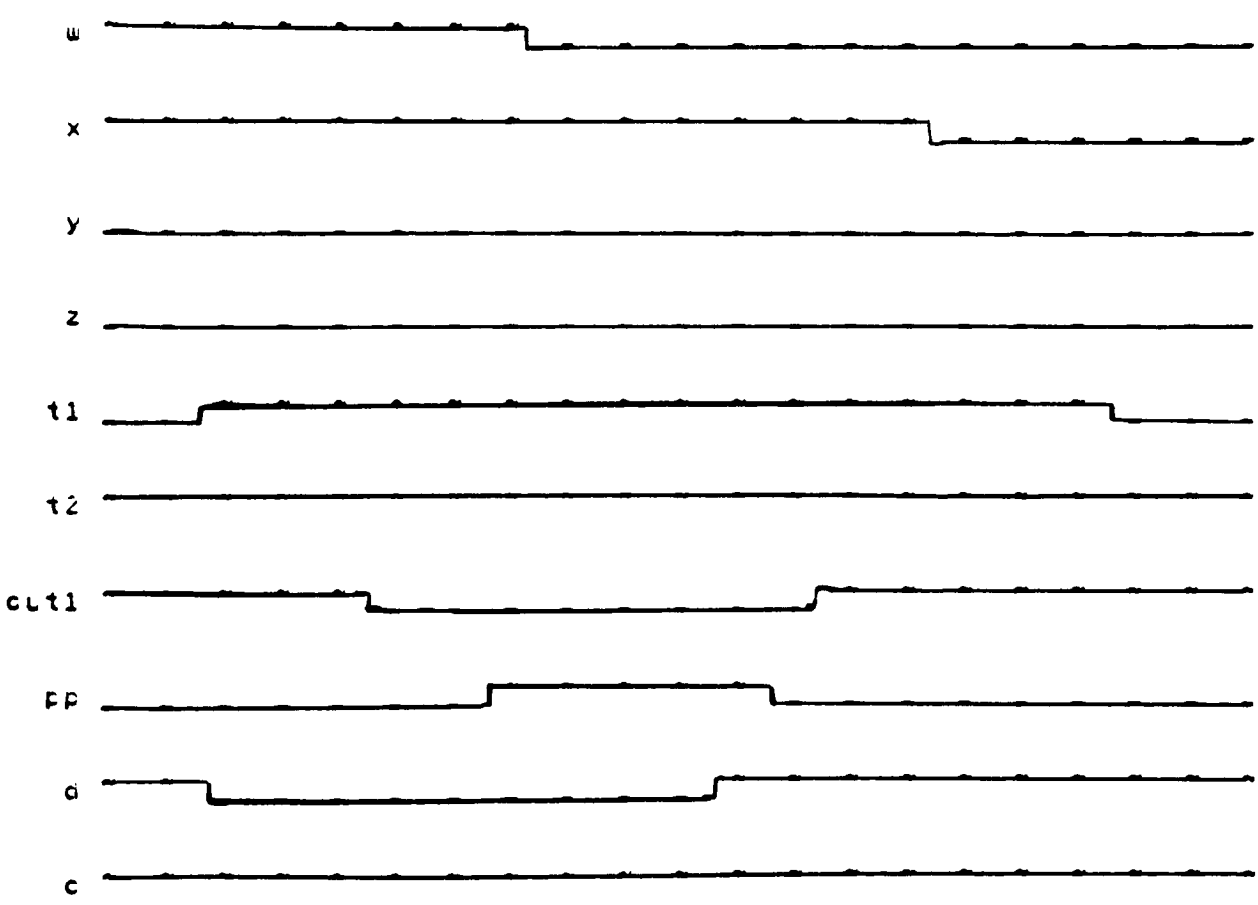




```
      TIME        NODE        OLDVALUE      NEW VALUE

  *        0          x            0             1
  *        0          y            0             1
  *        0          z            0             1
           0         t1            0             1
           0         t2            0             1
           0         t3            0             1
           0         t4            0             0
           0         t5            0             0
           0         t6            0             1
           0         fx            0             1
          11         t3            1             0
          11         t4            0             1
          13         t2            1             0
          13         t1            1             0
          26         t5            0             1
          39         fx            1             0
after 39 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       40          x            1             0
          53         t1            0             1
after 53 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       54          z            1             0
          70         t2            0             1
          70         t4            1             0
          79         t5            1             0
          92         fx            0             1
after 92 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       93          y            1             0
         111         t3            0             1
after 111 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *      112          x            0             1
  *      112          z            0             1
         123         t1            1             0
         124         t4            0             1
         125         t2            1             0
         133         t6            1             0
         138         t5            0             1
after 138 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *      139          y            0             1
         149         t3            1             0
         162         t6            0             1
         171         fx            1             0
after 171 nanoseconds.
```

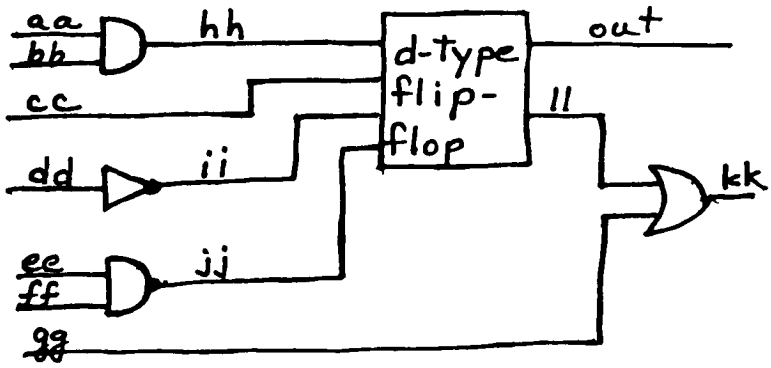
```
;testfile 13
;has a macro which includes a latch
(macro l2 (a b e qq qb)
(local d c)
(inv c a)
(or2 d b e)
(latch qq qb d c)
)
(macro u x y z t1 t2 out1 pp)
(or2 t1 x y)
(inv t2 z)
(l2 u t1 t2 out1 pp)
```





time 6 11 16 21 26 31 36 41 46 51 56 61 71 76 81 86 91 96101106111

```
;testfile dff2
;tests a 7474 d-type flip-flop
(node aa bb cc dd ee ff gg hh ii jj kk ll out)
(and2 hh aa bb)
(inv ii dd)
(nand2 jj ee ff)
(dff out ll hh cc ii jj)
(or2 kk ll gg)
```



	TIME	NODE	OLD VALUE	NEW VALUE
*	0	aa	0	1
*	0	bb	0	1
	0	cc	0	0
	0	dd	0	0
	0	ee	0	0
	0	ff	0	0
	0	gg	0	0
	0	hh	0	0
	0	ii	0	1
	0	jj	0	1
	0	kk	0	0
*	0	ll	0	0
*	0	out	0	1
	15	hh	0	1
	38	ll	0	1

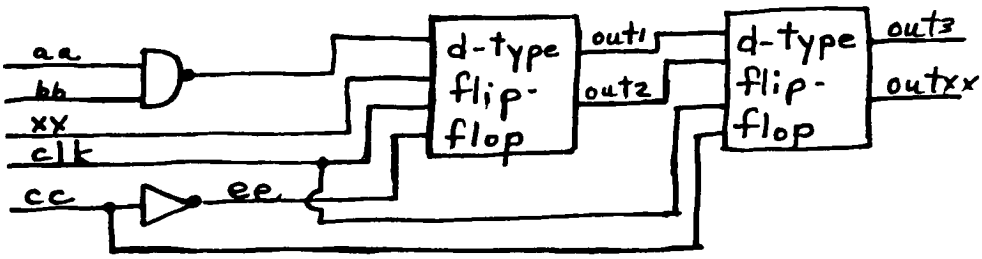
after 39 nanoseconds.

	TIME	NODE	OLD VALUE	NEW VALUE
*	[illegible]	aa	[illegible]	0
	46	[illegible]	0	1
	[illegible]	cc	1	1
	[illegible]	[illegible]	1	0

after [illegible] nanoseconds.

```
      TIME        NODE        OLDVALUE      NEW VALUE

  *        80         cc             0              1
           85         11             1             -1
           95        out             0              1
          100         kk             1             -1
          107         11            -1              0
after 115 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       120         cc             0              1
          123         kk            -1              0
          132         11             1              0
after 139 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       240         cc             0              1
          248         kk             0             -1
          258        out            -1              1
          265         11            -1              0
after 279 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

          290         kk            -1              0
after 319 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *       320         cc             0              0
after 359 nanoseconds.
```

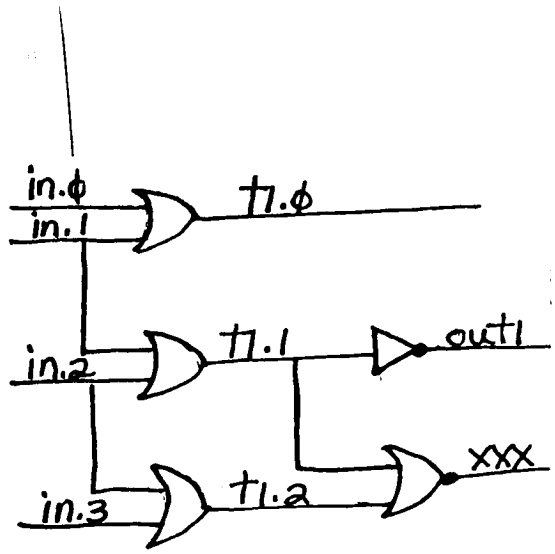
```
;testfile dff3
;contains two d flip-flops
(node aa bb xx clk cc dd ee out1 out2 out3 outxx)
(nand2 dd aa bb)
(inv ee cc)
(dff out1 out2 dd xx clk ee)
(dff out3 outxx out1 out2 clk cc)
```

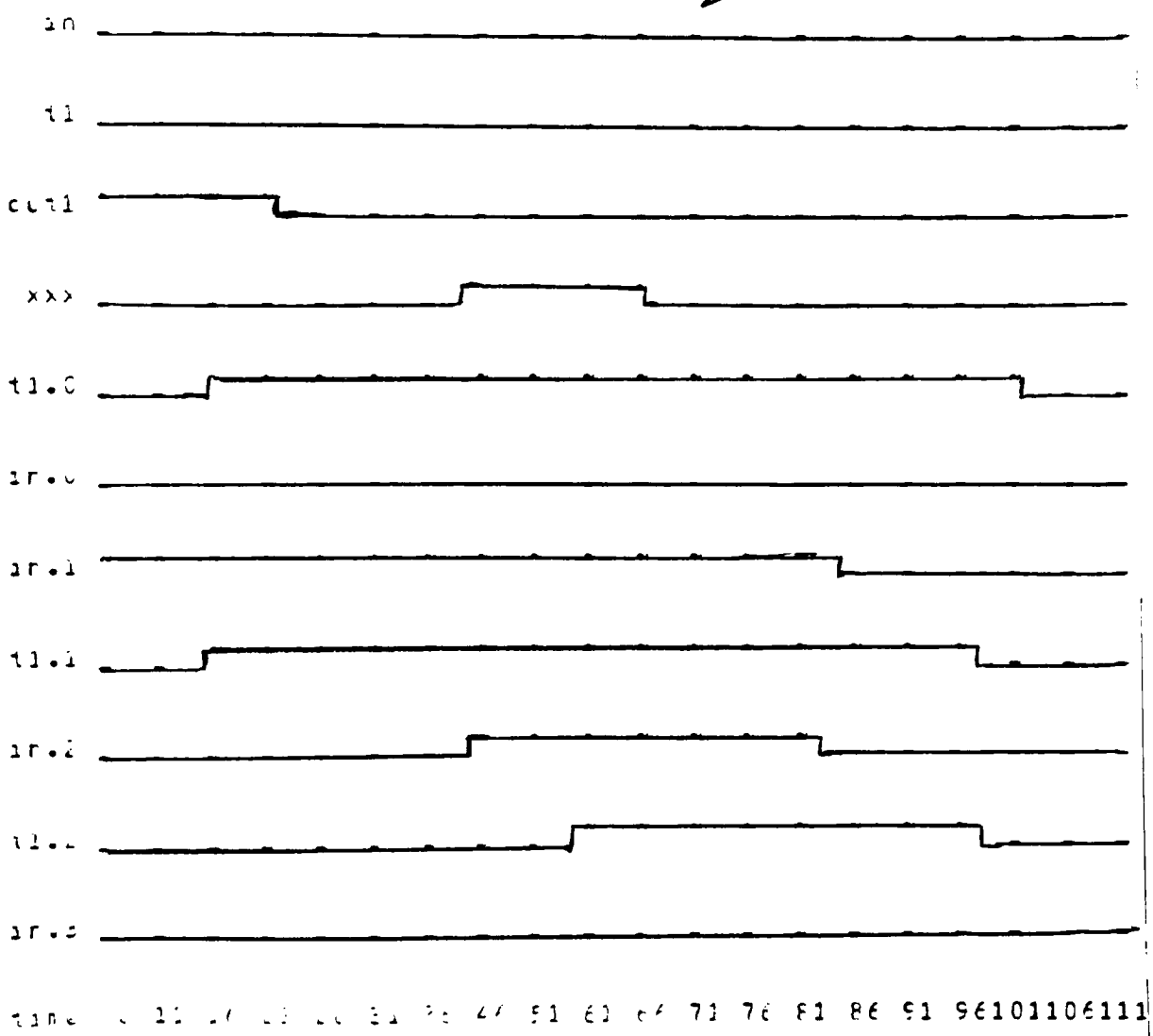


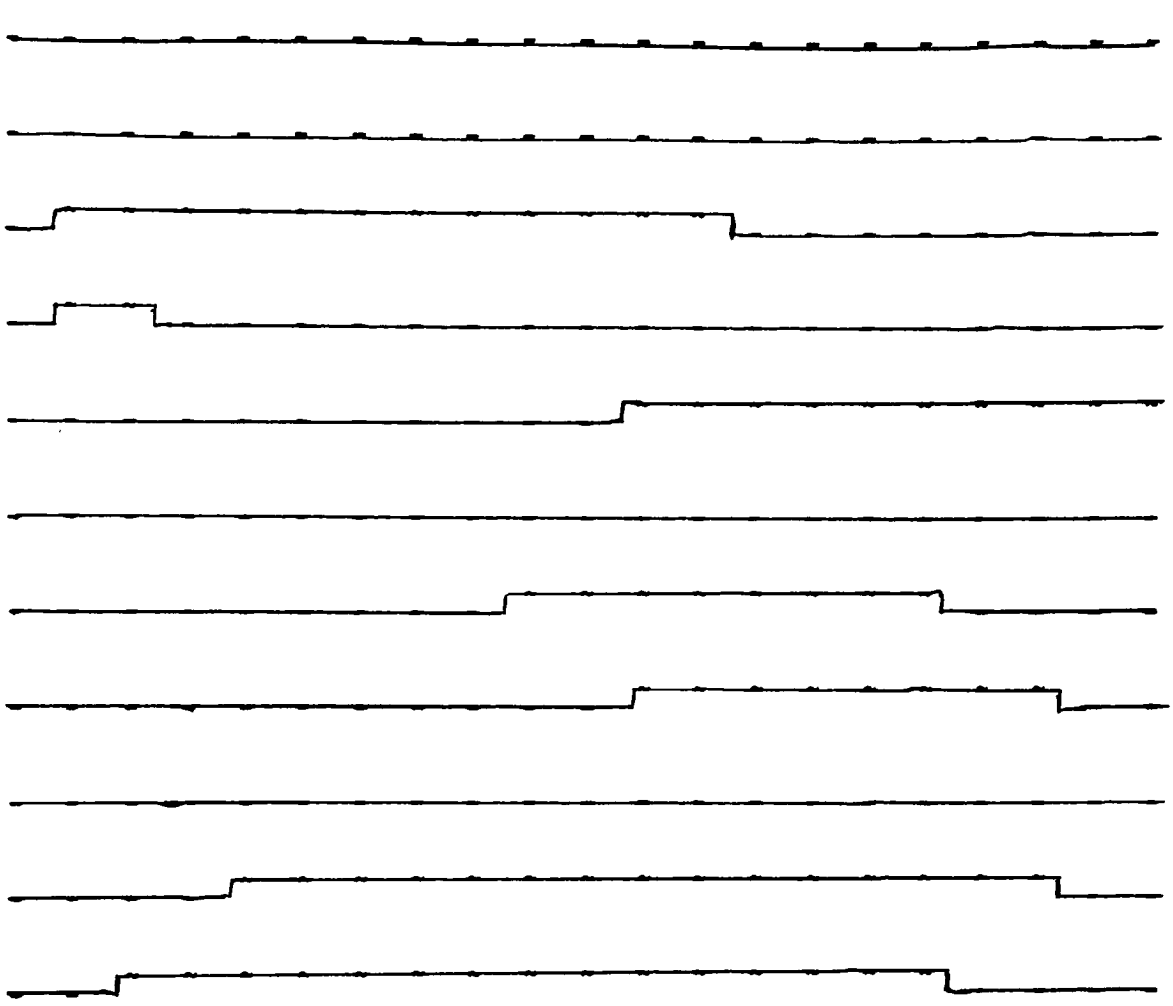
```
      TIME        NODE        OLDVALUE      NEW VALUE

  *          0          aa            0              1
  *          0          bb            0              1
             0          xx            0              0
  *          0          clk           0              1
  *          0          cc            0              1
             0          dc            0              1
             0          ee            0              1
  *          0        out1            0              0
  *          0        out2            0              1
             0        out3            0              1
             0       outxx            0              0
             7          dc            1              0
            14          ee            1              0
            34        out2            1             -1
after 35 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *         40          cc            1              0
            45        out1            0             -1
            5c       outxx            0             -1
            60        out3            1             -1
            61          ee            0              1
after 75 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *         80          xx            0              1
            97        out1           -1              1
           111        out2           -1              0
after 115 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *        120         clk            1              0
           134       outxx           -1              1
           135        out3           -1              0
after 155 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *        160          cc            0              0
  *        160         clk            0              1
after 195 nanoseconds.
      TIME        NODE        OLDVALUE      NEW VALUE

  *        200         clk            1              0
  *        200          cc            0              1
           213          ee            1              0
after 235 nanoseconds.
```

```
      TIME       NODE       OLDVALUE      NEW VALUE

  *      0         ee            0             1
  *      0         bb            0             1
  *      0         xx            0             0
         0        clk            0             0
         0         cc            0             0
         0         dc            0             1
         0         ee            0             1
         0       out1            0             0
         0       out2            0             1
  *      0       out3            0             1
  *      0      outxx            0             0
         7         dc            1             0
        32       out2            1            -1
        36       out1            0            -1
after 39 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *     40        clk            0             1
        55       out2            1            -1
        66      outxx            0            -1
after 79 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *     80         xx            0             1
        95       out1           -1             1
       100       out2           -1             0
after 119 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    120         cc            0             1
       126      outxx           -1             1
       134         ee            1             0
       144       out3           -1             0
after 159 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    160        clk            1             0
after 199 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    200         cc            1             0
  *    200        clk            0             1
       221         ee            0             1
after 239 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    240        clk            1             0
after 279 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    280         cc            0             0
       286         ee            1             0
after 319 nanoseconds.
      TIME       NODE       OLDVALUE      NEW VALUE

  *    320        clk            0             1
       329       out1            1             0
       357      outxx            1            -1
after 359 nanoseconds.
```

```
;file rlist
;tests the repeat construct
(node in t1 out1 xxx)
(repeat i 0 2
(nor2 t1.i in.i in.(+i 1))
)
(inv out1 t1.1)
(nor2 xxx out1 t1.2)
```



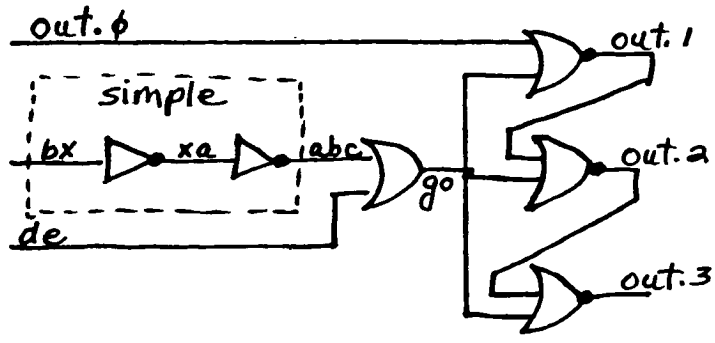




1161211261311411461511561661166 52176181186191196201206211221226

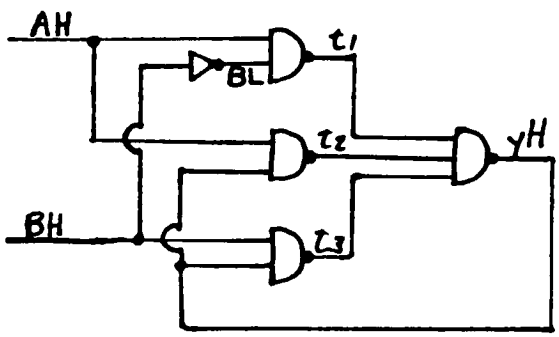
```
;testfile 11.
;combines include, macro and repeat
(include simple)
(node abc go de out)
(or2 go de abc)
(repeat 3 0 2
(or2 out.(+ 1) out.3 go)
)
```

```
;testfile simple
;this file contains a macro
(macro invert (a b)
(inv a1 a2)
)
(node xa bx abc)
(inv xa bx)
(invert abc xa)
```



```
        TIME        NODE        OLDVALUE      NEW VALUE

          0          xor            0              1
   *              0          box            0              1
          0         and.c           0              0
          0         nor.c           0              0
          0           or.c          0              0
          0          not.a          1              0
          0          out.1          0              1
          0          out.0          1              0
          0          out.2          0              0
          0          out.3          0              1
         12            xor          1              0
         21           and.c         1              1
         42            or.c          0              1
         50          out.3           1              0
         50          out.1           1              0
after 50 nanoseconds.
        TIME        NODE        OLDVALUE      NEW VALUE

   *       60       out.0            0              1
after 109 nanoseconds.
        TIME        NODE        OLDVALUE      NEW VALUE

   *       100          box            1              0
      140            xor          0              1
      145           and.c         1              0
      164            or.c          1              0
after 175 nanoseconds.
        TIME        NODE        OLDVALUE      NEW VALUE

   *       200       out.0            1              0
      205          out.2                          1
      205          out.3           0              1
      215          out.3           1              0
      215            xor          1              0
      215          out.1           0              1
      220          out.2           1              0
      220           and.c          0              1
      224            or.c          0              1
      226          out.3           0              1
      231          out.3           1              0
      232          out.1           1              0
after 289 nanoseconds.
```

The following results show a comparison between simulation runs on TIMSIM and RNL for the circuit illustrated.



TIMSIM

TIME	NODE	LEVEL	NEW VALUE
*	0	F	1
	0	H	0
	0	tL	1
	0	t1	1
	0	t2	1
	0	t3	1
	0	yH	0
	7	t3	1
	[illegible]	yH	1
	[illegible]	t2	1

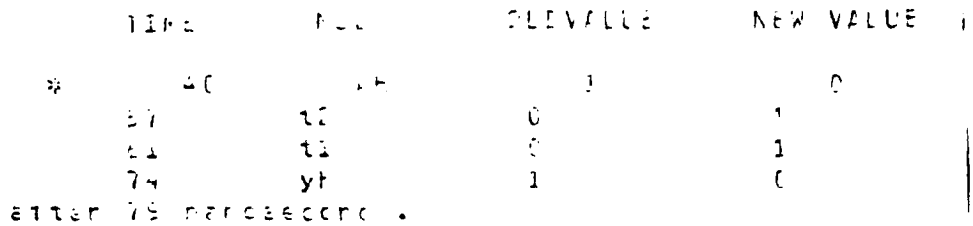
after [illegible] records.

RNL

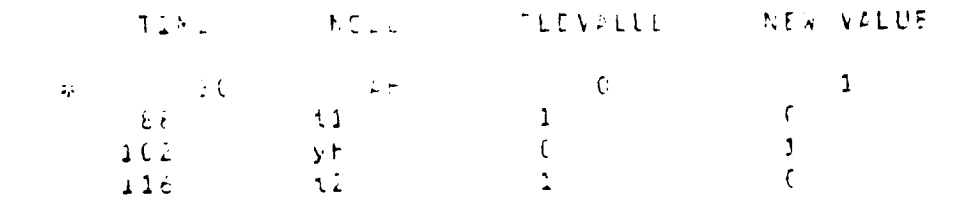
```
Step begins @ 0 ns.
STATE:
Current time= 100
output=X input=XXX
done
1 AH Bn
done
s
```

```
Step begins @ 100 ns.
BH=0 @ 0
An=0 @ 0
BL=1 @ 0.8
t3=1 @ 1.2
t2=1 @ 1.2
t1=1 @ 1.2
yH=0 @ 1.6
STATE:
Current time= 200
o    t=0b0 input=0b001
done
h An
done
s
```

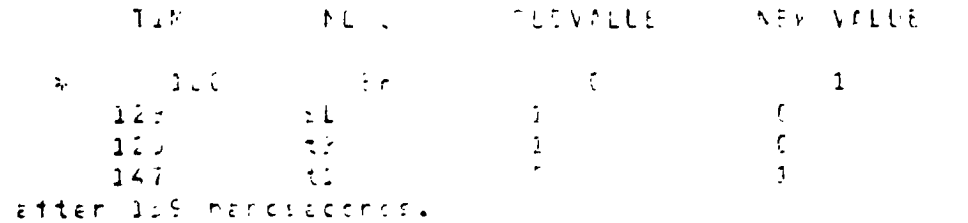
```
Step begins @ 200 ns.
AH=1 @ 0
t1=0 @ 0.3
yH=1 @ 1.9
t2=0 @ 2.2
STATE:
Current time= 300
output=0b1 input=0b101
done
```



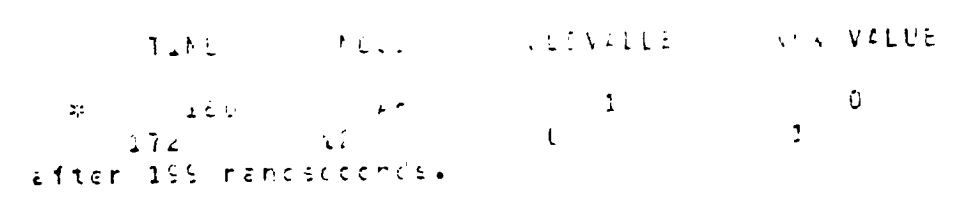
```
        TIME          NODE          OLDVALUE        NEW VALUE

  #          40            AH              1               0
             57            t2              0               1
             44            t1              0               1
             74            yH              1               0
after 75 nanoseconds.
```



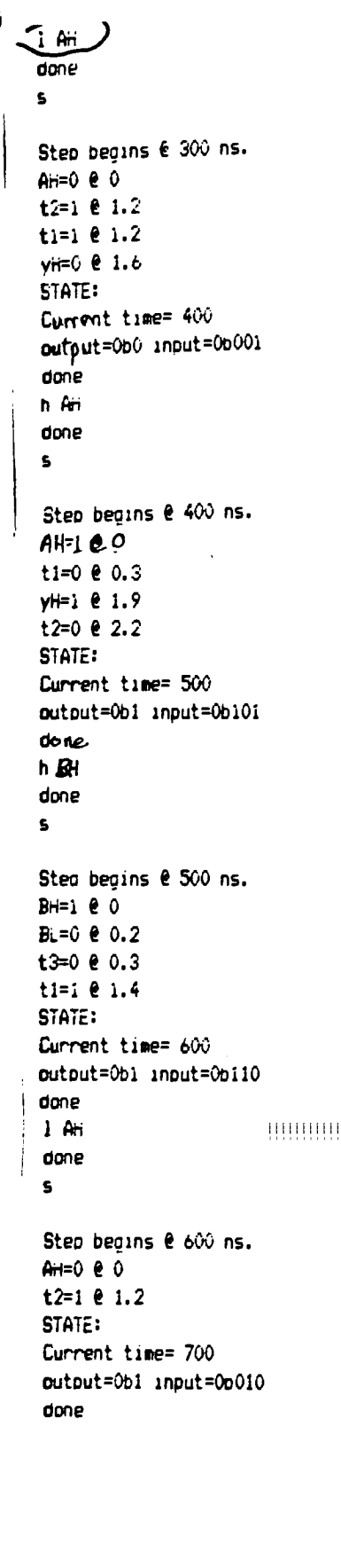
```
        TIME          NODE          OLDVALUE        NEW VALUE

  #          20            AH              0               1
             82            t1              1               0
            102            yH              0               1
            116            t2              1               0
after 119 nanoseconds.
```



```
        TIME          NODE          OLDVALUE        NEW VALUE

  #          10            BH              0               1
            127            BL              1               0
            129            t3              1               0
            147            t1              0               1
after 149 nanoseconds.
```



```
        TIME          NODE          OLDVALUE        NEW VALUE

  #          10            AH              1               0
            172            t2              0               1
after 199 nanoseconds.
```



```
i AH
done
s

Step begins @ 300 ns.
AH=0 @ 0
t2=1 @ 1.2
t1=1 @ 1.2
yH=0 @ 1.6
STATE:
Current time= 400
output=0b0 input=0b001
done
h AH
done
s

Step begins @ 400 ns.
AH=1 @ 0
t1=0 @ 0.3
yH=1 @ 1.9
t2=0 @ 2.2
STATE:
Current time= 500
output=0b1 input=0b101
done
h BH
done
s

Step begins @ 500 ns.
BH=1 @ 0
BL=0 @ 0.2
t3=0 @ 0.3
t1=1 @ 1.4
STATE:
Current time= 600
output=0b1 input=0b110
done
l AH
done
s

Step begins @ 600 ns.
AH=0 @ 0
t2=1 @ 1.2
STATE:
Current time= 700
output=0b1 input=0b010
done
```

APPENDIX C

ANALYSIS OF THE THREE-VALUED TRUTH TABLE FOR THE 7474 D-TYPE FLIP-FLOP

This appendix will outline the analysis of the functionality of the 7474 d-type flip-flop. This process is necessary in order to determine the correct outputs of the IC for all possible combinations of inputs using 3-valued logic. The three values are Ø, 1 and U. In this thesis, U stands for unknown, in which case U is either Ø or 1 but it cannot be determined which it is. There is also a special case in certain ICs, the 74 being one, in which U may represent an indeterminate state, e.g., some level between Ø and 1 or oscillating. In either case, however, it is sufficient to say that the value of U is unknown.

U is a level that arises in two situations. (1) One situation is at powerup, when output nodes have an unknown value until some logical combination of inputs to a gate determine its output levels. (2) During the operation of a circuit, the inputs of an IC assume a particular combination of values which result in the output values being indeterminate. An example in the 7474 is the situation where both $\overline{\text{preset}}$ and $\overline{\text{clear}}$ are Ø, trying to preset and clear at the same time.

The 7474 is a d-type positive-edge-triggered flip-flop. The inputs are $\overline{\text{preset}}$, $\overline{\text{clear}}$, clock and data and the outputs are known as Q and $\overline{Q}$. Considering first the 2-valued truth table (TTL Data Book) which is shown in Fig. C.1.

	$\overline{PRE}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
1	0	1	X	X	1	0
2	1	0	X	X	0	1
3	0	0	X	X	U	U
4	1	1	↑	1_0	1	0
5	1	1	↑	0_0	0	1
6	1	1	0	X	Q_0	$\overline{Q}_0$

Fig. C.1

As can be seen, for lines 1 and 2, the outputs are known. In line 3 when the inputs $\overline{\text{preset}}$ and $\overline{\text{clear}}$ are Ø, the outputs are indeterminate and are expressed as U. Lines 4 and 5 suggest a different problem.

The basic premise of event-directed simulation is that an event (a change) occurs, thereby causing other potential changes. Therefore, the truth table for a gate or IC is accessed when a change has occurred. In lines 4 and 5 of the truth table the change in clock is important. In order to know if the clock is making a Ø to 1 transition it is necessary to know its old value as well as its new value. The 7474 differs from all the other ICs used in this simulator in that the outputs depend not only on the present

value of the inputs, but also on the former values of both the inputs and the outputs. It is necessary to know the old value of data since that value is transmitted to Q on the rising clock pulse.

In determining the outputs on lines 6, 7, 8 of Fig. C.1, the memory values for the outputs are used. That is, for this particular set of inputs the outputs remain the same.

On the following pages, an analysis of the 3-valued truth table for the 7474 is undertaken. First a change in $\overline{\text{preset}}$ is considered, then a change in $\overline{\text{clear}}$. The situation in which the clock changes provides the largest number of cases to consider. Lastly a change in the data value is noted. The outcome of this analysis is a C function, stored in the library, which determines the output of the 7474 based on which input changed and how.

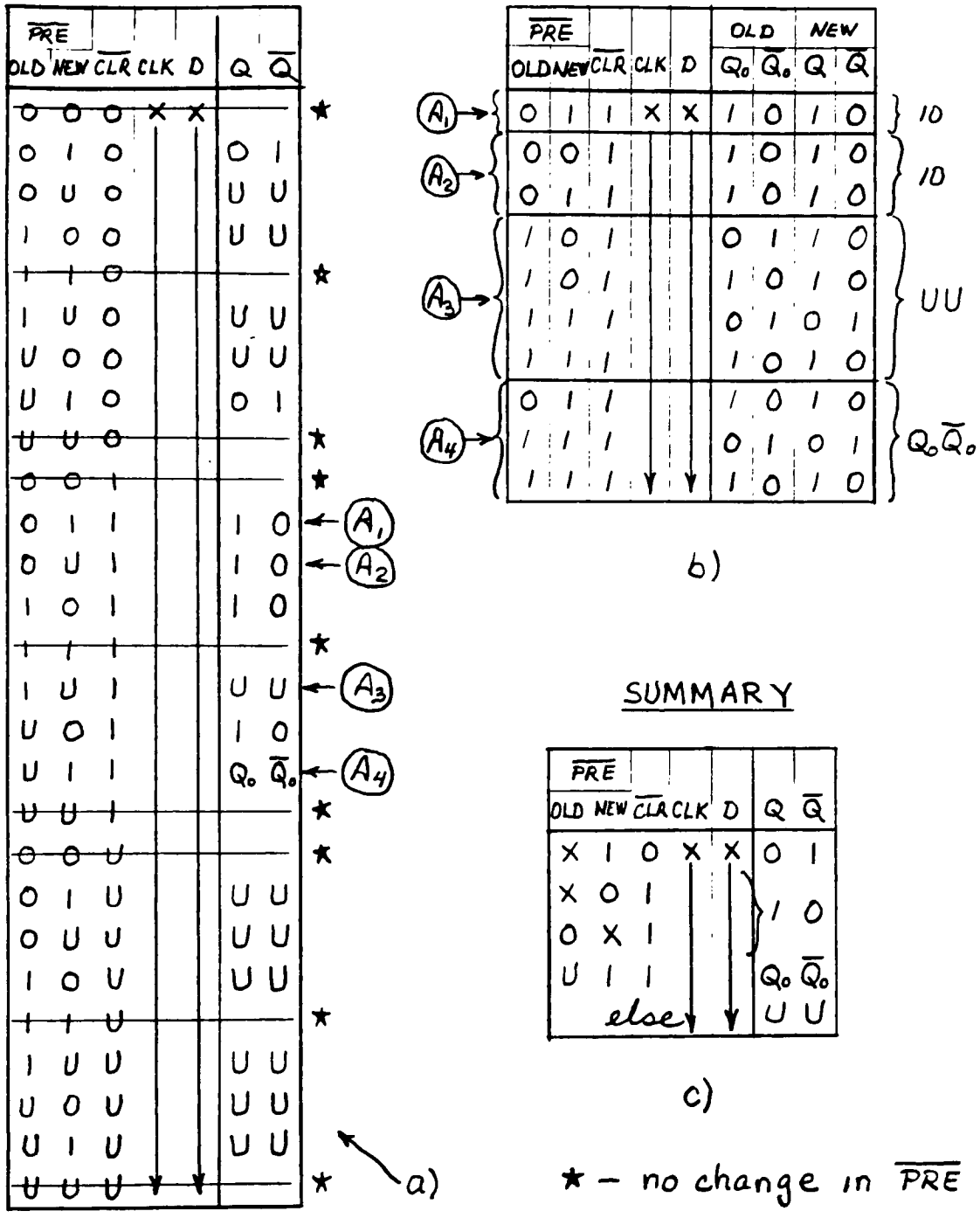


Fig. C.2 Ternary Truth Table for 7474 d-type flip-flop assuming a change in preset only. a) primary truth table b) individual evaluations of several lines c) summary of primary truth table.

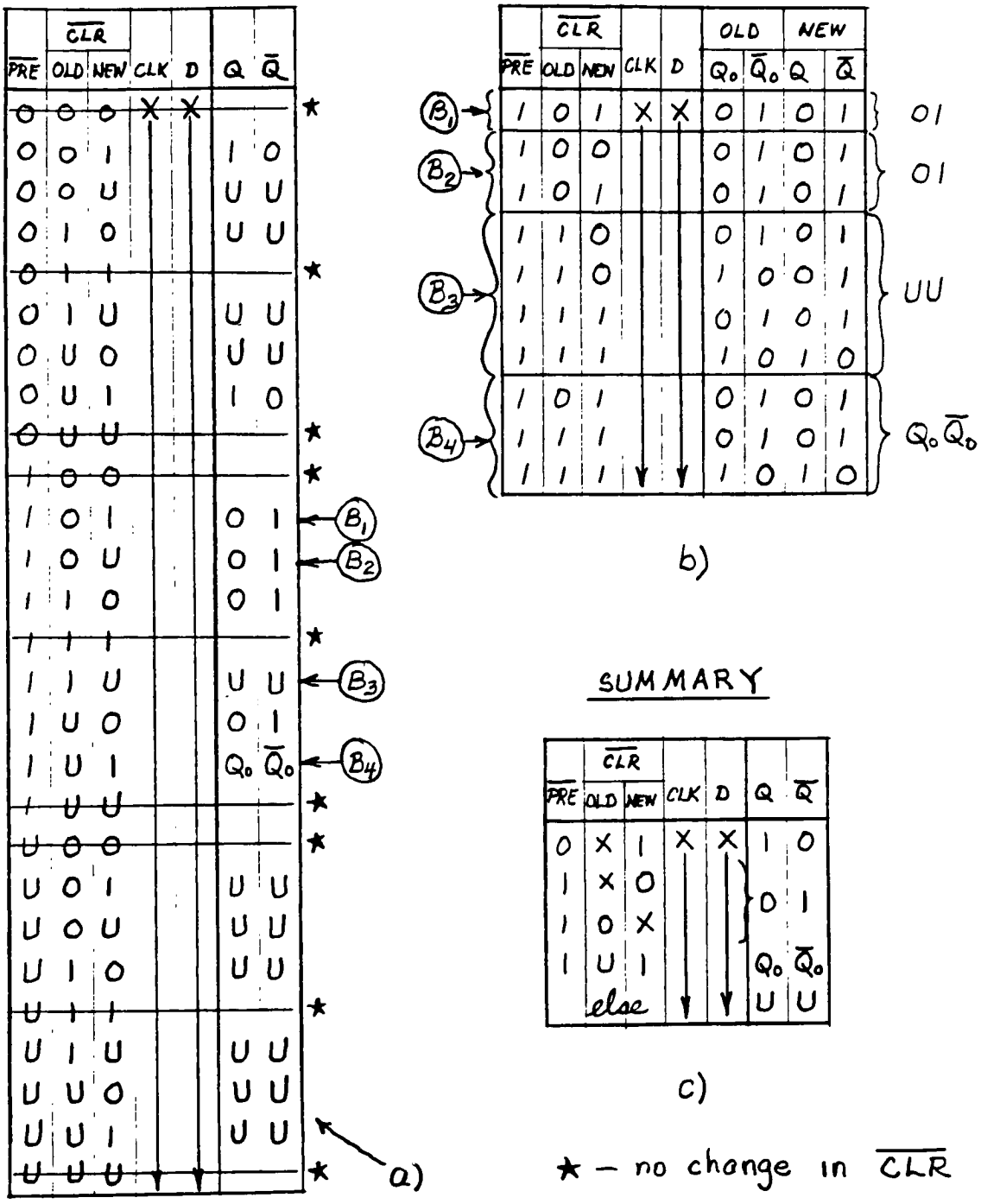


Fig. C.3 Ternary Truth Table for the 7474 d-type flip-flop assuming a change in clear only. a) primary truth table b) individual evaluation of several lines c) summary of primary truth table.

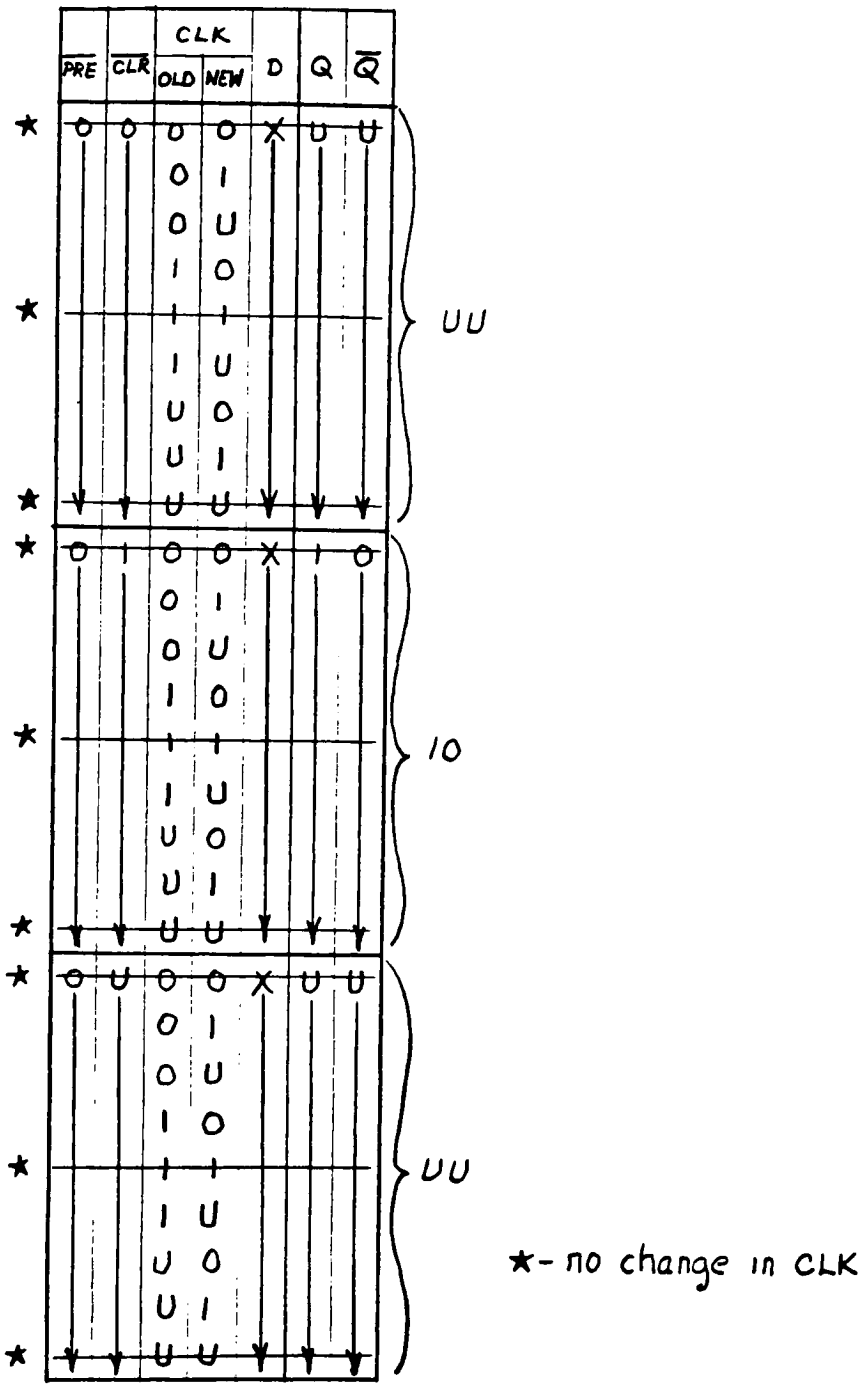


Fig. C. 4 Ternary Truth Table for the 7474 d-type flip-flop assuming a change in clock only and with various combinations of values for preset and clear.

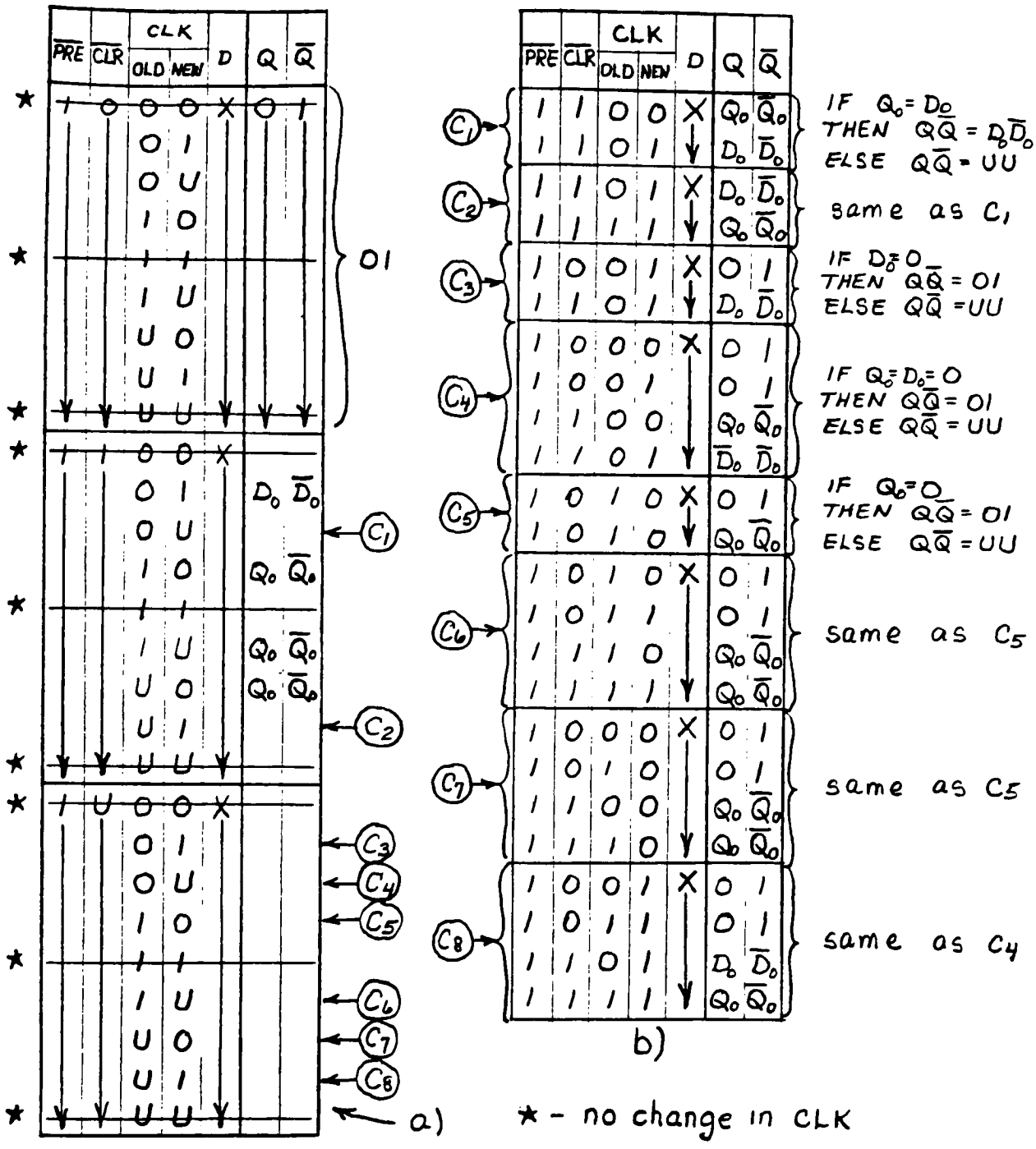


Fig. C. 5 Ternary Truth Table for the 7474 d-type flip-flop assuming a change in clock only and with various combinations of values for preset and clear. a) primary truth table b) individual evaluations of certain lines.

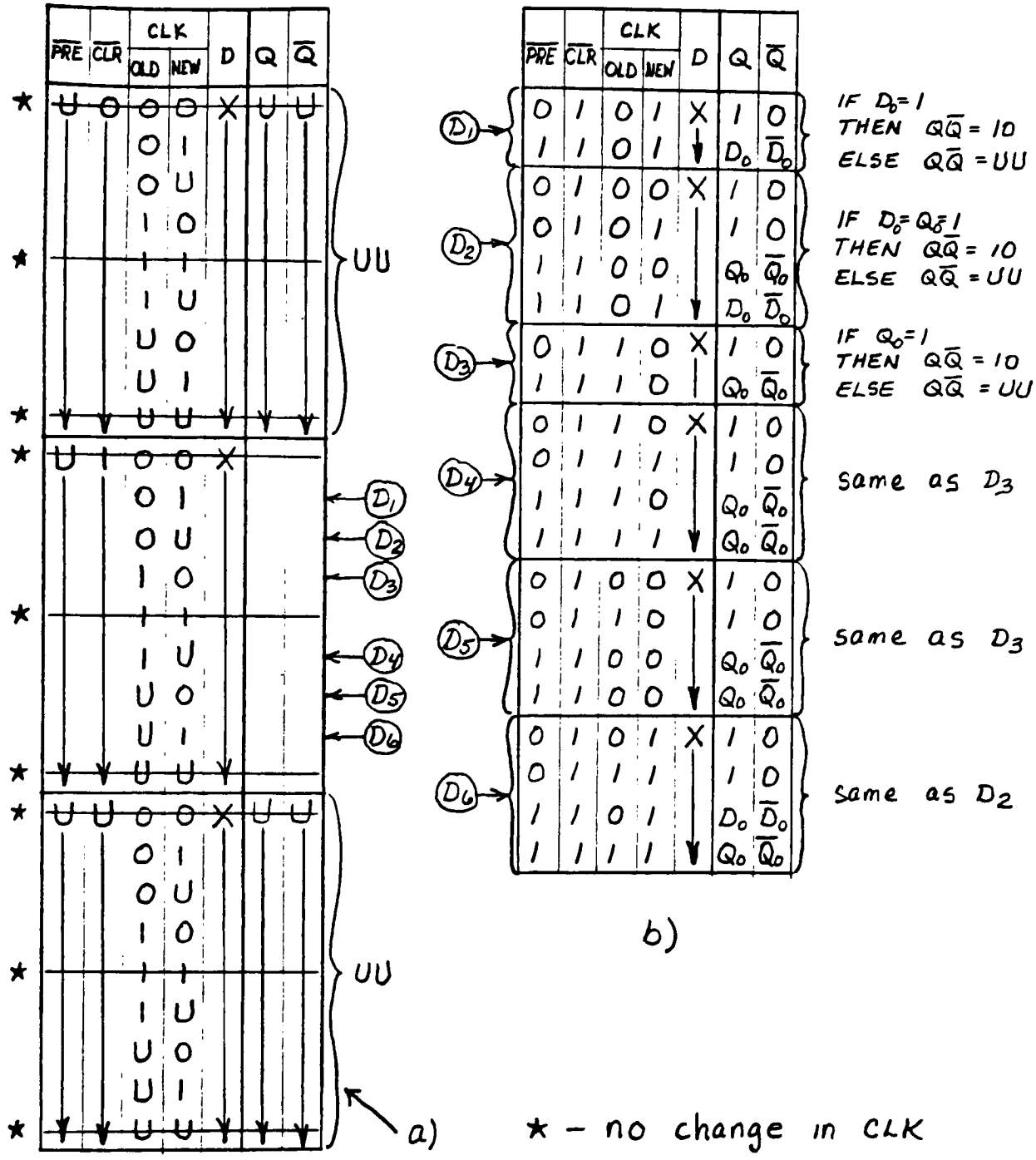


Fig. C. 6 Ternary Truth Table for the 7474 d-type flip-flop assuming a change in clock only and with various combinations of values for preset and clear. a) primary truth table b) individual evaluations of certain lines.

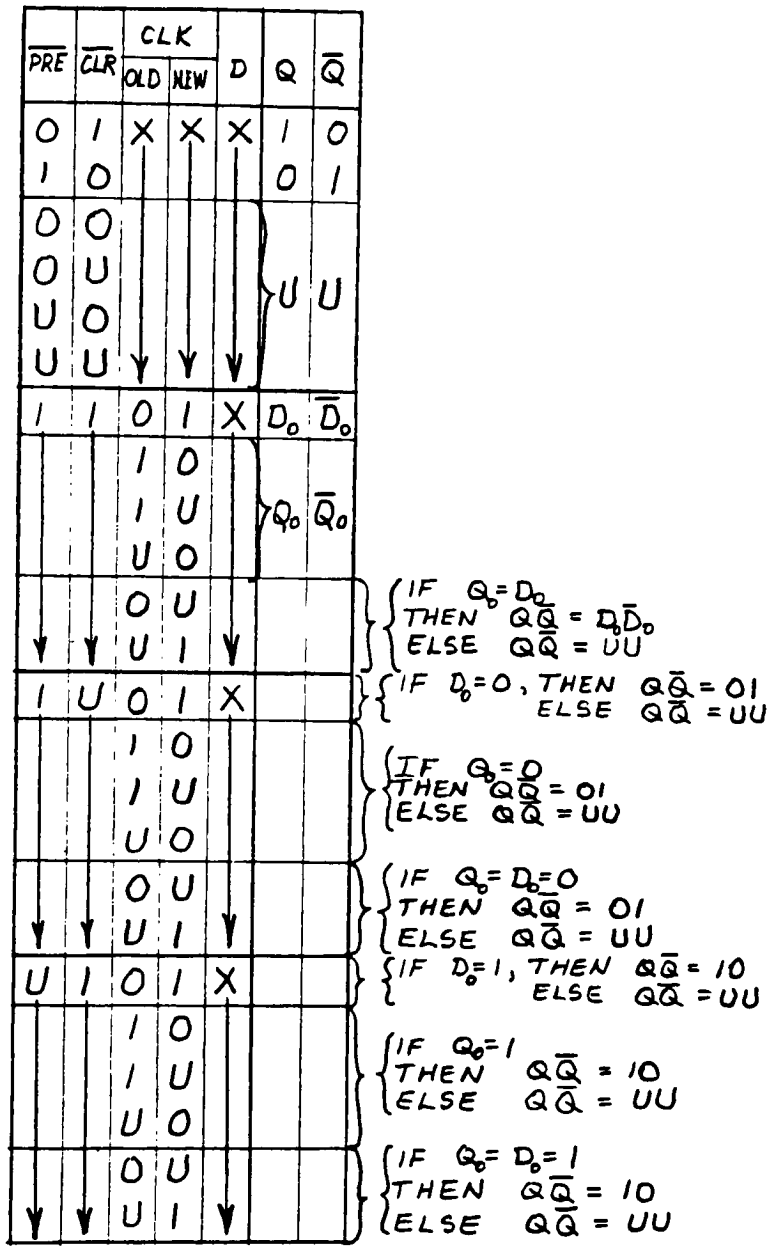


Fig. C. 7 Summary of the Truth Tables of 7474 d-type flip-flop for change in clock only.

REFERENCES

Ackland, Bryan D., Ahuja, Sudhir R., Lindstrom, Teri L., and Romero, Deborah J. 1985 "CEMU - A Concurrent Timing Simulator," IEEE 1985 International Conference on Computer-Aided Design," p.122.

Arnold, Jeffrey M., Terman, Christopher J. 1985 "A Multi-Processor Implementation of a Logic-level Timing Simulator," IEEE 1985 International Conference on Computer-Aided Design, p.116.

Bening, Lionel 1979 "Developments in Computer Simulation of Gate Level Physical Logic," Proceedings of the 1979 Design Automation Conference, p.561.

Breuer, Melvin A. 1972 Design Automation of Digital Systems, Volume 1, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Breuer, Melvin A. 1975 Digital System Design Automation: Languages, Simulation and Data Base, Computer Science Press, Inc., Woodland Hills, California.

Breuer, Melvin A., Friedman, Arthur D. 1976 Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Inc., Woodland Hills, California.

Calingaert, Peter 1979 Assemblers, Compilers and Program Translation, Computer Science Press, Inc., Potomac, Maryland.

Chawla, Basant R., Gummel Hermann K., Kozak, Paul 1975 "MOTIS - An MOS Timing Simulator," IEEE Transactions on Circuits and Systems, p.901.

Chen, C. F., Lo C-Y, Nham, H. N., Subramaniam, Prasad 1984 "The Second Generation MOTIS Mixed-Mode Simulator," Proceedings of the 1984 Design Automation Conference, p.10.

d'Abreu, Manuel A., 1985 "Gate-Level Simulation," IEEE Design and Test, p.63.

Grant, Floyd H. III, MacFarland, Douglas G. 1984 "Simulation With C", Proceedings of the 1984 Winter Simulation Conference, p.491.

Gonauser, M., Egger, F., Frantz, D. 1984 "SMILE - A Multilevel Simulation System,", IEEE 1984 International Conference on Computer Design: VLSI in Computers, p.188.

Hemming, Cliff W., Hemphill John M. 1975 "Digital Logic Simulation Models and Evolving Technology," Proceedings of the 1975 Design Automation Conference, p.85.

Henriksen, James O. 1983 "Event List Management - A Tutorial," Proceedings of the 1983 Winter Simulation Conference, p.543.

Henriksen, James O. 1984 "Discrete Event Simulation Languages - Current Status and Future Directions," Proceedings of the 1984 Winter Simulation Conference, p.83.

Howard, John K., Malm, Richard L., Warren, Larry M. 1983 "Introduction to the IBM Los Gatos Logic Simulation Machine," Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers, p.580.

Howie, H. Robert, Tavan, Richard M. 1971 "OLLS: The On-Line Logical Simulation System," Proceedings of the 1971 Design Automation Conference, p.314.

Ishiura, Nagisa, Uasuura, Hiroto, Kawata, Tetsuro, and Yajima, Shugo 1985 "High Speed Logic Simulation on a Vector Processor," IEEE International Conference on Computer- Aided Design, p.119.

Jaeschke, Rex 1986 Solutions in C, Addison-Wesley Publishing,Inc. Reading, Massachusetts.

Kearney, Michael A. 1984 "DECSIM: A Multi-level Simulation System for Digital Design," IEEE 1984 International Conference on Computer Design: VLSI in Computers, p.206.

Kernighan, Brian W., Ritchie, Dennis M. 1978 The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey.

Kochan, Stephen G. 1983 Programming in C, Hayden Book Company, Hasbrouck Heights, New Jersey.

Lathrop, Richard H., Kirk, Robert S. 1985 "An Extensible Object-Oriented Mixed-Mode Functional Simulation System," Proceedings of the 1985 Design Automation Conference, p.630.

Lewin, Douglas 1977 Computer-Aided Design of Digital Systems, Crane, Russak and Company, Inc., New York, New York.

Lieberherr, Karl J. 1984 "Towards a Standard Hardware Description Language,", Proceedings of the 1984 Design Automation Conference, p. 265.

Myers, Glenford J. 1979 The Art of Software Testing, John Wiley and Sons, Inc., New York, New York.

Nance, Richard E. 1983 "A Tutorial View of Simulation Model Development," Proceedings of the 1983 Winter Simulation Conference, p.325.

NETLIST and RNL; Tutorial for Beginners, UW/NWVLSI Consortium, University of Washington, Seattle, Washington.

Nham, H. N., Bose, A. K. 1980 "A Multiple Delay Simulator for MOS LSI Circuits," Proceedings of the 1980 Design Automation Conference, p.610.

Rubenstein, Charles 1983, "Computer-Aided Logic Design", Computers and Electronics, May 1983, p.68.

Schwarz, A. F., 1987, Computer-Aided Design of Microelectronic Circuits and Systems, Volume 2, Academic Press, Inc., Orlando, Florida.

Szygenda, S.A., Thompson E.W. 1975 "Digital Logic Simulation in a Time-Based, Table-Driven Environment, Part 1. Design Verification," Computer, p.24.

Terman, Christopher J. 1983 "RSIM - A Logic-Level Simulator," IEEE 1983 International Conference on Computer Design: VLSI in Computers, p.437.

TTL Data Book, Vol. 1 & 2 1985 Texas Instruments, Dallas, Texas.

Ulrich, Ernst G. 1969 "Exclusive Simulation of Activity in Digital Networks," Communications of the ACM, February, 1969, p.102.

Ulrich, Ernst G. 1976 "Non-Integral Event Timing for Digital Logic Simulation," Proceedings of the 1976 Design Automation Conference, p.61.

Ulrich, Ernst G. 1980 "Table Lookup Techniques for Fast and Flexible Digital Logic Simulation," Proceedings of the 1980 Design Automation Conference, p.560.

Vaucher, Jean G., Duval, Pierre 1975 "A Comparison of Simulation Event List Algorithms," Communications of the ACM., April 1975, p.223.

Whelan, Michael 1984 "Validation of Simulation Models by Testing against Formal Behavioral Specifications," IEEE 1984 International Conference on Computer Design: VLSI in Computers, p.201.

Wilcox, Phil 1979 "Digital Logic Simulation at the Gate and Functional Level," Proceedings of the Design Automation Conference, p.242.

Wilcox, P., Rombeek H. 1976 "F/Logic - An Interactive Fault and Logic Simulator for Digital Circuits," Proceedings of the Design Automation Conference, p.68.