

INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.
2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University
Microfilms
International

300 N. ZEEB ROAD, ANN ARBOR, MI 48106
18 BEDFORD ROW, LONDON WC1R 4EJ, ENGLAND

8016936

TULL, MONTE PAUL

A NEW METHOD FOR REALIZING PARALLEL PROCESSING MACHINES
USING MULTIPLE-VALUED LOGIC

The University of Oklahoma

PH.D.

1980

University
Microfilms
International

300 N. Zeeb Road, Ann Arbor, MI 48106

18 Bedford Row, London WC1R 4EJ, England

Copyright 1980

by

Tull, Monte Paul

All Rights Reserved

THE UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A NEW METHOD FOR REALIZING PARALLEL PROCESSING
MACHINES USING MULTIPLE-VALUED LOGIC

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

DOCTOR OF PHILOSOPHY

BY

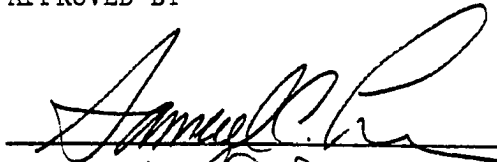

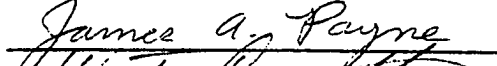
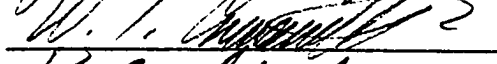

MONTE PAUL TULL

Norman, Oklahoma

1980

A NEW METHOD FOR REALIZING PARALLEL PROCESSING
MACHINES USING MULTIPLE-VALUED LOGIC

APPROVED BY

DISSERTATION COMMITTEE

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Dr. S. C. Lee whose invaluable support and guidance throughout the entire degree program helped to make it a very rewarding experience. Thanks are also due to the other members of the dissertation committee, Drs. W. T. Cronenwett, M. D. Devine, A. R. Magid, J. A. Payne, and K. L. Stanley, for their support and encouragement.

To my wife, Sandy, and children, Monte, Stacey, and Steffanie, whose devotion helped make this thesis a reality, I am forever grateful, and finally a special word of appreciation to Ms. La Vonne Lane Whitney for her untiring efforts, suggestions, and patience during the preparation of this dissertation.

TABLE OF CONTENTS

| | Page |
|---|------|
| ACKNOWLEDGEMENTS | iii |
| LIST OF TABLES | v |
| LIST OF FIGURES | vi |
| Chapter | |
| I. INTRODUCTION | 1 |
| II. CONSTRUCTING PARALLEL MACHINES USING MULTIPLE-VALUED LOGIC | 12 |
| III. PHYSICAL AND THEORETICAL LOGIC CIRCUIT CONCEPTS | 18 |
| IV. STATE INTEGRATED AND MULTIPLEXED COMBINATIONAL CIRCUITS | 59 |
| V. MEMORY ELEMENT DEVELOPMENT | 89 |
| VI. STATE INTEGRATED AND MULTIPLEXED SEQUENTIAL CIRCUITS | 109 |
| VII. CONCLUSION | 148 |
| LIST OF REFERENCES | 152 |
| APPENDIX | |
| A. SIMULATOR USER'S GUIDE | 156 |
| B. SIMULATOR SOURCE LISTING | 171 |

LIST OF TABLES

| TABLE | Page |
|---|------|
| 3.1. Gate and Signal Assumptions | 24 |
| 3.2. State Integrated and Multiplexed Logic System Arrangements | 25 |
| 3.3. B_4 and $B_2 \times B_2$ Element Summary | 29 |
| 4.1. Four-Valued Four-to-One Multiplexer Truth Table | 76 |
| 4.2. Selected Inputs and Expected Results for the State Integrated and Multiplexed Ripple-Carry Adder | 80 |
| 6.1. Four-Valued Register Transfer Combinations | 136 |
| 6.2. Simulated Transfers for the Four-Valued Register Transfer Circuit | 137 |
| A.1. QLOSIM Gate Types | 157 |
| A.2. Error Message Summary | 170 |

LIST OF FIGURES

| FIGURE | Page |
|--|------|
| 2.1. A Simple Processor Architecture | 13 |
| 2.2. Application of Encoder/Decoder Circuits | 15 |
| 2.3. State Integrated and Multiplexed Half Adder | 16 |
| 3.1. Multiple-Valued Functions for r = 4 | 19 |
| 3.2. Multiple-Valued Boolean Functions for r = 4 | 20 |
| 3.3. Binary Machine State Table | 32 |
| 3.4. TCM State Table | 33 |
| 3.5. Typical Gate Delay Model | 39 |
| 3.6. Circuit Topology Data Structure | 40 |
| 3.7. Time-Event Queue Structure | 42 |
| 3.8. Unary Gate Simulation Example | 44 |
| 3.9. Simulation Example Data Structure . . . | 44 |
| 3.10. Simulation Example Timing Diagram . . . | 46 |
| 3.11. Four-Valued Half Adder Circuit | 47 |
| 3.12. Four-Valued Half Adder Simulator Input Data | 49 |

| FIGURE | Page |
|---|------|
| 3.13. Four-Valued Half Adder Simulator | |
| Output | 49 |
| 3.14. Four-Valued RS r-Flop | 50 |
| 3.15. Next-State Table for Four-Valued | |
| RS r-Flop | 51 |
| 3.16. Four-Valued RS r-Flop Simulator | |
| Input Data | 51 |
| 3.17. Four-Valued RS r-Flop Simulator | |
| Output | 52 |
| 3.18. Circuit for Verification of | |
| $X \vee (X \dot{+} Y) = X \dot{+} Y$ | 54 |
| 3.19. Simulator Verification of | |
| $X \vee (X \dot{+} Y) = X \dot{+} Y$ | 55 |
| 3.20. I ² L AND Gate Realization | 57 |
| 3.21. I ² L OR Gate Realization | 58 |
| 4.1. B ₂ xB ₂ Representation of Boolean | |
| Functions | 63 |
| 4.2. B ₂ xB ₂ Circuit Containing | |
| Static-One-Hazard | 65 |
| 4.3. Simulation of B ₂ xB ₂ | |
| Static-One-Hazard | 66 |
| 4.4. Full Adder Karnaugh Maps | 70 |
| 4.5. Binary Full Adder Circuit | 71 |
| 4.6. Simulation of Four-Valued Multiplexed | |
| Adder | 73 |

| FIGURE | Page |
|--|------|
| 4.7. Four-Valued State Integrated and Multiplexed Four-to-One Multiplexer Circuit | 75 |
| 4.8. Four-Valued State Integrated and Multiplexed Four-to-One Multiplexer Simulation | 77 |
| 4.9. State Integrated and Multiplexed Four- Valued Ripple-Carry Adder | 79 |
| 4.10. Four-Valued State Integrated and Multiplexed Ripple-Carry Adder Simulation | 81 |
| 5.1. RS r-Flop Circuits | 91 |
| 5.2. Clocked Four-Valued r-Flops | 92 |
| 5.3. State Integrated RS SIM-Flop Next State Table | 93 |
| 5.4. Cross-Coupled NOR Gate RS SIM-Flop . . | 95 |
| 5.5. Simulation of NOR Gate RS SIM-Flop . . | 96 |
| 5.6. Modified RS SIM-Flop Circuit | 98 |
| 5.7. Simulation of Four-Valued RS SIM-Flop | 100 |
| 5.8. Detailed Simulation of Four-Valued RS SIM-Flop | 102 |
| 5.9. RS SIM-Flop Transient Behavior Simulation | 104 |
| 5.10. Four-Valued Clocked RS SIM-Flop | 106 |

| FIGURE | Page |
|--|------|
| 5.11. Four-Valued D-Type SIM-Flop | 106 |
| 5.12. Four-Valued D-Type SIM-Flop Next- State Table | 107 |
| 6.1. General Sequential Circuit Model . . . | 110 |
| 6.2. Four-Valued Flow Table | 112 |
| 6.3. Fundamental Mode Binary Flow Table . . | 113 |
| 6.4. State Adjacency Diagram | 114 |
| 6.5. Excitation Table | 115 |
| 6.6. Excitation and Output Karnaugh Maps . . | 116 |
| 6.7. Binary Circuit Realization | 117 |
| 6.8. Four-Valued TCM Realization | 117 |
| 6.9. TCM State Table | 118 |
| 6.10. TCM Input Transition Sequences | 119 |
| 6.11. TCM Subsequence Simulation for Figure 6.10(a) | 121 |
| 6.12. TCM Subsequence Simulation for Figure 6.10(b) | 123 |
| 6.13. TCM Subsequence Simulation for Figure 6.10(c) | 124 |
| 6.14. TCM Simulation Showing Oscillatory Condition | 125 |
| 6.15. TCM Clocked RS SIM-Flop Realization . . | 127 |
| 6.16. TCM Simulation Using Clocked RS SIM-Flops | 128 |

| FIGURE | Page |
|---|------|
| 6.17. TCM Simulation of Figure 6.10(c) Subsequence Using Controlled Four-Valued Clock | 130 |
| 6.18. TCM Simulation of Figure 6.10(a) Subsequence Using Controlled Four-Valued Clock | 132 |
| 6.19. Four-Valued State Integrated and Multiplexed Register Transfer Circuit | 134 |
| 6.20. Four-Valued Register Transfer Circuit Simulation | 140 |
| A.1 Example Circuit and Input Data | 160 |
| A.2 Illustration of a Complete Simulator Output | 165 |

A NEW METHOD FOR REALIZING PARALLEL PROCESSING
MACHINES USING MULTIPLE-VALUED LOGIC

CHAPTER I

INTRODUCTION

Parallel Processing Machines

The need to improve the speed of computation, often motivated by the desire to solve special classes of problems, has produced several computer processors that are capable of performing several operations simultaneously. The Illiac IV, STAR, CRAY-1, and ASC computers all rely on some degree of parallelism to achieve high performance. In general, a parallel processor is one that contains multiple arithmetic units and operates on multiple data streams [1]. The multiple hardware elements are often identical and must exhibit semi-independent concurrent operation. Overall, a parallel processor may or may not be capable of executing more than one machine instruction at a time. If not, the machine is referred to as a single instruction, multiple data stream (SIMD) processor. If more than one machine instruction can be executed concurrently, the machine is a multiple instruction, multiple data stream (MIMD) processor.

It is often difficult to classify parallel machines based strictly on concurrency of operation. For example, a form of parallelism is achieved in some machines by overlapping otherwise time sequential operations using a technique called pipelining. The method is often employed for machine or microinstruction fetch operations and allows the overlap of instruction execution with the fetching of the next instruction. Piping is also a feature of floating point arithmetic units and provides higher throughput rates. Pipelining, however, is not considered here as representative of the parallel processor concept since multiple identical or highly similar hardware elements capable of a variety of operations are not necessarily incorporated into a pipelined hardware design.

The main objective in constructing parallel machine hardware is to create an aggregate machine that can perform one of several operations concurrently in time. Specific types of problems that require the same operations to be applied to multiple quantities of data, i.e., matrix or vector manipulations, are particularly well suited as applications for parallel processors. In an SIMD machine each processor can execute the same instruction on different data segments. Each processor generates an intermediate result concurrently with the others. The overall problem solution time is therefore greatly reduced. Considerable effort may be involved to convert a problem into a form suitable for parallel processing, but this topic is not considered here. Specifically, we are interested in a new method that provides inherent hardware concurrency.

A New Approach

Currently, all parallel processors are realized with binary logic gates and storage devices. The identical elements of a machine that operate in parallel are simply multiple copies of the same binary logic circuit. Interconnection of the parallel elements also relies upon binary logic gate hardware.

A new method for simultaneous logic operation and, hence, the realization of parallel processors exists in the application of suitable multiple-valued logic gates and storage elements. Selection of a 2^m -valued logic allows the replacement of m binary machines with one multiple-valued machine. In the latter machine, the logic values of the individual binary machines exist simultaneously as multiple-valued logic levels. The multiple-valued storage elements (multi-stables) simultaneously store the information for all the m individual binary machines. The state of the individual machines need not be the same, and any one machine can change to a new state independently of the other machines. The technique is applicable to the entire range of typical hardware systems or subsystems, that is, from a simple circuit like an adder to an entire central processing unit (CPU). Both combinational and sequential circuits are necessarily included.

The encoding of several binary logic signals into a single system of multiple-valued logic levels is the basic premise of the idea being presented and represents, in a fashion, the multiplexing of several binary machines within a single multiple-valued machine. The scheme is somewhat analogous to frequency multiplexing in a communication system, whereby several conversations are simultaneously carried over

the same wire. The simultaneous storage of state information for two or more binary flip-flops by a single multi-stable suggests an appropriate descriptor, state integration. In other words, the states of the individual binary machines are integrated into a single composite state of the multiple-valued machine. A multiple-valued logic system organized to operate as two or more simultaneous lower radix machines will therefore be referred to as a state integrated and multiplexed digital system.

By simultaneously providing the logic functions of several machines within the circuitry of a single machine, the state integrated and multiplexed machine concept inherently exhibits the parallelism previously described. As an additional benefit the method significantly reduces the number of signal paths as compared to the multiple individual machines. The technique also produces a tightly coupled set of machines and potentially reduces the problem of control and information interchange between individual machines.

It should be clear that the individual machines within a state integrated and multiplexed digital system remain completely autonomous with respect to the other machines in both internal state and time. They are, however, only "logically" separate since they operate within the same circuit elements. The logical sub-machines form an inherently tightly coupled composite machine and can be arranged, via hardware and software techniques, to cooperate much the same as a parallel or multiple processor binary configuration. By focusing on parallel binary machines, the existing universe of knowledge dealing with binary systems is preserved and is especially important when viewing the work from a software perspective.

Research Objective

The state integrated and multiplexed digital system proposes a generalized use for multiple-valued logic that has not been previously investigated. This research will necessarily draw upon previous work in multiple-valued logic, but will differ in that earlier results have generally assumed that the multiple-valued logic elements operate in their respective natural radix. The work here will show that the multiple-valued logic devices, and circuits developed therefrom, can operate simultaneously as two or more lower radix devices and circuits. For example, a four-valued logic circuit can perform the same function as two identical binary logic circuits.

Since this work is somewhat new, there are several approaches for the research. First, one might assume--with little or no foundation--that the state integrated and multiplexed concept is feasible and consider various new architectures and organizations of automata that incorporate the essential features of the scheme. Certainly, this is justified to some extent, but a considerable amount of work must still be done to insure that the hardware is realizable; otherwise, the research may degenerate into a shuffling of "black boxes." As a second approach, the work could pursue electronic circuit development for logic devices. This is necessary since improvements in multiple-valued gates can lead to specific realizations, however restricted they might be. Neither of these two extreme approaches will be used, rather an overall justification of the concept will be based on some theoretical foundations followed by analysis of state integrated and multiplexed logic circuits. The design and analysis of both combinational and sequential

logic circuits will be considered. Specific realizations for logic gates are included in this work, but the primary emphasis is on logic circuit development and behavior analysis. Whenever needed, worst case operating conditions in the multiple-valued logic are assumed. It is anticipated that these assumptions will accommodate one or more actual realizations for the logic gates.

The research will attempt to justify the state integrated and multiplexed conjecture by showing what elements comprise the logic circuits and, further, investigate the response of meaningful and useful circuits. It is not reasonable to show results for all possible state integrated and multiplexed machines, but since a great deal more is known about binary logic than, say, three-valued or ternary logic, the emphasis is centered on replacing m , $m \geq 2$, parallel binary machines with one multiple-valued machine having a radix $r = 2^m$. In many of the specific circuit examples, the value of $m = 2$ is selected in order to keep the circuit development and analysis as simple as possible.

It will be shown that previously used multiple-valued storage devices will not suffice logically for a state integrated storage element and some work will be directed toward the development of a suitable multi-stable. Without this new device, state integrated sequential logic circuits are not easily realizable.

When the foregoing results are satisfactorily obtained, state integrated and multiplexed digital system design will become a new method for realizing two or more identical digital machines. The concept underlies and adds impetus to the construction of a true dual radix machine or processor. Such a machine would be capable of operating

in the state integrated and multiplexed mode as two or more identical processors, or it could be switched to function as a single processor operating in the natural radix of the multiple-valued logic employed. Therefore, this work hopes to solve half of the problem by showing that multiple-valued logic elements will support the replacement of two or more binary machines. Future follow-on research may be able to show detailed results for either the case of non-binary sub-machines or the dual radix processor concept.

Previous Work

Parallel processing logic has been used to increase the speed of several computer systems [1,2]. These large parallel processors are among the most powerful machines available today. All of these machines are realized by the application of binary logic elements. We are concerned here with an alternative realization method for parallel machines that uses multiple-valued logic elements. Successful manufacture of multiple-valued logic devices has been achieved using Integrated Injection Logic or I^2L [3,4]. Smith [5] has presented an overview of various electronic implementations for multiple-valued logic devices.

Much of the work in multiple-valued logic systems has involved devising algebraic structures. The Post algebras [6] have been widely used and form the basis for several combinational and sequential circuit design methods. The minimization of Post based multiple-valued switching functions has been the work of several authors [7-9]. Combinational design methods similar to those in a binary system have been proposed by Allen [10]. From experience gained from actual I^2L circuit fabrication, McCluskey [11,12] has proposed a new combinational design technique.

His work allows a rich set of primitive functions, some of which occur naturally in the current mode I^2L circuits.

Sequential circuit design for multiple-valued systems relies on the combinational work but also includes exploration of memory elements and the interaction of circuit components that result in hazard and race conditions. Wojcik [13,14] and Sheafor [15] have examined the problem of asynchronous sequential circuit design. Both studies considered the memory element requirement and the need for combinational circuits that are free of hazard conditions. Specific effort for the development of memory elements is found in [16-19]. In particular, Wills [19] has prescribed a concise model for the behavior of multiple-valued memory elements. Race conditions do not pose as great a problem as do hazards, but the subject has been considered by Moraga [20] for three-valued or ternary logic systems.

Boolean based multiple-valued systems have received attention primarily at the algebraic level and little consideration has been given for serious design of logic devices. Wojcik [21] has shown relationships between Post and Boolean single variable functions. Wojcik and Metze [22] have provided a method based on these relationships that allows minimization of higher-order Boolean functions after the function is mapped to a corresponding Post function.

One advantage in designing with higher-order Boolean based logic devices is the possibility of extending existing techniques of the B_2 binary Boolean system. The necessary mathematical basis for the B_2 and higher-ordered systems is given by Lee [23]. When the multiple-valued digital system operates in the natural or higher radix mode, an

important shortcoming of higher-ordered Boolean algebras is the fact that the AND, OR and COMPLEMENT functions do not comprise a functionally complete system for multiple-valued logic. For example, in the combinatorial sense, there are 256 functions of one variable in a four-valued logic system; however, there exist only 16 Boolean functions of one variable [21]. Without the inclusion of additional functions, the Boolean system consisting of AND, OR, and COMPLEMENT will not suffice for the general multiple-valued logic system. On the other hand, if the multiple-valued system represents a set of Boolean systems operating in parallel, as is the case for the state integrated and multiplexed digital system, then the Boolean algebra is functionally complete.

The evaluation of digital circuit designs for proper operation is often performed for binary systems through the use of circuit simulation computer programs [24]. Programs of this type are often categorized as design automation tools. Digital simulators may be of the gate-level type [25], or they may be structured at a higher or functional level [26]. A functional simulator is convenient for describing large systems, such as an entire computer, prior to the selection of circuit elements. With both types of simulators, the designer is able to explore the interaction of the pertinent variables and functions that comprise the digital system.

With multiple-valued digital systems the variables and functions become correspondingly more complex, and the need for a simulation tool cannot be overlooked. Circuit designs consisting of only a few gates can be exceedingly difficult to analyze by hand. The inclusion of new and unfamiliar multiple-valued gate functions can also confuse the task

of circuit evaluation if performed manually. Moraga [27] has reported a ternary gate-level simulator and in [28] proposes the model for an enhanced version of the simulator. A note of caution to prevent unnecessary confusion is that a true multiple-valued logic element simulator is being considered here and not binary logic simulators that may employ so-called multiple-valued techniques to depict dynamic binary logic phenomena such as 0-1 transitions [24].

Thesis Summary

This thesis is basically concerned with the investigation of logic design requirements for state integrated and multiplexed digital systems. Such systems, it is claimed by way of conjecture, offer a new method for realizing parallel processing machines. The methods of designing combinational and sequential logic circuits and memory elements will be analyzed. Specific results are limited to replacing two or more parallel binary machines with one multiple-valued logic machine. Meaningful circuit examples are used throughout, and actual results of circuit behavior are derived by using a logic circuit simulator.

Chapter II discusses the basic motivation for the state integrated and multiplexed digital system by first showing how an actual computer CPU can be realized in whole or in part to yield two or more CPUs or elements thereof. Chapter III presents the notation and circuit operating concepts to be applied in the analysis of logic circuits. Assumptions regarding delays and device signal responses are given followed by a discussion of pertinent theoretic work. Multiple-valued logic circuit simulation, to be used as a tool for evaluating logic

circuit behavior, is also presented. Finally, the realization of pertinent multiple-valued logic gates is given.

Logic circuit design is the topic of the next three chapters. Chapter IV presents the combinational circuit design philosophy and gives results for simulated examples. Chapters V and VI are concerned with sequential logic circuit design. In Chapter V the logical design of a new memory element is given. The device is shown to be able to simultaneously store the state or information for two or more binary machines. Several variations of the basic memory device are developed and allow both synchronous and asynchronous operation, depending on the device chosen. Chapter VI uses the results of the previous two chapters and develops the sequential logic circuit design process. Again, example circuits are shown along with simulated results.

Two appendices are included to provide user information and source listing for the multiple-valued logic simulator. Appendix A is a user's guide to the simulator, giving the capability range of the software and the means for supplying input data and execution control. Appendix B contains the software source listing.

CHAPTER II

CONSTRUCTION OF PARALLEL MACHINES

USING MULTIPLE-VALUED LOGIC

Digital System Components and Circuit Types

The construction of some parallel processors, such as Illiac IV [2], is centered about the application of several small binary processors operating in parallel to achieve an overall high rate of throughput. Without being overly concerned with a specific parallel processor architecture, consider a small single CPU architecture such as Intel 8080/8085 microprocessor [29]. A simplified block diagram of this processor is given in Figure 2.1. Note that the machine has several general parts. There is an arithmetic and logic unit (ALU) used for addition and logic operations. The internal registers comprise another major portion of the CPU. Several registers are used for data manipulation and storage. Register A is the accumulator and generally supplies an argument for arithmetic and logic functions, and the result of these operations is stored back in the accumulator as well. Other registers, such as the program counter (PC) and stack pointer (SP) have special uses. The last major part of the system is the control unit

which decodes instructions and controls their execution. In general, all of these operations are synchronized by the clock.

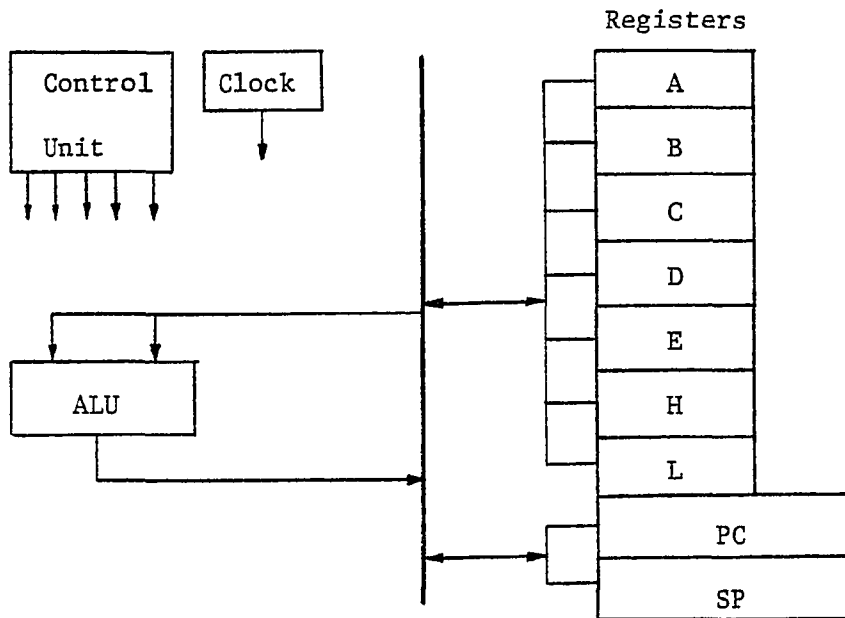


Figure 2.1. A Simple Processor Architecture

Using this architecture as a model, consider the types of binary logic circuits that are used to construct it. First, the ALU can be built as a combinational logic circuit which functions under the supervision of the control unit. The control unit, on the other hand, is basically a sequential circuit. The internal registers attach to an internal bus, and together these elements comprise a register transfer circuit which, for purposes here, will be considered a special type of sequential circuit. These three circuit types are present in practically every central processing unit.

To replicate such a binary processor, in whole or in part, in order to generate a multiprocessor or parallel processor architecture requires the external connection of two or more such machines. Now consider the use of multiple-valued logic devices for the construction

of an ALU. Applying the state integrated and multiplexing scheme using eight-valued logic functions, three such binary processors could be constructed to operate simultaneously within the multiple-valued logic. Further, the three processors could each operate independently of the others; that is, they could execute different instructions and contain different internal data and states.

To accomplish the construction of the multiple-valued processor, the logic functions must be identified, and the logic circuit design methods for the three circuit types mentioned previously must also be examined. Finding suitable methods will allow the specification of digital systems based upon the state integrated and multiplexed approach.

The Encoder/Decoder Interface

Supposing that a multiple-valued logic state integrated and multiplexed system can be built that will replace two or more identical processors, a possible problem exists for interfacing the system to existing external binary components. For example, it may be advantageous to use high density binary memories for the primary storage unit. An encoder/decoder interface is necessary to encode binary signals into multiple-valued signals, and a decoder is needed to convert in the reverse direction. Edwards [30] has presented I^2L circuits that encode and decode binary to four-valued signals, and this ability is exactly what is needed for conversion between binary components and a four-valued state integrated and multiplexed processor.

The encoder/decoder circuits are applied on all external processor signal paths that exit the processor and connect to binary

components. Normally, this group includes all bus signals that communicate with external binary devices. For the processor architecture given above, Figure 2.2 shows how the encoders and decoders are applied. To simplify the diagram, not all external signals are shown. For this

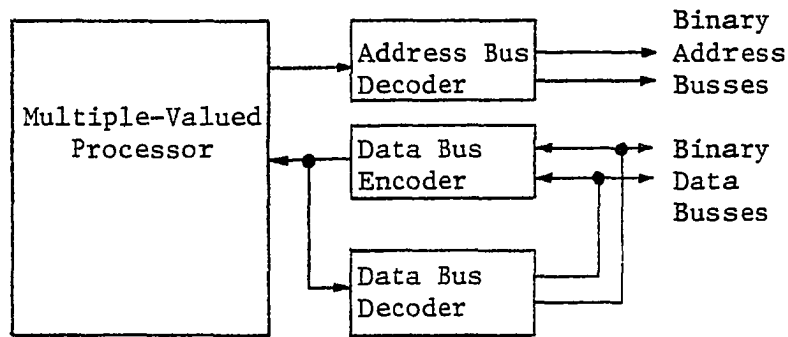


Figure 2.2. Application of Encoder/Decoder Circuits

particular machine architecture the address bus is one-way for the CPU, and no encoder is required. However, the data bus requires both an encoder and a decoder. Note that on the righthand side of the encoder/decoder circuits, the busses are binary and each signal path carries one binary signal. These binary signals could be connected to separate binary memory systems, each containing a different software program and data.

A Simple Circuit Example

To demonstrate the state integrated and multiplexed concept at a finer level, a simple circuit example will be given. The example is also useful for showing that the concept can be applied at various functional levels, i.e., an entire processor need not be designed. The detailed circuit behavior is not discussed at this time and is deferred until additional material is presented; however, this specific circuit example will help clarify the ideas being discussed.

Figure 2.3 shows a combinational half adder circuit. Instead of being binary, the logic gates should be considered as four-valued. Binary to quaternary encoders and decoders are shown on the circuit inputs and outputs, respectively. All signal paths between the encoders and decoders carry four-valued logic signals. The logic gates are Boolean AND, OR, and COMPLEMENT gates.

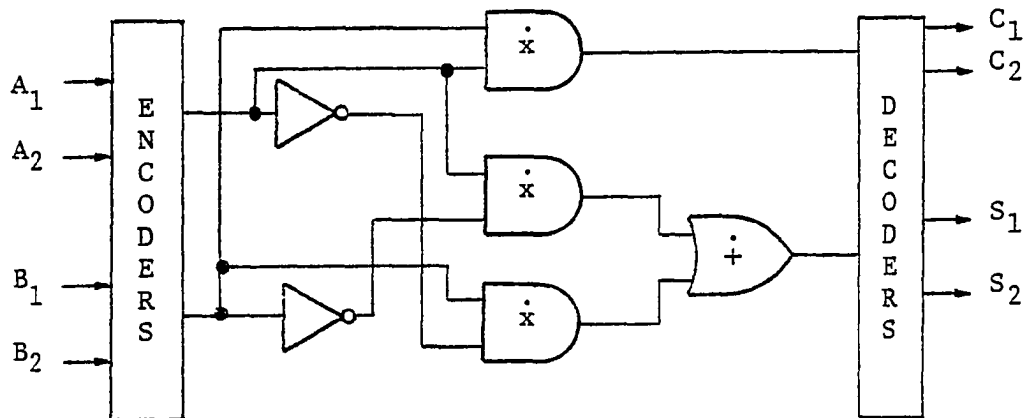


Figure 2.3. State Integrated and Multiplexed Half Adder

The circuit functions by encoding pairs of binary input signals A_1 , A_2 , and B_1 , B_2 into four-valued signals, summing these four-valued inputs, and generating the sum and carry-out. The four-valued sum is decoded into binary signals S_1 and S_2 . Likewise, the carry-out is decoded into C_1 and C_2 . In explicit terms, the A_1 and B_1 binary values are summed together and, simultaneously, the A_2 and B_2 binary inputs are summed together. The circuit, therefore, operates as if it were two binary half adders operating in parallel.

Circuits of this type are important when the function being performed is needed concurrently in time by two or more processes. Such could be the case for an adder in a CPU since the adder may be used to increment (add one to) a register such as the PC as well as perform a

normal addition operation resulting from the execution of an ADD instruction. This situation occurs so frequently that some binary CPUs possess a separate arithmetic unit to perform address calculations concurrently and separately from the normal arithmetic unit. With a state integrated and multiplexed arithmetic unit incorporated into a binary CPU, two or more arithmetic circuits are inherently provided. The penalty of encoding and decoding must be paid to achieve this capability. One advantage gained, however, is that fewer signal paths are required.

The overall goal of the thesis is to extend the state integrated and multiplexing concept to include more than arithmetic elements, although in practice this type of circuit may be a good place to initially apply the scheme. The remaining chapters will develop the three main circuit types and analyze circuit behavior so that the state integrated and multiplexed technique is established for a broad range of applications.

CHAPTER III

PHYSICAL AND THEORETICAL LOGIC CIRCUIT CONCEPTS

Notation

For an r -valued digital system denote the r distinct logic values as $0, 1, 2, \dots, r-1$. These numerical elements represent the values that a logic variable may assume. A functionally complete Post algebra [6] with zero element 0 and universal element $r-1$ consists of the set of elements $S_r = \{0, 1, \dots, r-1\}$, and the operations defined as

$$\text{MIN: } X_1 \wedge X_2 = \text{MIN} (X_1, X_2)$$

$$\text{MAX: } X_1 \vee X_2 = \text{MAX} (X_1, X_2)$$

$$\text{CYCLE: } X_1 \stackrel{\rightarrow a}{=} X_1 + a \pmod{r}.$$

The strong negation unary function is defined as

$$\bar{X}_1 = (r-1) - X_1.$$

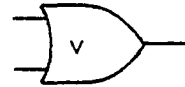
This function is often considered for implementable switching algebras but cannot be used solely to replace the CYCLE operation without sacrificing the functional completeness property. Functional completeness necessitates the ability to express r^r functions of a single variable. Figure 3.1 shows the function maps, truth tables and associated logic symbols for the foregoing functions with $r = 4$.

| | | | | | |
|-------|---|-------|---|---|---|
| | | X_2 | | | |
| | | 0 | 1 | 2 | 3 |
| X_1 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 1 | 1 | 1 |
| | 2 | 0 | 1 | 2 | 2 |
| | 3 | 0 | 1 | 2 | 3 |



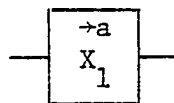
(a) MIN Function

| | | | | | |
|-------|---|-------|---|---|---|
| | | X_2 | | | |
| | | 0 | 1 | 2 | 3 |
| X_1 | 0 | 0 | 1 | 2 | 3 |
| | 1 | 1 | 1 | 2 | 3 |
| | 2 | 2 | 2 | 2 | 3 |
| | 3 | 3 | 3 | 3 | 3 |



(b) MAX Function

| | | | | |
|-------|---|-----|-----|-----|
| | | a=1 | a=2 | a=3 |
| X_1 | 0 | 1 | 2 | 3 |
| | 1 | 2 | 3 | 0 |
| | 2 | 3 | 0 | 1 |
| | 3 | 0 | 1 | 2 |



(c) Cycle Function

| | |
|-------|-------------|
| X_1 | \bar{X}_1 |
| 0 | 3 |
| 1 | 2 |
| 2 | 1 |
| 3 | 0 |



(d) Strong Negation

Figure 3.1. Multiple-Valued Functions for $r = 4$.

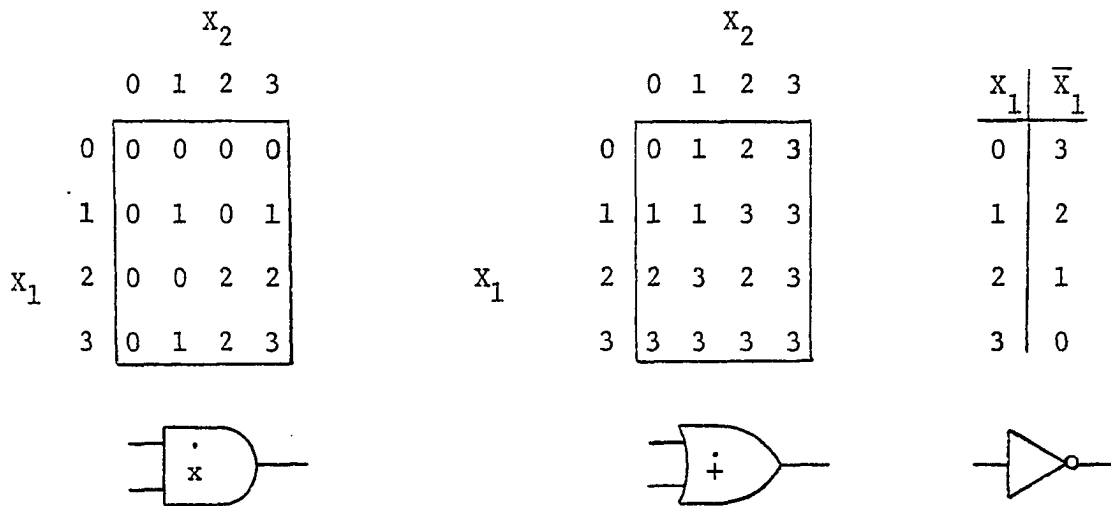
A functionally complete higher-order Boolean algebra with elements having cardinality $r = 2^m$, m an integer ≥ 1 , is formed by the operations defined as

$$\text{AND: } X_1 \dot{x} X_2$$

$$\text{OR: } X_1 \dot{+} X_2$$

$$\text{COMPLEMENT: } \bar{X}_1 = (r-1) - X_1.$$

The symbols $(\dot{x}, \dot{+})$ are chosen to depict the multiple-valued nature of these operations and prevent confusion with the binary system when examining logic functions and gates. For the binary case ($r = 2$), the familiar $(\cdot, +)$ will be used for the AND and OR operations, respectively. The function maps and logic symbols for the four-valued Boolean system are given in Figure 3.2. Binary AND gates and OR gates will be shown without the internal symbols.



(a) AND Function

(b) OR Function

(c) COMPLEMENT

Figure 3.2 Multiple-Valued Boolean Functions for $r = 4$.

Behavioral Concepts

As defined in the previous section, the r values associated with an r -valued system are denoted by $S_r = \{0, 1, \dots, r-1\}$. Signal or variable transitions within the r values can be ordered by three methods [13]. For $0 \leq 1 \leq 2 \leq \dots \leq r-1$ and a variable transition from a value i to a value j , $i, j \in R$, the ordering is linear if the variable temporarily assumes all values between i and j . If $0 \leq 1 \leq 2 \leq \dots \leq r-1$ and the transitions from $r-1$ to 0 and 0 to $r-1$ can be made directly, the ordering is cyclic. If it is possible to transfer directly from any value i to any value j , the ordering is complete.

Physical representation of multiple-valued logic levels could be implemented by several methods. Multiple frequency signals in electronic devices or multiple color (frequency) methods in electromagnetic radiation are two possible schemes. By far, the most common method encountered in present-day technology will be an electronic voltage or current representation of the logic values. In the voltage case, the voltage levels $[v_0, v_1, v_2, \dots, v_{r-1}]$ have a linear ordering, $v_0 \leq v_1 \leq v_2 \leq \dots \leq v_{r-1}$. The correspondence between these levels and the logic values assigns voltage level v_i to logic value i . Thus, the voltage scheme requires the logic values to be linearly ordered. A similar argument exists for the current representation method. Since both voltage and current methods presently exist and since the assumption of linearly ordered variables is a worst case condition, it is assumed in this thesis that all variables will take on linearly ordered values.

The linear ordering assumption places constraints on the transitions that can occur on signal lines of a multiple-valued circuit.

Each logic gate in the circuit produces an output that may change from i to j . Such a change, which results from one or more input transitions, is constrained to reflect all values in linear order between i and j . Each value will exist on the line for some arbitrary finite length of time.

The logic device reaction to changes on an input line is also an important consideration. If the input value makes a transition from i to j , the gate may or may not react to intermediate logic values. If the gate's inertial delay, Δ_I , is exceeded by an input signal, the gate will react to the logic value. It could be the case that the device reacts to some intermediate values but not others. Perhaps a worst case condition, which is the one assumed here, is that the gate reacts to all intermediate values. Depending on the gate type, some or all of these reactions may cause a change in the output of the device. This assumption thus causes $\Delta_I = 0$.

In all cases it is assumed that the gate exhibits a transport delay of finite time. For transport delay Δ_T , an input change at time t_1 that causes an output change will necessitate the change on the gate output at time t_2 , where $t_2 = t_1 + \Delta_T$. It should be pointed out that the output begins to change at t_2 . If the change is a multiple-level transition, each intermediate value will be reflected on the output in linear order at some time following t_2 .

To accurately specify the transition time, consider a signal changing from logic value i to logic value j . Let Δ_R and Δ_F be the rise and fall time, respectively, between adjacent logic levels. The following discrete rise/fall time procedure is used to calculate the time

at which logic levels will be observed on a line that is changing value beginning at t_2 :

- 1) If $|i-j| = 1$, the change is reflected at t_2 .
- 2) If $i-j > 1$, the signal is falling. The logic value $i-1$ is reflected at t_2 . Logic levels $i-2, i-3, \dots, j$ are observed respectively at $t_2 + \Delta_F, t_2 + 2\Delta_F, \dots, t_2 + n\Delta_F$, where $n = (i-j) - 1$.
- 3) If $j-i > 1$, the signal is rising. The logic value $i+1$ is reflected at time t_2 . Logic levels $i+2, i+3, \dots, j$ are observed respectively at $t_2 + \Delta_R, t_2 + 2\Delta_R, \dots, t_2 + n\Delta_R$, where $n = (j-i) - 1$.

The foregoing procedure thus allows the first adjacent logic value to be observed immediately following the gate transport delay. Succeeding logic transitions, if any, are delayed by the rise or fall time delay. Now, with some loss of generality, it is assumed that $\Delta_R = \Delta_F$ for all signals regardless of source.

Table 3.1 briefly summarizes the gate and signal delay assumptions made for purposes of this thesis. These operating conditions place as few restrictions as possible on physical devices so that circuits, which might be implemented in the future by any of several methods, can be studied prior to actual construction. In particular, it is felt that the discrete rise/fall time assumption is a valid approximation for the study of circuit behavior. This is especially true when a unit time circuit simulator is used, wherein the designer can specify the various gate delay times. The rise/fall time can thus be scaled relative to the transport delays as necessary to more accurately reflect actual circuit operation.

Table 3.1. Gate and Signal Assumptions

| CHARACTERISTIC | ASSUMPTION | REMARKS |
|----------------------|---------------------------|--|
| 1. Signal Transition | Linearly Ordered | |
| 2. Transport Delay | $\Delta_T > 0$ | Defined for each gate in the circuit; constant, i.e., non-ambiguous. |
| 3. Inertial Delay | $\Delta_I = 0$ | Gates react to all logic levels. |
| 4. Rise/Fall Delay | $\Delta_R = \Delta_F > 0$ | Constant throughout the circuit. |

Theoretical Concepts

Consider an r -valued logic system and let $r = 2^m$, $m \geq 2$. Using such a multiple-valued system, previous research has focused on the design of digital systems that operate in the natural radix of the given logic system. In contrast, it is suggested here that the r -valued logic system, with $r = 2^m$, can simultaneously support m binary systems and, hence, m binary digital machines. Thus, for $m = 2$, two binary systems simultaneously exist in the four-valued logic. "Simultaneously" implies that the four-valued or quaternary logic level signals and memory contents represent the information flow and internal state of two independent binary machines. Since the information for the machines is multiplexed via the logic levels and since the memory elements simultaneously store the state of both machines, the system is termed a state integrated and multiplexed digital system.

It is possible to generalize these concepts to higher logic levels and radii. In general, for $r = q^m$, we can speculate that there exist m lower radix machines, each having a radix q . Utilizing the modeling of lower radix logic in a higher radix logic [33], even more

general statements are possible, but due to the complications introduced by this extension, our attention is restricted to radix q systems, $r = q^m$, and in particular $r = 2^m$. Table 3.2 depicts several multiple-valued logic systems and the corresponding radix q systems supported by each.

Table 3.2. State Integrated and Multiplexed Logic System Arrangements

| <u>LOGIC LEVELS (r)</u> | <u>RADIX (q)</u> | <u>RADIX q SYSTEM QUANTITY (m)</u> |
|-------------------------|------------------|------------------------------------|
| 4 | 2 | 2 |
| 8 | 2 | 3 |
| 9 | 3 | 2 |
| 16 | 2 | 4 |
| 16 | 4 | 2 |
| 25 | 5 | 2 |
| 27 | 3 | 3 |
| 32 | 2 | 5 |
| 36 | 6 | 2 |

As stated above, the particular systems of concern possess a multiple-valued radix with $r = 2^m$. The algebras of interest are the 2^m -valued Boolean $B_2 \times B_2 \times \dots \times B_2$ and B_{2^m} systems. From an implementation viewpoint it is acknowledged that the Post algebras must also be considered since the specification of multiple-valued AND, OR and COMPLEMENT gates is possible in the functionally complete Post systems. Thus, the Post algebras can provide the building blocks for the other multiple-valued Boolean systems. Four-valued Boolean AND and OR gate realizations, which have a basis in the Post algebra operations, are presented later in this chapter.

To explore the logical structure and logic circuit behavior of the state integrated and multiplexed digital system requires the properties

of the higher ordered Boolean algebras whenever binary sub-machines are assumed. In the following discussion four-valued systems, $r = 2^2$, are used without significant loss of generality. This case is emphasized since it clearly exemplifies and supports the state integrated and multiplexed property, and it would likely be the first radix chosen for implementation. Extension of the concepts to larger values of m is possible and fairly straightforward.

The B_4 Boolean system is an ordered set, $S(B_4) = \{0,1,2,3\}$, on which the closed operations AND (\dot{x}) and OR ($\dot{+}$) are defined. The following postulates hold for B_4 :

P1 The operations \dot{x} and $\dot{+}$ are commutative; that is,
for each pair of elements a and b in $S(B_4)$,

$$a \dot{+} b = b \dot{+} a$$

and
$$a \dot{x} b = b \dot{x} a.$$

P2 Each operation \dot{x} and $\dot{+}$ is distributive over the other; that is, for any three elements a , b , and c in $S(B_4)$,

$$a \dot{+} (b \dot{x} c) = (a \dot{+} b) \dot{x} (a \dot{+} c)$$

$$a \dot{x} (b \dot{+} c) = (a \dot{x} b) \dot{+} (a \dot{x} c).$$

P3 There exist in $S(B_4)$ distinct identity elements, denoted here as 0 and 3, relative to the operations $\dot{+}$ and \dot{x} , respectively; that is, for every element b in $S(B_4)$,

$$0 \dot{+} b = b \dot{+} 0 = b$$

$$3 \dot{x} b = b \dot{x} 3 = b.$$

P4 For every element b in $S(B_4)$ there exists an element* b' in $S(B_4)$ such that

$$b \dot{+} b' = 3$$

$$b \dot{x} b' = 0$$

where 0 and 3 are the identity elements of Postulate 3.

The idempotent, associative, absorbtive, and involution properties hold for B_4 [23]. These are summarized below:

Idempotent: $a \dot{x} a = a$; $a \dot{+} a = a$

Associative: $a \dot{x} (b \dot{x} a) = (a \dot{x} b) \dot{x} a$; $a \dot{+} (b \dot{+} a) = (a \dot{+} b) \dot{+} a$

Absorption: $a \dot{x} (a \dot{+} b) = a$; $a \dot{+} (a \dot{x} b) = a$

Involution: $(a')' = a$

DeMorgan's Theorem also holds for B_4 [23] and, for each pair of elements, a and b , in $S(B_4)$, can be stated algebraically as

$$(a \dot{x} b)' = a' \dot{+} b'$$

$$(a \dot{+} b)' = a' \dot{x} b'.$$

Another important result is that the identity elements of B_4 , 0 and 3, are complements of each other. The proof is the same as for B_2 [36].

A second four-valued Boolean algebra is the $B_2 \times B_2$ system formed by the Cartesian product (\times) of two binary Boolean algebras. The elements of $B_2 \times B_2$ are the set $S(B_2 \times B_2) = \{00, 01, 10, 11\}$. The universal elements of this system are the additive identity (00) and the multiplicative identity (11). The isomorphism of B_4 with $B_2 \times B_2$ is given with the following theorem.

*The prime (') and bar (-) symbols will be used interchangeably to denote complementation.

Theorem 3.1. All four-valued Boolean algebras are isomorphic.

Proof: Let C and D be generalized four-valued Boolean algebras.

To construct the elements of C , first note that C must contain two distinct universal elements. Call these 0_C and 1_C . Let $c \in S(C) \neq 0_C, 1_C$. The complement of c , written c' , must also be an element of C . To show c' is another distinct element, consider the following. If $c' = 0_C$, then $c = (c')' = 0_C' = 1_C$, which is a contradiction. If $c' = 1_C$, then $c = (c')' = 1_C' = 0_C$, also a contradiction. If $c' = c$, then the property $c \dot{+} c' = 1_C$ implies that $c \dot{+} c = 1_C$. For this to be true $c = 1_C$. Therefore, $c \neq c'$ and c' is the fourth element of C . The elements of C are thus established, and $S(C) = \{0_C, 1_C, c, c'\}$. In a similar manner the four-valued Boolean system D can be constructed with $S(D) = \{0_D, 1_D, d, d'\}$.

Consider the mapping $\emptyset: C \rightarrow D$, where

$$\emptyset(0_C) = 0_D$$

$$\emptyset(1_C) = 1_D$$

$$\emptyset(c) = d$$

$$\emptyset(c') = d'.$$

The mapping is obviously one-one and onto. For elements $w, x \in C$ and $y, z \in D$, such that $\emptyset(w) = y$ and $\emptyset(x) = z$,

$$\emptyset(w \dot{+} x) = \emptyset(w) \dot{+} \emptyset(x) \tag{1}$$

$$\text{and} \quad \emptyset(w \dot{\times} x) = \emptyset(w) \dot{\times} \emptyset(x) . \tag{2}$$

Equation (1) holds by noting that

$$0_C \dot{+} x = x \quad 0_D \dot{+} z = z$$

$$1_C \dot{+} x = 1_C \quad 1_D \dot{+} z = 1_D$$

$$\text{and} \quad x \dot{+} x' = 1_C \quad z \dot{+} z' = 1_D .$$

Equation (2) holds by similar argument,

$$\begin{array}{ll} 0_C \dot{x} x = 0_C & 0_D \dot{x} z = 0_D \\ 1_C \dot{x} x = x & 1_D \dot{x} z = z \\ x \dot{x} x' = 0_C & z \dot{x} z' = 0_D \end{array}$$

Thus, \emptyset is a mapping and the isomorphism is established. QED

This theorem allows the statement that $B_2 \times B_2$ is isomorphic to B_4 . Table 3.3 summarizes both systems and their respective elements. Because of the established equivalence between B_4 and $B_2 \times B_2$, the use or referral to one system or the other is simply a matter of convenience in notation. For example, when denoting signal values and logic truth tables, $S(B_4) = \{0,1,2,3\}$ is possibly more convenient, but when discussing or visualizing the state integrated and multiplexing concept for actual variables, the notation using $S(B_2 \times B_2) = \{00,01,10,11\}$ is more meaningful.

Table 3.3. B_4 and $B_2 \times B_2$ Element Summary

| <u>B_4 Element</u> | <u>$B_2 \times B_2$ Element</u> | <u>B_4 Complement</u> | <u>$B_2 \times B_2$ Complement</u> |
|---------------------------------|--|------------------------------------|---|
| 0 | 00 | 3 | 11 |
| 1 | 01 | 2 | 10 |
| 2 | 10 | 1 | 01 |
| 3 | 11 | 0 | 11 |

For work that appears in later chapters, it will be necessary to briefly present some properties of the Post algebras. Of particular concern are the postulates and theorems that enable algebraic manipulation of switching functions. Rather than state these formally, we will simply note that the P_4 algebra with elements $S(P_4) = \{0,1,2,3\}$ and operations MIN, MAX, CYCLE and strong negation have the following properties for general elements x , y , and z [34]:

Commutative: $x \wedge y = y \wedge x$; $x \vee y = y \vee x$

Associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$; $x \vee (y \vee z) = (x \vee y) \vee z$

Distributive: $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$;

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Idempotent: $x \wedge x = x$; $x \vee x = x$

Absorption: $x \wedge (x \vee y) = x$; $x \vee (x \wedge y) = x$

Involution: $(x')' = x$

Universal

Elements: $x \vee 0 = x$; $x \wedge 3 = x$

In addition, DeMorgan's Theorem holds for P_4 [34] and can be stated as:

$$(x \wedge y)' = x' \vee y'$$

$$(x \vee y)' = x' \wedge y' .$$

Note that

$$x \wedge x' = 0$$

and

$$x \vee x' = 3$$

do not hold for the P_4 algebra.

Throughout the remainder of this thesis $B_2 \times B_2$ is the primary algebra employed for specific circuit examples, even though B_4 elements may be cited. This should not cause undue confusion if the equivalence of the two systems is kept in mind. To further aid the discussion, by logically separating the sub-machines in the four-valued state integrated and multiplexed circuits, the notation of a left machine, M_2^L , and a right machine, M_2^R , is utilized. The superscripts, L and R, imply left and right, respectively. The subscript 2 refers to the radix-2 or binary system. Considering the $S(B_2 \times B_2) = \{00, 01, 10, 11\}$, the elements are regarded as ordered 2-tuples $(a_i a_j)$. The a_i and a_j parts represent

the binary logic value of M_2^L and M_2^R , respectively. Thus, $(a_i a_j) = (10)$ means that the logic value of M_2^L is 1, while that of M_2^R is 0. The corresponding logic value notation in B_4 is 2 but, of course, has precisely the same implication for M_2^L and M_2^R .

By a machine M_R^α is meant a deterministic, finite-state, sequential machine of the Mealy or Moore type having a set of inputs I , a set of internal states S , and a set of outputs Z . When two machines of the same radix, M_R^α and M_R^β , are specified the set of states of M_R^α are designated by $\{S_i^\alpha\}$ and the set of states of M_R^β by $\{S_j^\beta\}$. Using the left and right designations for α and β and restricting the radix to the binary case, the set of states for M_2^L and M_2^R become $\{S_i^L\}$ and $\{S_j^R\}$, respectively.

The construction of the product machine formed by combining two binary machines is $M_2^L \times M_2^R$ and follows closely the definition of the general composite machine given by Smith and Kohavi [35]. However, in the discussion below, the concept is generalized further by making the inputs to each logical machine independent of the other machine, thus resulting in a total composite machine.

Definition 3.1. With respect to two machines M_2^L and M_2^R having sets of states $\{S_i^L\}$ and $\{S_j^R\}$, respectively, a total composite machine (TCM) is that machine which contains the set of states $\{S_i^L \times S_j^R\} = \{S_{ij}\}$, where (x) denotes the Cartesian product and S_{ij} is the new symbol for the state of the TCM which corresponds to S_i^L in M_2^L and (simultaneously) to S_j^R in M_2^R . If Z_{ip}^L and Z_{jq}^R are the outputs produced by the inputs I_p and I_q when M_2^L and M_2^R , respectively, are in states S_i^L and S_j^R , then the corresponding output of the TCM is designated by $Z_{ip}^L Z_{jq}^R = Z_{ijpq}$.

In the case of a Moore machine, the outputs are independent of I_p and I_q ; thus, the outputs of M_2^L , M_2^R , and the TCM become simply Z_i^L , Z_j^R , and Z_{ij} , respectively.

If the next-state values of M_2^L and M_2^R are related to the present-state values and input values by the expressions

$$S_i^L I_p \rightarrow S_r^L \quad \text{and} \quad S_j^R I_q \rightarrow S_t^R,$$

then the next state value of the TCM is related to its present-state value and input by the corresponding expression.

$$S_{ij} I_{pq} \rightarrow S_{rt}.$$

Thus, the state table for a TCM can be constructed from the state tables of M_2^L and M_2^R .

Figure 3.3 shows a state table for a binary asynchronous machine, M_2 . Both the next-state and output subtables are given. Circled entries in the table indicate stable machine states. Letting $M_2^L = M_2^R = M_2$, the TCM is formed by $M_2^L \times M_2^R$. For M_2 having s states (in this case $s = 3$), $M_2^L \times M_2^R$ has $s^2 = 9$ states. The next-state table for the TCM is given in Figure 3.4.

| S_i | NEXT-STATE S | | OUTPUT Z | |
|-------|-----------------|-------|-------------|-------|
| | I_1 | I_2 | I_1 | I_2 |
| S_1 | S_2 | S_1 | - | Z_1 |
| S_2 | S_2 | S_3 | Z_2 | - |
| S_3 | S_1 | S_3 | - | Z_3 |

Figure 3.3. Binary Machine State Table

| S_{ij} | NEXT-STATE S | | | | OUTPUT Z | | | |
|----------|-----------------|----------|----------|----------|-------------|----------|----------|----------|
| | I_1 | I_2 | I_3 | I_4 | I_1 | I_2 | I_3 | I_4 |
| S_{11} | S_{22} | S_{21} | S_{12} | S_{11} | - | - | - | Z_{11} |
| S_{12} | S_{22} | S_{23} | S_{12} | S_{13} | - | - | Z_{12} | - |
| S_{13} | S_{21} | S_{23} | S_{11} | S_{13} | - | - | - | Z_{13} |
| S_{21} | S_{22} | S_{21} | S_{32} | S_{31} | - | Z_{21} | - | - |
| S_{22} | S_{22} | S_{23} | S_{32} | S_{33} | Z_{22} | - | - | - |
| S_{23} | S_{21} | S_{23} | S_{31} | S_{33} | - | Z_{23} | - | - |
| S_{31} | S_{12} | S_{11} | S_{32} | S_{31} | - | - | - | Z_{31} |
| S_{32} | S_{12} | S_{13} | S_{32} | S_{33} | - | - | Z_{32} | - |
| S_{33} | S_{11} | S_{13} | S_{31} | S_{33} | - | - | - | Z_{33} |

Figure 3.4. TCM State Table

Before further development of TCM properties, the concepts of state and machine equivalence are required. Lee [23] defines these as follows:

Definition 4.2. Let S_i and S_j be two states of machines M_2^α and M_2^β (M_2^α and M_2^β may be the same machine). S_i and S_j are said to be equivalent if for any sequence of input symbols applied to them, the output sequences are identical. If S_i and S_j are not equivalent, we say they are distinguishable.

Definition 4.3. Let M_2^α and M_2^β be two machines. M_2^α and M_2^β are said to be equivalent if for every state of M_2^α there exists at least one equivalent state in M_2^β and visa versa. If M_2^α and M_2^β are not equivalent, we say that they are distinguishable.

The following theorem for the TCM, which is based on a similar theorem in [35], can now be proven.

Theorem 3.2. If M_2^L and M_2^R having sets of states $\{S_i^L\}$ and $\{S_j^R\}$, respectively, are reduced machines, then the TCM having the set of states $\{S_{ij}\}$ is a reduced machine.

Proof: Let the next-state values of M_2^L and M_2^R , respectively, be related to the present-state values and to the input by the expressions:

$$\begin{array}{ll} S_{i p}^L \xrightarrow{I} S_r^L & S_{j q}^R \xrightarrow{I} S_t^R \\ S_{m p}^L \xrightarrow{I} S_u^L & S_{n q}^R \xrightarrow{I} S_v^R \end{array}$$

The next-state values for the TCM are given by

$$S_{ij} \xrightarrow{I} S_{rt} \quad S_{mn} \xrightarrow{I} S_{uv} .$$

S_{ij} is equivalent to S_{mn} if and only if Z_{ijpq} is identical to Z_{mnpq} for every p and q , and likewise for S_{rt} and S_{uv} . But $Z_{ijpq} = Z_{ip}^L Z_{jq}^R$ and $Z_{mnpq} = Z_{mp}^L Z_{nq}^R$. Also $S_{rt} = S_r^L S_t^R$ and $S_{uv} = S_u^L S_v^R$. The equivalence of S_{rt} and S_{uv} implies the equivalence $S_r^L = S_u^L$ and $S_t^R = S_v^R$. However, since the original machines M_2^L and M_2^R are in reduced form, S_r^L and S_u^L cannot be equivalent, and S_t^R cannot be equivalent to S_v^R . Therefore, S_{rt} is not equivalent to S_{uv} . A special case occurs if for every p and q

$$\begin{array}{ll} S_{i p}^L \xrightarrow{I} S_r^L & S_{j q}^R \xrightarrow{I} S_t^R \\ S_{m p}^L \xrightarrow{I} S_r^L & S_{n q}^R \xrightarrow{I} S_t^R . \end{array}$$

Then $S_{ij} \xrightarrow{I} S_{rt}$ and $S_{mn} \xrightarrow{I} S_{rt}$. Since S_i^L and S_m^L are not equivalent, $Z_{ip}^L \neq Z_{mp}^L$, and similarly $Z_{jq}^R \neq Z_{nq}^R$ since S_j^R is known not to be

equivalent to S_n^R . This implies $Z_{ijpq} \neq Z_{mnpq}$; hence $S_{ij} \neq S_{mn}$. QED

From this theorem we gain the insight that the TCM is reduced and further work toward its reduction is not required. That is, for a TCM composed of two identical binary machines, the TCM is a reduced machine if the binary machine is first reduced. The reduction process is much easier if it is performed on the binary machine rather than the TCM since the TCM has many more states.

It should be clear at this point that a TCM is exactly a state integrated and multiplexed machine. The realization of such a machine in the form of a multiple-valued sequential circuit is the topic of a subsequent chapter.

Logic Circuit Analysis Using Simulation

Logic simulation is the process of formulating a model of a logic circuit and exercising the model for purposes of signal evaluation as a function of time and some set of applied inputs. Two applications are typically made of logic simulators. The first is the evaluation of a new circuit design for logical correctness, timing, signal characteristics, and possibly race and hazard conditions. The second application is for the evaluation of fault conditions. Fault simulation is important for the generation of circuit tests and the determination of circuit operation under the presence of fault conditions.

This section is concerned with the formulation and description of a four-valued (quaternary) logic simulator. Such a simulator, termed QLOSIM and operating in a timesharing environment, has been constructed using the FORTRAN language. The simulator is considered to be satisfactory for use in evaluating four-valued circuit designs based on the

assumptions presented in previous sections. No fault simulation capability is provided, although this is a logical extension to the work presented here. To supplement the formal simulator description contained in this section, an appendix is provided to assist a user in the actual execution of the software. Much of the following discussion is based on material found in [24].

Logic circuit simulators are generally classified as deterministic [31] since for a given input condition the output is uniquely determined. The two primary types of logic simulators, i.e., gate-level and functional, are both considered deterministic. Both types generally handle only precisely defined logic functions, gates, or functional "black-box" circuits. A gate-level simulator often places the designer closer to the actual circuit implementation than does a functional simulator. Gate-level simulation requires the actual specification of the logic gates as well as the topology of the circuit being simulated. With this approach a critical evaluation of circuit behavior can be made. Because of these qualities the gate-level simulator was chosen for implementation and is believed to be adequate for supporting and demonstrating the concepts of state integrated and multiplexed digital systems. Further, it is anticipated that the four-valued simulator presented here will satisfy not only the immediate needs of this thesis, but will be general enough to support other research efforts in multiple-valued logic.

The input information for the QLOSIM logic simulator consists of:

1. Description of the circuit to be simulated.

2. Circuit input values specified with timing information.
3. Initial value of circuit inputs and feedback lines.
4. Signals to be monitored and reported.

Ease of data entry for these parameters is an important consideration. Selection of a timesharing environment and FORTRAN list directed I/O was felt to be a versatile method for input and control of the logic simulator. The DECSys-10* computer was chosen as the host machine.

To enhance the universality of the simulator for multiple-valued logic, a rich set of four-valued functions are contained in the simulator. All of these functions have been proposed in the literature [4,10,11,21]. Non-unary function gates can contain two, three, or four inputs. All gates contain one output, and OR-tied outputs are not supported. Fan-in and fan-out constraints are ignored.

In addition to the functions described in the previous chapter, the strongly negated or complemented functions for the MIN, MAX, AND, and OR are included in the simulator. These are commonly written as $\overline{\text{MIN}}$, $\overline{\text{MAX}}$, $\overline{\text{NAND}}$ and $\overline{\text{NOR}}$. The generalized form of the LITERAL or window function is supported and is defined as:

$$\text{LITERAL: } BX_i^{a,b} = \begin{cases} 0 & \text{if } X_i < a \text{ or } X_i > b \\ B & \text{if } a \leq X_i \leq b \end{cases}$$

where $0 \leq B \leq 3$.

McCluskey [11] has reported three functions that are useful in I^2L fabrication of four-level logic circuits. Two of these, the PLUS and INHIBIT, can be implemented somewhat naturally in the I^2L circuitry.

*DEC is a trademark of Digital Equipment Corporation, Maynard, Massachusetts.

These functions are defined as follows:

$$\text{PLUS: } X_1 \oplus X_2 = \begin{cases} X_1 + X_2 & \text{if } X_1 + X_2 \leq 3 \\ 3 & \text{if } X_1 + X_2 > 3 \end{cases}$$

$$\text{INHIBIT: } BX_i^{\circ} = \begin{cases} 0 & \text{if } X_i \neq 0 \\ B & \text{if } X_i = 0 \end{cases}$$

where $0 \leq B \leq 3$. The $\overline{\text{PLUS}}$ function is also included. The third function is a universal gate that utilizes the $\overline{\text{MAX}}$, $\overline{\text{PLUS}}$ and INHIBIT functions.

The gate is described by

$$f(W_1, W_2, X_1, X_2, Y_1, Y_2) = Y_1^{\circ} Y_2^{\circ} \overline{(W_1 \vee W_2) \oplus X_1 \oplus X_2}^B$$

where $0 \leq B \leq 3$.

In addition to other I²L circuits, Dao [4] has described a multiple-valued multiplexer. Clever use of the multiplexer in digital circuits can often create simplistic designs [32]. The quaternary logic simulator includes a 4-to-1 multiplexer having four-valued inputs described below as X_1, X_2, X_3, X_4 , and Y_1 . The Y_1 input selects one of the X_i inputs and causes the gate output to be set to the value of the selected input. The multiplexer output is thus given by

$$f(X_1, X_2, X_3, X_4, Y_1) = \begin{cases} X_1 & \text{if } Y_1 = 0 \\ X_2 & \text{if } Y_1 = 1 \\ X_3 & \text{if } Y_1 = 2 \\ X_4 & \text{if } Y_1 = 3. \end{cases}$$

Each gate is assumed to react to input conditions as described previously. A transport delay, Δ_T , must be provided for each gate in

the circuit. The value specified is constant for that particular gate. The inertial delay, Δ_T , is assumed to be zero for every gate. This yields a typical gate model as given in Figure 3.5. The rise and fall time delays, Δ_R and Δ_F , are implemented according to the discrete rise/fall time assumption given in the previous section. Further, it is assumed that the rise and fall times are equal and constant throughout the circuit.

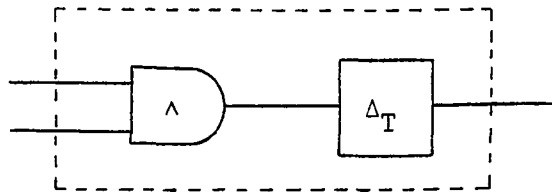


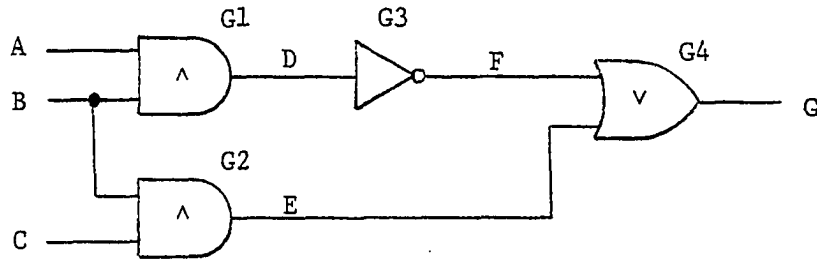
Figure 3.5. Typical Gate Delay Model

The QLOSIM logic simulator is a unit-time, event-driven simulator containing several lists implemented as arrays. These lists are used to store circuit gates, interconnections, logic values, time values, and various flags. During the execution of the simulator, all lists reside in main memory.

The circuit topology is integrated into three lists with each list containing links or pointers to the other two lists. Referring to Figure 3.6(b), the three lists are shown as the circuit node list (NLIST), the gate list (GLIST) and the gate fan-in list (INLIST). Each circuit node (interconnect) and gate are assigned unique identifiers or names by the user. This is depicted for the sample circuit in Figure 3.6(a).

To outline the method used for circuit traversal during the simulation process, the circuit of Figure 3.6 will be used. First, note

that each list is implemented by arrays with each array element having an implied relative position or index. The index is not actually stored in the list, rather each element is referenced by its index or subscript. List pointers are simply the index or subscript value that point to the desired elements in another list. For example, both nodes A and B are inputs to gate G1. Both nodes contain a "1" in the NLIST first gate pointer field indicating that they input to the first gate (G1) in the GLIST. A pointer value of zero indicates the null or non-existent pointer. Node G contains a zero pointer since it is an output only node.



(a)

| <u>NLIST</u> | | | <u>INLIST</u> | | |
|--------------|---------|-----------------------|---------------|--------------|-------------------|
| Index | Node ID | Pointer to First Gate | Index | Node Pointer | Next Gate Pointer |
| 1 | A | 1 | 1 | 1 | 0 |
| 2 | B | 1 | 2 | 2 | 2 |
| 3 | C | 2 | 3 | 2 | 0 |
| 4 | D | 3 | 4 | 3 | 0 |
| 5 | E | 4 | 5 | 4 | 0 |
| 6 | F | 4 | 6 | 6 | 0 |
| 7 | G | 0 | 7 | 5 | 0 |

| <u>GLIST</u> | | | | |
|--------------|---------|---------------------|----------------------|---------------------|
| Index | Gate ID | Fan-In List Pointer | Fan-In List Quantity | Output Node Pointer |
| 1 | G1 | 1 | 2 | 4 |
| 2 | G2 | 3 | 2 | 5 |
| 3 | G3 | 5 | 1 | 6 |
| 4 | G4 | 6 | 2 | 7 |

(b)

Figure 3.6 Circuit Topology Data Structure

Continuing with the example, the fan-in list pointer field in GLIST provides a link to the INLIST. The number of successive entries in the INLIST that are associated with a particular gate is found in the GLIST fan-in quantity field. The quantity field is necessary since the number of gate inputs is variable depending on the type of gate. Using the gate G1, it is seen that two INLIST entries, beginning with the first element, are associated with G1. The first INLIST entry (Index=1) contains a pointer back to the first NLIST element, i.e., node A. Since node A does not fan-out to any other gates, the next gate pointer for the first INLIST entry is set to zero. In contrast, node B is the second INLIST entry, and it is also an input to gate G2. This circuit connection is represented by setting the INLIST next gate pointer value equal to two, thus pointing to gate G2.

Using this data structure arrangement, it is possible to (1) start with a given node and determine all gates that have the node as an input, and (2) start with a given gate and determine all input and output nodes associated with the gate. These two basic circuit topology requirements are necessary for performing the simulation process.

Figure 3.6 shows only selected elements that are contained in the NLIST and GLIST. The NLIST also contains the immediate logic value for the node, a flag that indicates whether or not the node is monitored and reported, and a link to the event queue which is discussed below. To conserve memory space, the logic value and report flag fields are packed into one array element. The additional GLIST fields are the gate type and the transport delay value.

As mentioned previously, the simulator is event driven. This implies that the simulator's internal time counter is advanced to the next event requiring evaluation. Using this method, no computer overhead is incurred for intervals during which no circuit changes occur. To implement this scheme, a time-event queue is necessary to record or schedule all future circuit changes. The particular queue cell structure used by the quaternary simulator is given in Figure 3.7. One time-event queue cell is used for each circuit change to be simulated. Five hundred memory resident cells are provided in the simulator.

The queue is implemented as a linked list in which the cells are linked in ascending order by time of change. The next time-event cell link will always point to the cell having the next highest time

| | | | | |
|----------------------|-------------------------|-----------------------------------|--------------------------|---------------------------------------|
| Time of Change | Node List Pointer | Logic Value and Delete Flag | Next T-E Cell Link | Next T-E Cell on Same Node Link |
|----------------------|-------------------------|-----------------------------------|--------------------------|---------------------------------------|

Figure 3.7. Time-Event Queue Structure

value. Logic changes or events scheduled for a given node are also linked together. This facility simplifies scanning of the queue for the purposes of unscheduling events under certain conditions and the inspection of future scheduled node changes and logic values.

Still referring to Figure 3.7, the node list pointer links the queue cell with a node list cell. The node pointed to by this field is the particular circuit node for which the logic change is scheduled. The logic value to be assumed is packed with the delete flag. The delete flag is used to tag processed cells that are subsequently deleted by a garbage collection routine. Queue deletions occur following all processing for a given time value.

Based on these list data structures, the simulation process is similar to those previously described for binary logic simulators [24]. The unique function for multiple-valued simulation is the scheduling of multiple-valued signal transitions under the assumptions given previously. The linear ordering assumption, in particular, leads to subtle algorithmic processes. Future events are scheduled in the simulator using the time-event queue. For a given gate output change, the quaternary case requires that zero to three new events are scheduled to represent the output transitions. However, it may be the case that one or more of these future events will not occur. To depict this situation and better demonstrate the multiple-valued process, a brief example is given.

Figure 3.8 shows a hypothetical unary function. The gate transport delay, Δ_T , is assumed to be 10 time units. Further, assume that the signal rise/fall delay is one time unit. Figure 3.9 provides a representation of the pertinent data structure elements necessary for the example function. To set the initial conditions, assume that an X_1 transition from logic value 1 to 0 has just occurred, and the future events for X_2 have been scheduled at $t_1 + 10$, $t_1 + 11$ and $t_1 + 12$. Letting t_1 arbitrarily equal 50, the TETIME values are 60, 61, and 62, respectively. The corresponding logic value for each time is given in the TEVAL field. Note that the pointers for node X_2 have the following structure and are shown pictorially by solid lines rather than quantitatively. The node list time-event pointer (NLTEPT) for node X_2 points to the last time-event list entry for this node. The next cell on the same node field (TENCSN) links the remaining two entries for node X_2 (solid lines). The time-event pointer links all three X_2 node entries

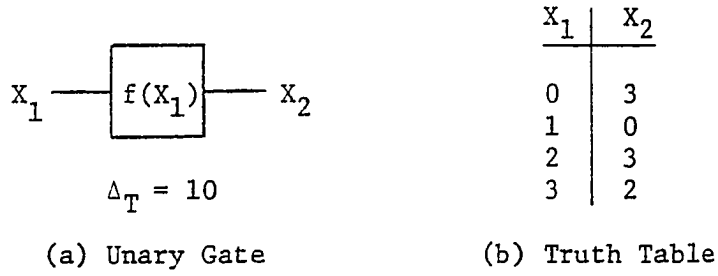


Figure 3.8. Unary Gate Simulation Example

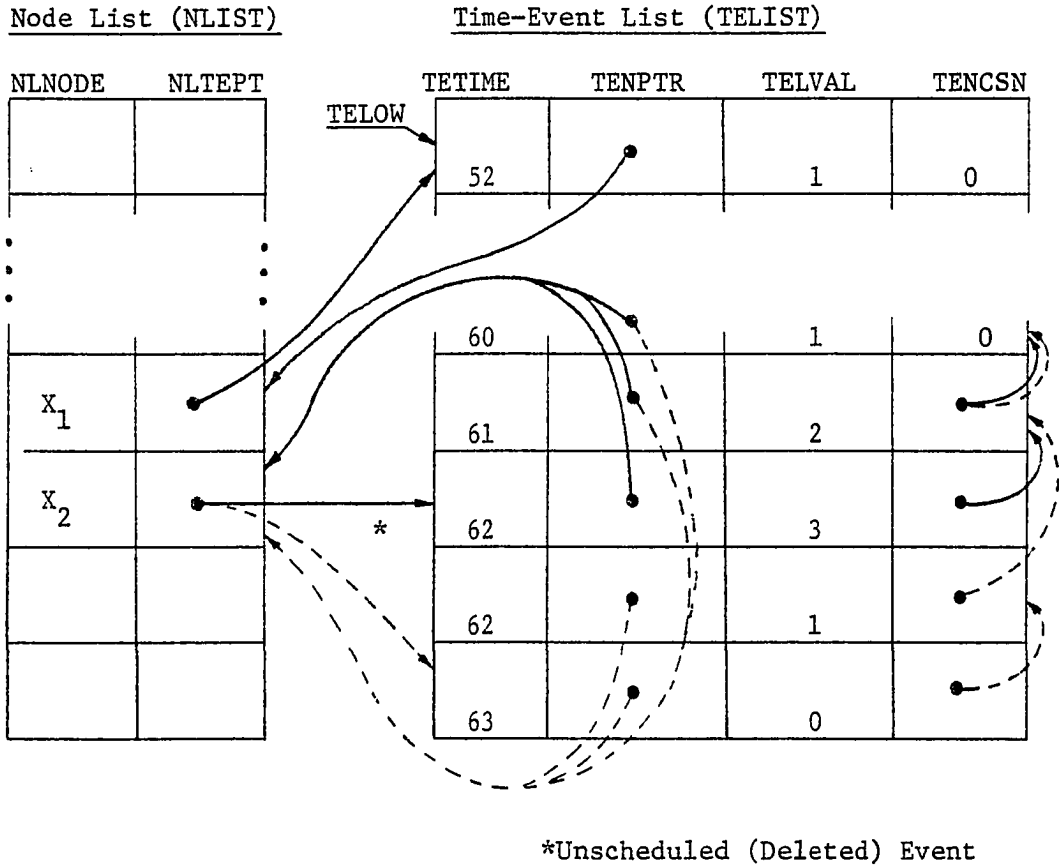


Figure 3.9. Simulation Example Data Structure

to the X_2 node cell in the node list. Only one time-event list entry exists for node X_1 , but it is linked in a similar fashion.

The variable `TELOW` points to the next time-event cell (event) to be processed. This occurs at Time 52 and represents a change on node X_1 from logic value 0 to 1. When this cell is processed, logic value transitions for node X_2 must be scheduled based upon the gate function as well as future events residing in the time-event list. With a transport delay of 10 time units, the earliest X_2 change should occur at Time 62. However, a future event for node X_2 is already scheduled at Time 62. To resolve this future event conflict, the simulator scans the time-event list for node X_2 and examines the `TETIME` field. If `TETIME` is greater than or equal to the current time plus transport delay, 62 in this case, the cell is marked for deletion. This cell represents a previously scheduled event that will not occur and must be unscheduled. New events for X_2 are scheduled based upon the latest (in time) event that remains undeleted. The latest event occurs at Time 61 and has a logic value of 2. New future event logic values to be added to the time-event queue consider that X_2 will have a logic value of 2 at Time 61.

Unscheduled requires the rearrangement of pointers and links between the lists after the new events for node X_2 at Time 62 and 63 are added. The results of this process are shown by the dotted lines. The final result is that X_2 now has a 0 to 1 transition scheduled at Time 60, a 1 to 2 transition scheduled at Time 61, a 2 to 1 transition at Time 62 and a 1 to 0 transition at Time 63. The transition from 2 to 3 at Time 62 is purged from the time-event list.

A timing diagram for this example is given in Figure 3.10. The diagram is drawn using discrete time units to depict the nature of the simulation. Note that the linear ordering assumptions are preserved. New events are always scheduled with regard to this assumption and thus represent the operation of devices that exhibit this behavior.

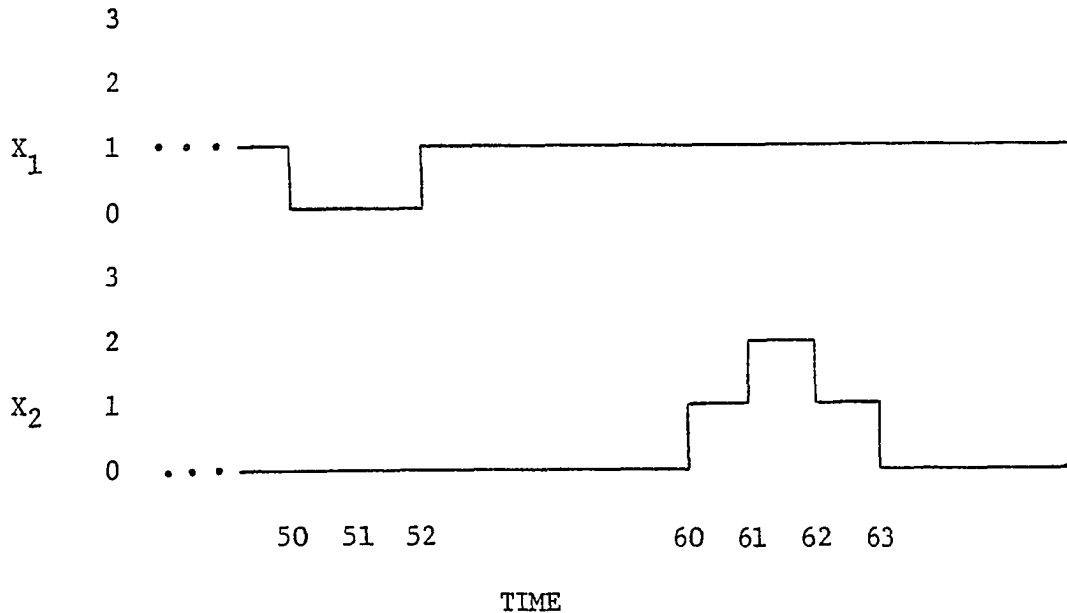


Figure 3.10. Simulation Example Timing Diagram

To illustrate actual results obtainable from the quaternary logic simulator, two circuits that form a basis for later developments will be given. Both a combinational and a sequential circuit are simulated. Input options necessary for generation of the particular report formats exhibited here can be found in Appendix A.

The first circuit considered is the four-valued state integrated and multiplexed half adder that was introduced in Chapter II. The half adder logic circuit is given in Figure 3.11 and is shown without the encoder/decoder interface. Recall that the logic gates and signals are four-valued and that the circuit simultaneously produces the sum and

carry outputs as if it were two identical binary half adders operating in parallel. Nodes A and B are inputs to the circuit. The sum output is S, and the carry output is C.

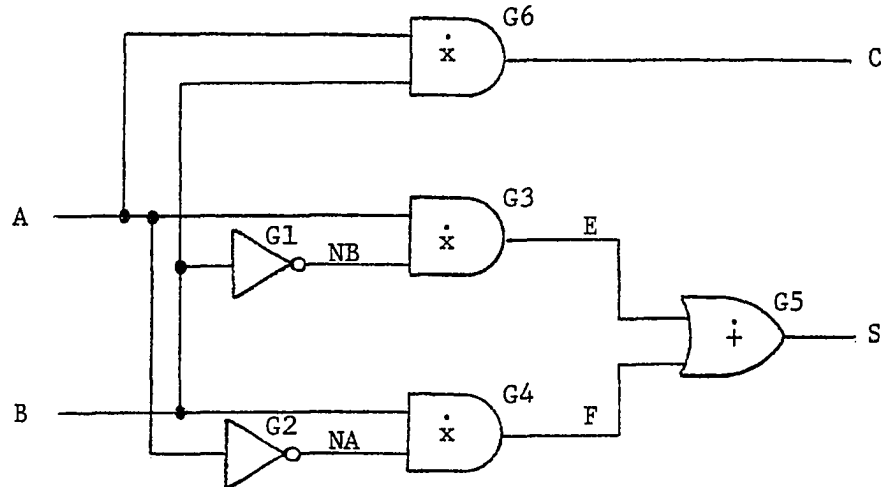


Figure 3.11. Four-Valued Half Adder Circuit

A listing of the simulator input file is given in Figure 3.12. Different groups of input records are delimited with '\$\$'. The input file groups, in order of occurrence, specify the circuit topology, initial node conditions for nodes A and B, the dynamic input values for input nodes A and B, and the circuit nodes to be traced or reported. Note that nodes A, B, S, and C are made reportable so that only these nodes will appear in the simulator tabular output report.

Figure 3.13 shows the results of the half adder simulation. To conserve space, only the tabular simulator output is given. For this simulator execution the rise/fall time was specified via the user terminal as $\Delta_R = \Delta_F = 1$ time unit. This is clearly evident in the output listing by the time required for nodes A and B to change from

logic level 0 to logic level 2. As shown in the input file listing, the transport delay, Δ_T , for each gate is 10 time units.

By analyzing the simulator output, it is seen that output node S produces the simultaneous sum of inputs A and B. For example, at Time 32, T settles to a value of 1, indicating that the simultaneous sum of $A = 2$ and $B = 3$ is 1. Thinking of the values for A and B as the 2-tuples (10) and (11), it is clear that the elementwise sum is (01). The carry output C is 2, or alternately (10), and has settled by Time 12. This value for C indicates a carry from the left 2-tuple element, which is indeed the case for these values of A and B. At Time 100, node B begins a change to a value of 1. The sum given at Time 133 is the simultaneous sum of $A = 2$ and $B = 1$. The remaining input changes for A and B and the resulting sum and carry outputs can be examined in a similar manner.

It is interesting to note that output node S tends to wander about before finally stabilizing on the final value. This is due to

```
'G1' 'COMP' 10 'B' 'NB' /
'G2' 'COMP' 10 'A' 'NA' /
'G3' 'AND2' 10 'A' 'NB' 'E' /
'G4' 'AND2' 10 'B' 'NA' 'F' /
'G5' 'OR2' 10 'E' 'F' 'S' /
'G6' 'AND2' 10 'A' 'B' 'C' /
'$$' /
'A' 0
'B' 0
'$$' /
'A' 1,2 200,1 300,0 /
'B' 1,3 100,1 200,0 /
'$$' /
'A' 'B' 'S' 'C' /
'$$' /
```

Figure 3.12. Four-Valued Half Adder Simulator Input Data

QUATERNARY LOGIC SIMULATOR - QLOSIM

HADDER.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | A | 0 | 1 |
| 1 | B | 0 | 1 |
| 2 | A | 1 | 2 |
| 2 | B | 1 | 2 |
| 3 | B | 2 | 3 |
| 11 | C | 0 | 1 |
| 12 | C | 1 | 2 |
| 21 | S | 0 | 1 |
| 22 | S | 1 | 2 |
| 23 | S | 2 | 3 |
| 31 | S | 3 | 2 |
| 32 | S | 2 | 1 |
| 100 | B | 3 | 2 |
| 101 | B | 2 | 1 |
| 111 | C | 2 | 1 |
| 112 | C | 1 | 0 |
| 120 | S | 1 | 0 |
| 121 | S | 0 | 1 |
| 132 | S | 1 | 2 |
| 133 | S | 2 | 3 |
| 200 | A | 2 | 1 |
| 200 | B | 1 | 0 |
| 220 | S | 3 | 2 |
| 221 | S | 2 | 1 |
| 222 | S | 1 | 0 |
| 230 | S | 0 | 1 |
| 300 | A | 1 | 0 |
| 320 | S | 1 | 0 |

SIMULATION TERMINATED AT TIME 320

Figure 3.13. Four-Valued Half Adder Simulator Output

the assumption that each gate reacts to all input changes; therefore, each gate will attempt to reflect its respective truth table values on its output. The ramifications of this action are discussed in more detail in the remaining chapters.

The second example circuit is a basic four-valued cross-coupled set-reset r-flop that has been suggested by several authors [15,16,19]. Wills [19] provides a thorough analysis of the circuit. Referring to Figure 3.14, the four-valued circuit is similar to the cross-coupled binary NOR gates in the familiar binary system; however, for the multiple-valued case the gates are the $\overline{\text{MAX}}$ type. The next-state table for the

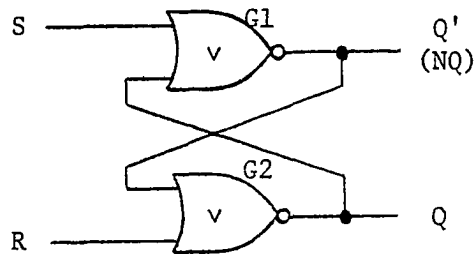


Figure 3.14. Four-Valued RS r-Flop

RS r-flop is shown in Figure 3.15. The dashed entries represent states that are the result of undesirable inputs on the R and S inputs. These are the multiple-valued equivalent to the undesirable input in the binary case when $R = S = 1$.

Figure 3.16 gives the input file specification for the RS r-flop. The transport delay, Δ_T , is 10 time units for each gate. The initial conditions specified in the file place the r-flop in the reset condition at Time 0 with $Q = 0$ and $NQ = Q' = 3$. All four nodes in the circuit are reported.

| | | Q(t+1) | | | | | | | | | | | | | | | |
|------|---|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | RS | | | | RS | | | | RS | | | | RS | | | |
| | | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| Q(t) | 0 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | - | 0 | 1 | - | - | 0 | - | - | - |
| | 1 | 1 | 1 | 2 | 3 | 1 | 1 | 2 | - | 1 | 1 | - | - | 0 | - | - | - |
| | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | - | 1 | 1 | - | - | 0 | - | - | - |
| | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | - | 1 | 1 | - | - | 0 | - | - | - |

Figure 3.15. Next-State Table for Four-Valued RS r-Flop

```

'G1' 'NMAX2' 10 'S' 'Q' 'NQ' /
'G2' 'NMAX2' 10 'R' 'NQ' 'Q' /
'$$' /
'R' 0
'S' 0
'Q' 0
'NQ' 3
'$$' /
'S' 1,2 100,0 400,3 500,0 /
'R' 200,1 300,0 600,3 700,0 /
'$$' /
'S' 'R' 'Q' 'NQ' /
'$$' /

```

Figure 3.16. Four-Valued RS r-Flop Simulator Input Data

QUATERNARY LOGIC SIMULATOR - QLOSIM

RSFF.EX2 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S | 0 | 1 |
| 2 | S | 1 | 2 |
| 11 | NQ | 3 | 2 |
| 12 | NQ | 2 | 1 |
| 21 | Q | 0 | 1 |
| 22 | Q | 1 | 2 |
| 100 | S | 2 | 1 |
| 101 | S | 1 | 0 |
| 200 | R | 0 | 1 |
| 300 | R | 1 | 0 |
| 400 | S | 0 | 1 |
| 401 | S | 1 | 2 |
| 402 | S | 2 | 3 |
| 412 | NQ | 1 | 0 |
| 422 | Q | 2 | 3 |
| 500 | S | 3 | 2 |
| 501 | S | 2 | 1 |
| 502 | S | 1 | 0 |
| 600 | R | 0 | 1 |
| 601 | R | 1 | 2 |
| 602 | R | 2 | 3 |
| 610 | Q | 3 | 2 |
| 611 | Q | 2 | 1 |
| 612 | Q | 1 | 0 |
| 620 | NQ | 0 | 1 |
| 621 | NQ | 1 | 2 |
| 622 | NQ | 2 | 3 |
| 700 | R | 3 | 2 |
| 701 | R | 2 | 1 |
| 702 | R | 1 | 0 |

SIMULATION TERMINATED AT TIME 702

Figure 3.17. Four-Valued RS r-Flop Simulator Output

The QLOSIM simulator output shown in Figure 3.17 is, again, only the tabular portion of the output report. Using the next-state table for the RS r-flop, Figure 3.15, the simulator output shows that the circuit does indeed perform as expected. For the first circuit stimulation, the set input, S, rises from 0 to 2. The circuit responds by entering state $Q = 2$ at Time 22. The S input then returns to logic value 0. At Time 200 the reset input, R, rises to 1 and returns to 0 at Time 300. As predicted by the next-state table, the r-flop remains in state $Q = 2$. The next circuit input occurs at Time 400 with S making a transition from 0 to 3. This changes the r-flop state from $Q = 2$ to $Q = 3$. Again, S returns to 0. At Time 600 R goes from 0 to 3; the r-flop resets to state $Q = 0$, and the simulation terminates with R at 0 and $Q = 0$.

This is not an exhaustive simulation of all transitions, but it demonstrates the use of the simulator for partial operational verification of a simple sequential circuit. The RS r-flop is an important basic circuit from which other r-flops can be constructed. This circuit will be considered again in Chapter V.

In addition to studying circuit behavior the logic simulator can be used for evaluating four-valued mixed algebraic expressions, i.e., expressions that contain both Post and Boolean operations. A relationship that will be used in Chapter V can be handled in this manner. Consider the mixed algebraic expression

$$X \vee (X \dot{+} Y) = X \dot{+} Y .$$

This relationship can be verified by using the circuit given in Figure 3.18, where gates G1 and G2 represent the left side of the expression and gate G3 represents the right side. Inputting all possible values for X and Y into the circuit and observing that nodes Z and W always have the same response for a given input condition verifies the relationship. The simulator output for this circuit is given in Figure 3.19 and clearly shows that the conditions are met. In a similar manner, the relation

$$X \wedge (X \dot{\bar{x}} Y) = X \dot{\bar{x}} Y$$

can be shown to be true. This simulator capability is an additional benefit that was not anticipated during its construction, but the results generated above are quite necessary for later work in memory element development.

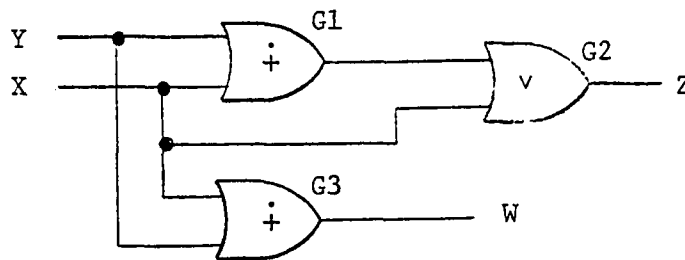


Figure 3.18. Circuit for Verification of
 $X \vee (X \dot{+} Y) = X \dot{+} Y$

Multiple-Valued Boolean Gate Realizations

It is clear that the construction of a state integrated and multiplexed system with binary sub-machines will require the development of multiple-valued Boolean AND and OR gates. Fabrication of multiple-valued I²L circuits has proven successful [3,4] and has

QUATERNARY LOGIC SIMULATOR - QLOSIM

IDENT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | Y | 0 | 1 |
| 11 | W | 0 | 1 |
| 21 | Z | 0 | 1 |
| 100 | Y | 1 | 2 |
| 110 | W | 1 | 2 |
| 120 | Z | 1 | 2 |
| 200 | Y | 2 | 3 |
| 210 | W | 2 | 3 |
| 220 | Z | 2 | 3 |
| 300 | X | 0 | 1 |
| 300 | Y | 3 | 2 |
| 301 | Y | 2 | 1 |
| 302 | Y | 1 | 0 |
| 311 | W | 3 | 2 |
| 312 | W | 2 | 1 |
| 321 | Z | 3 | 2 |
| 322 | Z | 2 | 1 |
| 400 | Y | 0 | 1 |
| 500 | Y | 1 | 2 |
| 510 | W | 1 | 2 |
| 511 | W | 2 | 3 |
| 520 | Z | 1 | 2 |
| 521 | Z | 2 | 3 |
| 600 | Y | 2 | 3 |
| 700 | X | 1 | 2 |
| 700 | Y | 3 | 2 |
| 701 | Y | 2 | 1 |
| 702 | Y | 1 | 0 |
| 710 | W | 3 | 2 |
| 720 | Z | 3 | 2 |
| 800 | Y | 0 | 1 |
| 810 | W | 2 | 3 |
| 820 | Z | 2 | 3 |
| 900 | Y | 1 | 2 |
| 910 | W | 3 | 2 |
| 920 | Z | 3 | 2 |
| 1000 | Y | 2 | 3 |
| 1010 | W | 2 | 3 |
| 1020 | Z | 2 | 3 |
| 1100 | X | 2 | 3 |
| 1100 | Y | 3 | 2 |
| 1101 | Y | 2 | 1 |
| 1102 | Y | 1 | 0 |
| 1200 | Y | 0 | 1 |
| 1300 | Y | 1 | 2 |
| 1400 | Y | 2 | 3 |

Figure 3.19. Simulator Verification of $X \vee (X \dot{+} Y) = X \dot{+} Y$

resulted in logic design techniques that can be used to specify four-valued Boolean gates. Although other circuit fabrication technologies might later be shown to yield less expensive gates, the I^2L technology is sufficiently developed to provide a basis for the Boolean gate specifications. No attempt is made to focus on the uniqueness of these particular circuit designs, rather they are offered to show feasibility. Only the two-input gates will be designed. Since the four-valued Boolean COMPLEMENT gate is identical to the normal four-valued I^2L COMPLEMENT gate [4], no new work is required for it.

The four-valued AND gate design will be given first. Details of the design procedure are omitted but are given by McCluskey [12]. Because of the richness of operations available in I^2L , several variations in the design are possible. The availability of the four-valued 4-to-1 I^2L multiplexer, described previously, allows the use of a design technique given in [32]. For a two-input AND gate, one of the inputs is used to control the multiplexer select input. Because of the symmetry of the AND gate, either input can be used in this manner. From the function map of the AND gate, Figure 3.2, four functions are written in the other input variable. These four functions become the four data inputs into the multiplexer.

Arbitrarily choose input variable X_2 to control the multiplexer. From Figure 3.2 the four functions in X_1 can be written using operations previously discussed. Referring to the first column in the AND gate function map, it is clear that

$$f_1(X_1) = 0 ,$$

thus giving the first function. Likewise, the last column in the map gives

$$f_4(X_1) = 3 .$$

The second function, from column two, can be written

$$f_2(X_1) = 1X_1^{1,1} \vee 1X_1^{3,3} ,$$

where \vee is the MAX operation. The third function is given as

$$f_3(X_1) = 2X_1^{2,3} .$$

Functions $f_1(X_1)$ and $f_2(X_1)$ are constant and are therefore realized somewhat naturally. In the I^2L technology they are generated by constant current sources in the circuit. Functions $f_2(X_1)$ and $f_3(X_1)$ both have I^2L realizations [4]. As mentioned above, other functionally equivalent realizations are also possible. For example, the MAX operation in the expression for $f_2(X_1)$ can be replaced by the PLUS operation. Applying the functions to the I^2L multiplexer, the two-input AND gate is realized and is shown in Figure 3.20. The X_2 input controls the select input S, and the four functions input to D_1 , through D_4 , respectively. The multiplexer output is the AND function.

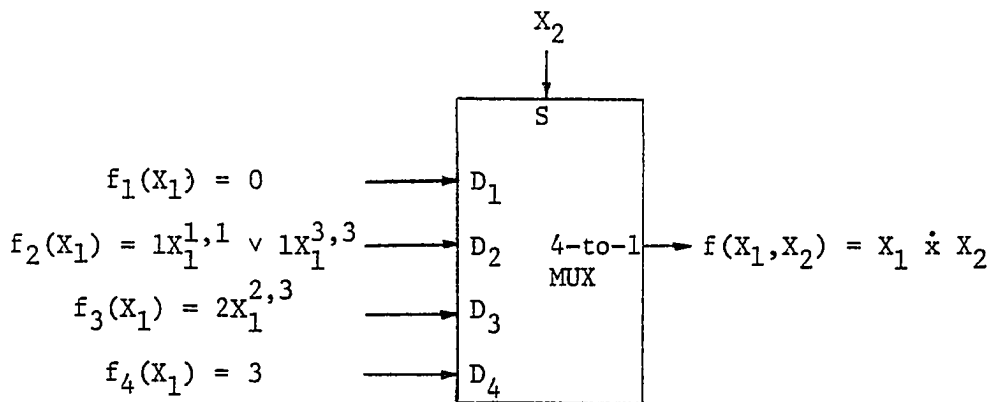


Figure 3.20. I^2L AND Gate Realization

Using the same design technique, the two-input OR gate expressions are written from the OR gate function map in Figure 3.2 as

$$\begin{aligned} f_1(X_1) &= X_1 \\ f_2(X_1) &= 1X_1^{0,1} \vee 3X_1^{2,3} \\ f_3(X_1) &= \overline{1X_1^{0,0} \vee 1X_1^{2,2}} \\ f_4(X_1) &= 3. \end{aligned}$$

The resulting realization is shown in Figure 3.21. Again, $f_2(X_1)$ and $f_3(X_1)$ have realizations in I^2L .

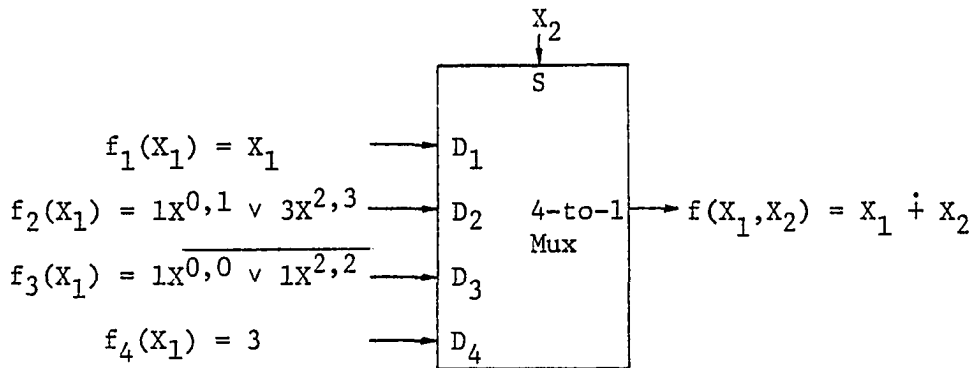


Figure 3.21. I^2L OR Gate Realization

The AND and OR gate circuits contain several transistor elements and will be considerably more expensive to fabricate than binary gates. However, there has been little or no previous work that suggested any sort of AND and OR gate realizations. In the following chapters the state integrated and multiplexed logic circuits will be examined in detail, and it is hoped that this work will provide motivation for improved multiple-valued Boolean gate realizations.

CHAPTER IV

STATE INTEGRATED AND MULTIPLEXED COMBINATIONAL CIRCUITS

Circuit Behavior Characteristics

Combinational logic circuits have the property that their outputs at any time are determined strictly by the circuit inputs at that time. The combinational circuit, therefore, does not possess feedback connections, but this circuit type is important for the design of other circuits that do contain feedback connections, i.e., sequential circuits. The restriction of time in the relationship of combinational circuit input to output is rather aloof in the physical sense because all circuit devices exhibit some delay in their operation. This physical reality will be of some significance in this study since the response of circuit elements that possess delay and the resulting circuit behavior is the primary focus of the analysis.

Classification of multiple-valued combinational circuit behavior is based on that given by Sheafor [15] and Wojcik [13,14]. Let $S_r = \{0,1,\dots,r-1\}$ denote the set of logical values of the algebraic system, and let $V_r = \{v_0, v_1, \dots, v_{r-1}\}$ be the physical quantities associated with the logical values. The logical and physical quantities are assumed to

be linearly ordered. For the following discussion let $\vec{X}_j = \{X_{(k-1)j}, \dots, X_{0j}\}$ be a k -tuple with each $X_{hj} \in S_r$.

Definition 4.1. Two k -tuples \vec{X}_i and \vec{X}_j are adjacent if and only if they differ in exactly one component X_{hi} and X_{hj} , respectively, $0 \leq h \leq k-1$, and $|X_{hi} - X_{hj}| \leq 1$.

Denote a k -variable function of the algebraic system by $f(\vec{X}) = f(X_{k-1}, \dots, X_0)$ and let \vec{X}_i and \vec{X}_j be distinct argument k -tuples of $f(\vec{X})$.

Definition 4.2. $f(\vec{X})$ contains an operation hazard if for any two adjacent k -tuples \vec{X}_i and \vec{X}_j , $|f(\vec{X}_i) - f(\vec{X}_j)| \geq 2$.

It should be noted that an operation hazard is a property of the function itself, and not of the function realizations, so that it cannot be eliminated. The hazard behavior is forced when the input changes from \vec{X}_i to \vec{X}_j and the output must change by more than a single value. If this output is used as an input to another circuit, incorrect operation may result.

Definition 4.3. Let \vec{X}_i be a k -tuple of inputs to a circuit, and let this k -tuple change to \vec{X}_j . This change is said to be a single-value single-input change (SVSIC) if \vec{X}_i and \vec{X}_j are adjacent. It is said to be a multiple-value single-input change (MVSIC) if \vec{X}_i and \vec{X}_j differ in exactly one component but are not adjacent. Otherwise, the change is a multiple-input change.

Definition 4.4. Let \vec{X}_i and \vec{X}_j be two input k -tuples (not necessarily adjacent). The output transition from $f(\vec{X}_i)$ to $f(\vec{X}_j)$ is said to be proper if the output assumes the values:

- a) $f(\vec{X}_i), f(\vec{X}_i)+1, \dots, f(\vec{X}_j)-1, f(\vec{X}_j)$, in exactly that order when $f(\vec{X}_i) < f(\vec{X}_j)$.

- b) $f(\vec{X}_i), f(\vec{X}_i)-1, \dots, f(\vec{X}_j)+1, f(\vec{X}_j)$, in exactly that order,
when $f(\vec{X}_i) > f(\vec{X}_j)$.
- c) $f(\vec{X}_i)$ only, when $f(\vec{X}_i) = f(\vec{X}_j)$.

In other words, a proper transition is one which proceeds directly from the initial value to the final value, and if any intermediate values occur, they are traversed in linear order.

Definition 4.5. A function $f(\vec{X})$ is said to be multiple-value tolerant with respect to input X_i (MVT(X_i)) if, given any k -tuple \vec{X}_j , consecutive single-value single-input changes in variable X_i produces a proper transition at the output. A function $f(\vec{X})$ is multiple-value tolerant (MVT) if it is MVT(X_i) for all i .

Sheafor [15] has shown that this definition is equivalent to saying that the function is monotone non-increasing or monotone non-decreasing in variable X_i .

Subsequent to these definitions and the assumptions given in Chapter III, the following additional operational characteristics are stated for purely combinational circuits.

1. Multiple-valued changes are allowed for any input variable.
2. The logic value of an input variable can change at any time.

Thus, any purely combinational circuit, e.g., a simple adder circuit, may experience multiple-valued multiple-input changes without regard to time. Multiple-valued input changes must be permitted in a state integrated and multiplexed system. For a four-valued system assume that both binary sub-machines have an input with logic value zero. For the

left machine input to change to logic value of 1, the input signal must change from 0 to 2. Acceptable operation must follow even though an intermediate logic value of 1 is experienced on the circuit input.

This permissive operating environment does not imply that all combinational circuits will or should be subjected to an uncontrolled stimulus, rather there exist no restrictions within the logic gates or quaternary simulator that prevent events of this type from occurring. The application of combinational circuits often employs a means of controlling the input and output signals. This is normally done by using a control or strobe signal to AND the input or output signals, thus allowing the timely application of input information and the timely sampling of the circuit output. In essence, the practical methods used in binary logic circuits to control logic signals can also be applied in the multiple-valued circuits; however, some undesirable gate reactions cannot be prevented. These issues will be investigated in the following sections.

Static Hazard Conditions

The occurrence of multiple-valued changes on circuit inputs has a major impact on state integrated and multiplexed combinational circuits. The $B_2 \times B_2$ functions, AND, OR, and COMPLEMENT, are shown again in Figure 4.1, except here the variables and function outputs are given as 2-tuples. From this representation it is easily seen that the operations are performed elementwise on the elements $a_i a_j$ that comprise the 2-tuples. Considering the AND function, it is seen that the operation is not multiple value tolerant in either X_1 or X_2 . When $X_1 = 01$ and X_2 changes from a value of 01 to 11, the output will take on an

intermediate value of 00. The same condition exists when X_1 and X_2 are interchanged. A similar condition exists in the OR function when X_1 or $X_2 = 10$ and the other variable changes from 01 to 11.

| | | | | | | | | | | | | |
|-------|-------|----|----|----|-------|-------|----|----|----|--|------------|-------------|
| | X_2 | | | | | X_2 | | | | | X_1 | \bar{X}_1 |
| | 00 | 01 | 10 | 11 | | 00 | 01 | 10 | 11 | | 00 | 11 |
| X_1 | 00 | 00 | 00 | 00 | X_1 | 00 | 01 | 10 | 11 | | 01 | 10 |
| | 01 | 00 | 01 | 01 | | 01 | 01 | 11 | 11 | | 10 | 01 |
| | 10 | 00 | 00 | 10 | | 10 | 11 | 10 | 11 | | 10 | 01 |
| | 11 | 00 | 01 | 10 | | 11 | 11 | 11 | 11 | | 11 | 00 |
| | AND | | | | | OR | | | | | COMPLEMENT | |

Figure 4.1. $B_2 \times B_2$ Representation of Boolean Functions

This sort of behavior is hazard free in the multiple-valued sense of Definition 4.2. However, when the individual left and right machines are considered, the 2-tuples are adjacent, and static hazard conditions exist for the right logical circuit. For both the AND and OR functions the right circuit output experiences an intermediate 0 logic value. This behavior is described more precisely by the following binary system definitions [36].

Definition 4.6. When a binary output is to remain at the value 1 and a momentary 0 output is possible during the transition between two adjacent input states, the hazard is called a static-one-hazard.

Definition 4.7. When a binary output is to remain at the value 0 and a momentary 1 output is possible during the transition between two adjacent input states, the hazard is called a static-zero-hazard.

As described above, static hazards in $B_2 \times B_2$ gates result from the inherent nature of the four-valued AND and OR functions. It is

difficult to logically solve this sort of inherent operational difficulty without introducing specialized gate inputs to control the internal operation of the gate. The introduction of additional control makes the entire application of four-valued Boolean gates more expensive and less of an alternative. We will, therefore, not attempt to provide any method of control and will accept the inherent static hazard condition in the AND and OR gates.

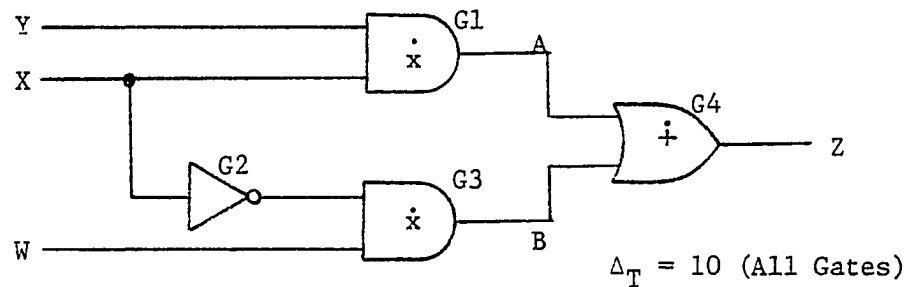
Static hazards also occur in B_{2^m} combinational circuits for the same classical reasons that they occur in binary combinational circuits, namely through an imbalance of signal delays on different paths of the logic circuit. To exemplify this situation for the $B_2 \times B_2$ system, a brief analysis will be given for the static-one-hazard case. The circuit operation of the example is slightly complicated by the inherent hazard condition of the AND and OR gates, and the inherent condition should be distinguished from the classical hazard conditions being examined. For the discussion, the four-valued $B_2 \times B_2$ circuit is considered as representing both a "left" and "right" binary circuit or machine.

Figure 4.2 shows a four-valued circuit that contains a static-one-hazard. The output of the circuit is given by the sum-of-products expression

$$Z = (X \dot{x} Y) \dot{+} (X' \dot{x} W) .$$

The hazard results because of the longer delay through gates G2 and G3 when the value of X changes from a 1 to 0 for either the left or right circuit. This has the effect of switching from the first to the second term in the output expression.

In the four-valued $B_2 \times B_2$ system there are several types of transitions in X that will exhibit the static-one-hazard for the left and/or right circuits. Three such changes are depicted in the simulation results in Figure 4.3. For this simulation $\Delta_T = 10$ time units for all gates, and the rise/fall time is 1 time unit. Initially, $W = X = Y = 3$ so that both the left and right binary circuits are held at 1 ($Z = 3$)



$$Z = (X \dot{\times} Y) \dot{+} (X' \dot{\times} W)$$

Figure 4.2. $B_2 \times B_2$ Circuit Containing Static-One-Hazard

by the first term of the output expression. The right circuit hazard is exhibited by X changing from 3 to 2 at Time 1. At Time 21 it is seen that Z dips momentarily to a value of 2 before returning to 3. This behavior shows that the right circuit switches from the first to second term of the output expression, but the left circuit remains unchanged. The X input then returns to a value of 3 at Time 50, but for this transition Z does not glitch since gate $G1$ is able to switch to a 3 before gate $G3$ (node B) drops to 0.

The left circuit static-one-hazard is slightly more complicated. At Time 100 X drops to a value of 1. Because X must pass through the value of 2 and since the gates are assumed to react to this

QUATERNARY LOGIC SIMULATOR - QLOSIM

SIHAZ.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE | |
|------|------|-------------|-----------|------------------------------------|
| 1 | X | 3 | 2 | } Right Circuit Static 1 Hazard |
| 11 | NX | 0 | 1 | |
| 11 | A | 3 | 2 | |
| 21 | B | 0 | 1 | |
| 21 | Z | 3 | 2 | |
| 31 | Z | 2 | 3 | |
| 50 | X | 2 | 3 | |
| 60 | NX | 1 | 0 | |
| 60 | A | 2 | 3 | |
| 70 | B | 1 | 0 | |
| 100 | X | 3 | 2 | } Left Circuit Static 1 Hazard |
| 101 | X | 2 | 1 | |
| 110 | NX | 0 | 1 | |
| 110 | A | 3 | 2 | |
| 111 | NX | 1 | 2 | |
| 111 | A | 2 | 1 | |
| 120 | B | 0 | 1 | |
| 120 | Z | 3 | 2 | |
| 121 | B | 1 | 2 | |
| 121 | Z | 2 | 1 | |
| 131 | Z | 1 | 2 | |
| 132 | Z | 2 | 3 | |
| 150 | X | 1 | 2 | |
| 151 | X | 2 | 3 | |
| 160 | NX | 2 | 1 | |
| 160 | A | 1 | 2 | |
| 161 | NX | 1 | 0 | |
| 161 | A | 2 | 3 | |
| 170 | B | 2 | 1 | |
| 170 | Z | 3 | 2 | |
| 171 | B | 1 | 0 | |
| 171 | Z | 2 | 3 | |

Figure 4.3. Simulation of $B_2 \times B_2$ Static-One-Hazard Condition - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

SIHAZ.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 200 | X | 3 | 2 |
| 201 | X | 2 | 1 |
| 202 | X | 1 | 0 |
| 210 | NX | 0 | 1 |
| 210 | A | 3 | 2 |
| 211 | NX | 1 | 2 |
| 211 | A | 2 | 1 |
| 212 | NX | 2 | 3 |
| 212 | A | 1 | 0 |
| 220 | B | 0 | 1 |
| 220 | Z | 3 | 2 |
| 221 | B | 1 | 2 |
| 221 | Z | 2 | 1 |
| 222 | B | 2 | 3 |
| 222 | Z | 1 | 0 |
| 230 | Z | 0 | 1 |
| 231 | Z | 1 | 2 |
| 232 | Z | 2 | 3 |
| 250 | X | 0 | 1 |
| 251 | X | 1 | 2 |
| 252 | X | 2 | 3 |
| 260 | NX | 3 | 2 |
| 260 | A | 0 | 1 |
| 261 | NX | 2 | 1 |
| 261 | A | 1 | 2 |
| 262 | NX | 1 | 0 |
| 262 | A | 2 | 3 |
| 270 | B | 3 | 2 |
| 271 | B | 2 | 1 |
| 272 | B | 1 | 0 |

} Left and Right
Circuit Static 1
Hazard

SIMULATION TERMINATED AT TIME 272

Figure 4.3. Simulation of $B_2 \times B_2$ Static-One-Hazard Condition - Part 2

value, both a left and right static-one-hazard condition occurs for Z. The right circuit condition at Time 120 actually results from the inherent static hazard of the four-valued G1 AND gate; however, the effect on Z is the same. At Time 132, the left circuit is held at 1 by term one of the output expression, and the right circuit is held at 1 by term two. When X returns to 3 at Time 150, gate G1 again reacts to the intermediate value of $X = 2$, resulting in a dip in the Z output.

A left and right circuit static-one-hazard condition occurs again when X changes from 3 to 0 beginning at Time 200. The value of Z is seen to drop to 0 at Time 222 before rising to 3 at Time 232. Both the left and right circuits are held at $Z = 1$ by the second term of the output expression. When X rises back to 3, no effect is shown in Z as both circuit halves switch back to the first term.

Correction of the static hazard conditions can be performed for B_{2m} combinational circuits in the same manner as for binary circuits. In the binary circuit additional intermediate terms are added to the output expression to hold the output at the proper value [36]. For the example above, the addition of the term $(W \dot{x} Y)$ to the expression will prevent the static-one-hazard conditions. The resulting circuit contains an additional gate, and the OR gate has three inputs instead of two. Simulation of this circuit is possible, but it is not necessary to show the results since, as expected, the new term $(Y \dot{x} W)$ will hold Z to a value of 3 for all of the previous input conditions. Thus, the static-one-hazards are eliminated.

Further classification of B_{2m} combinational circuit behavior can be made, but this is beyond the scope necessary for this work.

Results presented in the next chapter permit freedom from transient or hazardous conditions that occur in the combinational circuits. These conditions normally have their greatest effect on sequential circuits where they can falsely trigger memory elements and result in incorrect circuit operation. By and large, transient and hazard conditions will be of only passing concern because of the memory elements that are developed. With this in mind, the combinational circuit design specification is considered next.

Design Methodology

The binary adder is a highly used combinational circuit and is necessary in some form in practically every computer central processing unit since it comprises a major portion of the arithmetic and logic unit. In this section, a one bit $B_2 \times B_2$ multiplexed full adder will be designed. This example will show the use of binary combinational design methods for B_{2^m} combinational circuits. The binary/quaternary encoder and decoder circuits discussed in Chapter II are not given but can be added to this or any other circuit as necessary.

In the binary full adder, designate the inputs as A , B and C_I , where A and B are the single bit binary numbers to be added and C_I is the carry input. The outputs of the circuit are the sum S and the carry-out C_O . The procedure used for the circuit design will be the Karnaugh map method, but other binary procedures, such as the Quine-McCluskey method [36], can also be employed. The implementation will be straightforward and no further attempts at minimality or minimum cost circuits will be made.

The Karnaugh map for the sum output is shown in Figure 4.4(a).

Circling on 1's yields the sum-of-products expression for S,

$$S = (A' \cdot B \cdot C') + (A \cdot B' \cdot C') + (A' \cdot B' \cdot C) + (A \cdot B \cdot C) .$$

Similarly, the Karnaugh map for the carry-out is shown in Figure 4.4(b) and yields the sum-of-products expression

$$C_0 = (A \cdot B) + (B \cdot C) + (A \cdot C) .$$

Implementing these results with binary gates, the circuit in Figure 4.5 is obtained.

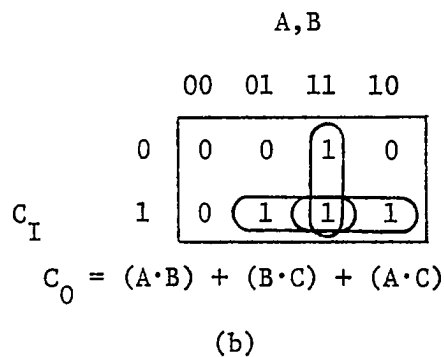
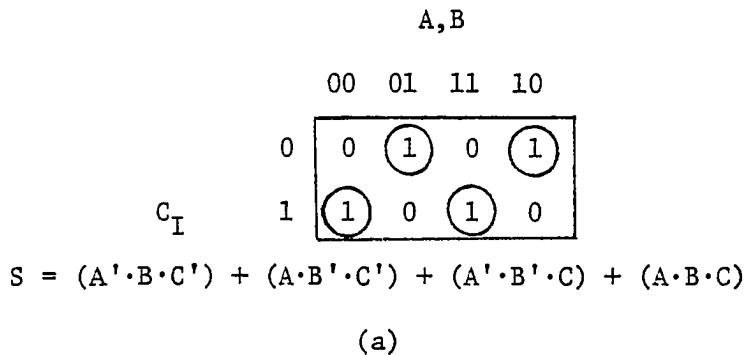


Figure 4.4. Full Adder Karnaugh Maps

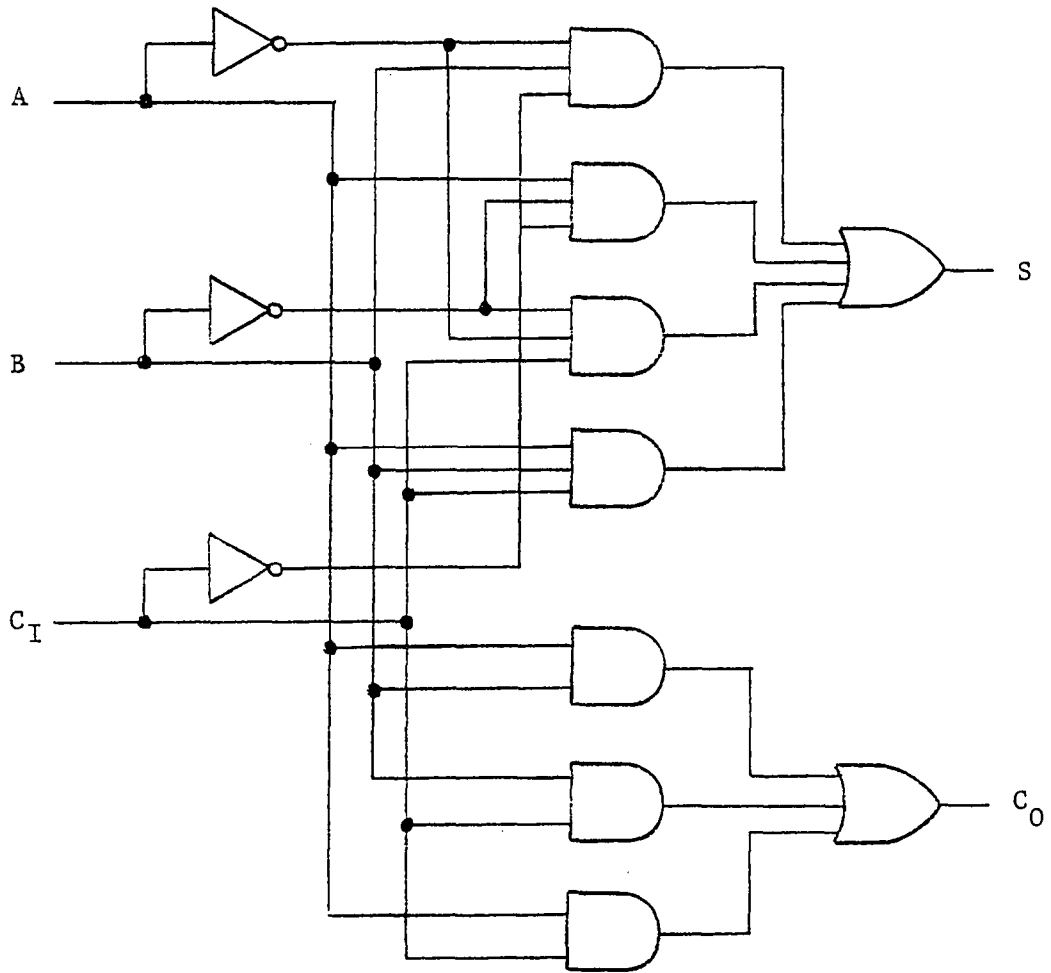


Figure 4.5. Binary Full Adder Circuit

Designing the state integrated and multiplexed full adder does not require a new synthesis procedure. To convert the binary full adder to a $B_2 \times B_2$ full adder, the binary gates are replaced on a one-to-one basis with the corresponding four-valued Boolean gates. The switching functions for the S and C_0 outputs are identical to the binary expressions except for notation of the AND and OR operations. All circuit interconnects remain the same, but now each lead in the circuit carries a multiplexed four-valued signal with each signal viewed as a 2-tuple. The circuit inputs and outputs are four-valued and simultaneously represent the inputs and output for two binary full adders. Likewise, the carry-out operates in the multiplexed fashion; that is, the signal simultaneously represents the carry-out for both binary adders. Thus, if $C_0 = 01$, the carry-out for the left adder is 0 and the carry-out for the right adder is 1. In other words, the $B_2 \times B_2$ full adder operates as if two binary adders were each operating independently of the other.

Simulation of the four-valued full adder shows that the circuit performs as expected. A simulator output for selected input conditions is given in Figure 4.6. Initial values for A , B , and C_I are zero. The first stable sum output occurs at Time 32. The value of S represents the sum of $A = 1$ and $B = 2$ with $C_I = 0$. The C_0 output is zero for these input values. The sum output value of 3 is interpreted as a value of one for both the left and right logical binary adder circuits. At Time 200, B changes to 0, and the S output changes to 1. The carry input, C_I , is asserted for the right logical circuit at Time 300 which changes the sum to zero but produces a carry-out of value one. The $C_0 = 1$

QUATERNARY LOGIC SIMULATOR - QLOSIM

FADDER.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | A | 0 | 1 |
| 1 | B | 0 | 1 |
| 2 | B | 1 | 2 |
| 21 | CO | 0 | 1 |
| 21 | S | 0 | 1 |
| 22 | CO | 1 | 0 |
| 22 | S | 1 | 2 |
| 23 | S | 2 | 3 |
| 31 | S | 3 | 2 |
| 32 | S | 2 | 3 |
| 200 | B | 2 | 1 |
| 201 | B | 1 | 0 |
| 220 | CO | 0 | 1 |
| 220 | S | 3 | 2 |
| 221 | CO | 1 | 0 |
| 221 | S | 2 | 1 |
| 230 | S | 1 | 0 |
| 231 | S | 0 | 1 |
| 300 | CI | 0 | 1 |
| 320 | CO | 0 | 1 |
| 330 | S | 1 | 0 |
| 400 | A | 1 | 2 |
| 420 | CO | 1 | 0 |
| 420 | S | 0 | 1 |
| 421 | S | 1 | 2 |
| 430 | S | 2 | 3 |
| 500 | CI | 1 | 2 |
| 520 | CO | 0 | 1 |
| 520 | S | 3 | 2 |
| 521 | CO | 1 | 2 |
| 530 | S | 2 | 1 |
| 531 | S | 1 | 0 |
| 600 | A | 2 | 3 |
| 600 | B | 0 | 1 |
| 600 | CI | 2 | 3 |
| 601 | B | 1 | 2 |
| 602 | B | 2 | 3 |
| 620 | CO | 2 | 3 |
| 620 | S | 0 | 1 |
| 621 | S | 1 | 2 |
| 622 | S | 2 | 3 |

SIMULATION TERMINATED AT TIME 622

Figure 4.6. Simulation of Four-Valued Multiplexed Full Adder

represents a carry-out for the right adder circuit. Subsequent input changes occur at Time 400, 500, and 600 and produce varying but similar results.

Close scrutiny of the simulator output for the full adder shows that the outputs of the circuit indicate the presence of transient or hazardous conditions. This is especially evident in the C_0 output. Since the circuit does not contain static hazards, the hazard conditions observed must be tolerated. As mentioned previously, these conditions are significant only if they falsely trigger a subsequent circuit or if the circuit output is used prior to achieving full stability.

Combinational Circuit Examples

In this section, simulator results for two combinational state integrated and multiplexed circuits will be given. Both circuits are important for the binary case and occur quite frequently in digital system design. The first is a four-to-one multiplexer circuit that gives interesting results for the state integrated and multiplexed case. The second circuit is the extension of the four-valued state integrated and multiplexed full adder to form a four bit state integrated and multiplexed ripple-carry adder.

Figure 4.7 shows the circuit of the four-to-one state integrated and multiplexed multiplexer circuit. The circuit is derived directly from one-half of a 74153 binary integrated circuit [37]. For the binary case the circuit operation is simple. The four data inputs to the circuit are 1C0, 1C1, 1C2, and 1C3. The A and B inputs select one of the data inputs whose value is reflected on the 1Y output, provided the enable or E1 input is zero.

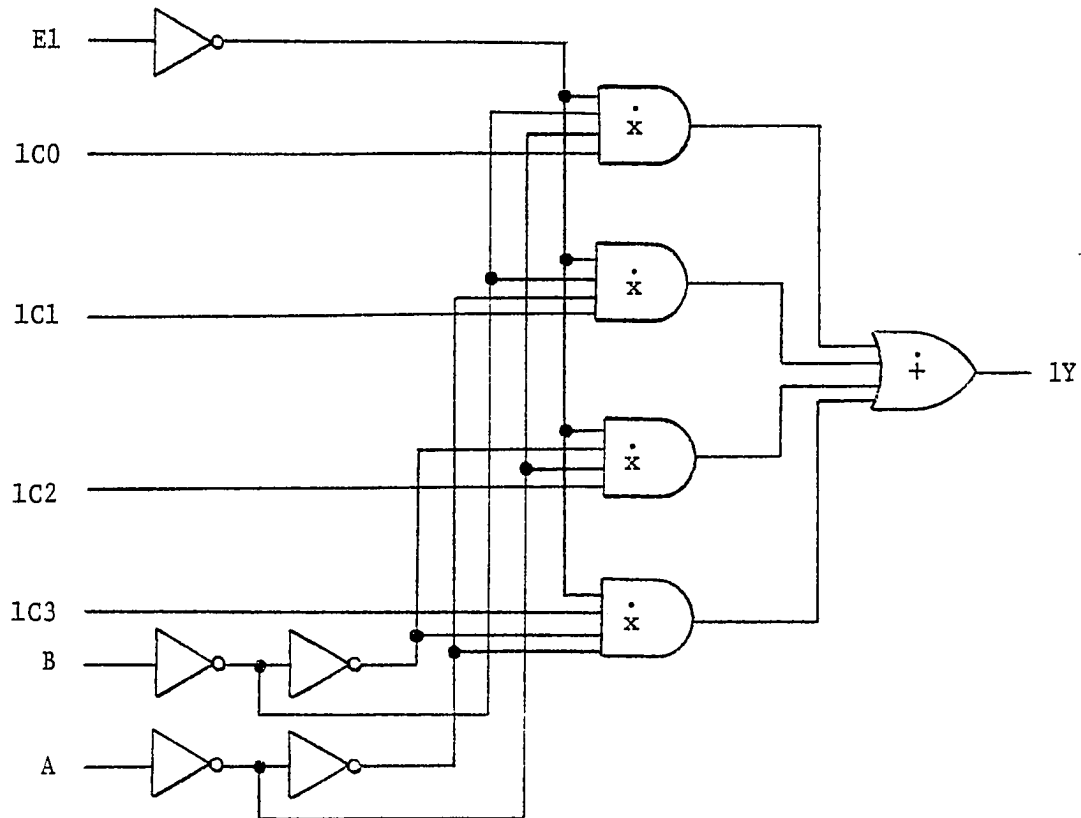


Figure 4.7. Four-Valued State Integrated and Multiplexed Four-to-One Multiplexer Circuit

For the four-valued state integrated and multiplexed case, the the multiplexer circuit operation is more complex. First, the data inputs represent data values for the logical left and right circuit inputs. Second, the A and B inputs simultaneously select a data input for the left circuit and a data input for the right circuit. Finally, the E1 input can selectively enable or disable the left, right, or both logical circuits.

Table 4.1 summarizes the four-valued multiplexer operation for the various A and B input values. It is assumed in the table that the E1 input will have a value of zero in order to enable both the left and right circuits. As shown by Table 4.1, it is possible to select any combination of data inputs for the left and right circuits by proper application of multiplexed A and B inputs. Figure 4.8 gives a simulator output which shows the various A and B input combinations and the

Table 4.1. Four-Valued Four-to-One Multiplexer Truth Table

| <u>SELECTOR INPUTS</u> | | <u>SELECTED INPUTS</u> | |
|------------------------|----------|------------------------|-------------------|
| <u>B</u> | <u>A</u> | <u>LEFT CKT.</u> | <u>RIGHT CKT.</u> |
| 0 | 0 | 1C0 | 1C0 |
| 0 | 1 | 1C0 | 1C1 |
| 0 | 2 | 1C1 | 1C0 |
| 0 | 3 | 1C1 | 1C1 |
| 1 | 0 | 1C0 | 1C2 |
| 1 | 1 | 1C0 | 1C3 |
| 1 | 2 | 1C1 | 1C2 |
| 1 | 3 | 1C1 | 1C3 |
| 2 | 0 | 1C2 | 1C0 |
| 2 | 1 | 1C2 | 1C1 |
| 2 | 2 | 1C3 | 1C0 |
| 2 | 3 | 1C3 | 1C1 |
| 3 | 0 | 1C2 | 1C2 |
| 3 | 1 | 1C2 | 1C3 |
| 3 | 2 | 1C3 | 1C2 |
| 3 | 3 | 1C3 | 1C3 |

resulting 1Y output. At the start of the simulation the four data inputs, 1C0 through 1C3, are set to logic values 0 through 3, respectively. All possible values for A and B are cycled through the circuit. Using Table 4.1, it can be verified that the final circuit responses are correct for each input combination. Beginning at Time 2000 in the simulation the effect of the E1 input is demonstrated. Note that E1 = 1 disables the right logical circuit, E1 = 2 disables the left logical circuit, and E1 = 3 disables both.

QUATERNARY LOGIC SIMULATOR - QLOSIM

MUX.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | E1 | 3 | 2 |
| 1 | 1C1 | 0 | 1 |
| 1 | 1C2 | 0 | 1 |
| 1 | 1C3 | 0 | 1 |
| 2 | E1 | 2 | 1 |
| 2 | 1C2 | 1 | 2 |
| 2 | 1C3 | 1 | 2 |
| 3 | 1C3 | 2 | 3 |
| 3 | E1 | 1 | 0 |
| 100 | A | 0 | 1 |
| 140 | 1Y | 0 | 1 |
| 200 | A | 1 | 2 |
| 240 | 1Y | 1 | 0 |
| 300 | A | 2 | 3 |
| 340 | 1Y | 0 | 1 |
| 400 | B | 0 | 1 |
| 400 | A | 3 | 2 |
| 401 | A | 2 | 1 |
| 402 | A | 1 | 0 |
| 430 | 1Y | 1 | 0 |
| 441 | 1Y | 0 | 1 |
| 442 | 1Y | 1 | 0 |
| 500 | A | 0 | 1 |
| 540 | 1Y | 0 | 1 |
| 600 | A | 1 | 2 |
| 640 | 1Y | 1 | 0 |
| 700 | A | 2 | 3 |
| 740 | 1Y | 0 | 1 |
| 800 | B | 1 | 2 |
| 800 | A | 3 | 2 |
| 801 | A | 2 | 1 |
| 802 | A | 1 | 0 |
| 840 | 1Y | 1 | 2 |

Figure 4.8. Four-Valued State Integrated and Multiplexed Four-to-One Multiplexer Simulation - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

MUX.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 841 | 1Y | 2 | 3 |
| 842 | 1Y | 3 | 2 |
| 900 | A | 0 | 1 |
| 940 | 1Y | 2 | 3 |
| 1000 | A | 1 | 2 |
| 1030 | 1Y | 3 | 2 |
| 1031 | 1Y | 2 | 1 |
| 1041 | 1Y | 1 | 2 |
| 1100 | A | 2 | 3 |
| 1140 | 1Y | 2 | 3 |
| 1200 | B | 2 | 3 |
| 1200 | A | 3 | 2 |
| 1201 | A | 2 | 1 |
| 1202 | A | 1 | 0 |
| 1230 | 1Y | 3 | 2 |
| 1231 | 1Y | 2 | 3 |
| 1232 | 1Y | 3 | 2 |
| 1241 | 1Y | 2 | 3 |
| 1242 | 1Y | 3 | 2 |
| 1300 | A | 0 | 1 |
| 1340 | 1Y | 2 | 3 |
| 1400 | A | 1 | 2 |
| 1430 | 1Y | 3 | 2 |
| 1431 | 1Y | 2 | 1 |
| 1440 | 1Y | 1 | 2 |
| 1500 | A | 2 | 3 |
| 1540 | 1Y | 2 | 3 |
| 2000 | E1 | 0 | 1 |
| 2030 | 1Y | 3 | 2 |
| 2200 | E1 | 1 | 2 |
| 2230 | 1Y | 2 | 1 |
| 2400 | E1 | 2 | 3 |
| 2430 | 1Y | 1 | 0 |

SIMULATION TERMINATED AT TIME 2430

Figure 4.8. Four-Valued State Integrated and Multiplexed Four-to-One Multiplexer Simulation - Part 2

The second four-valued combinational circuit investigated is the state integrated and multiplexed four-bit ripple-carry adder. This circuit is formed by cascading four state integrated and multiplexed full adder circuits. The full adder operation was demonstrated in the previous section. Figure 4.9 shows the interconnect arrangement for the four-bit ripple-carry adder. CI_0 is the external carry input, and CO_3 is the final carry output. The S outputs are the sum of the A and B inputs.

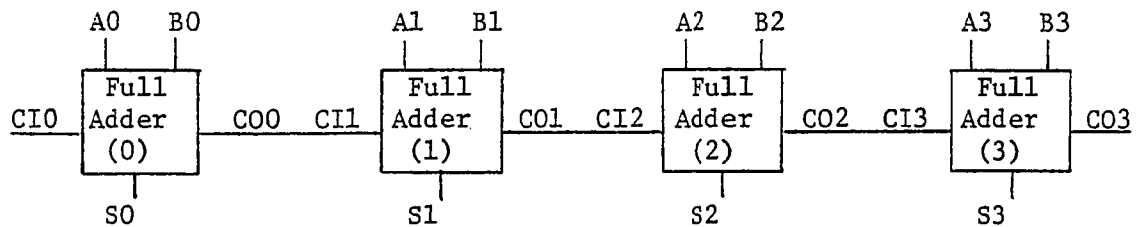


Figure 4.9. State Integrated and Multiplexed Four-Valued Ripple-Carry Adder

As implemented here, the ripple-carry adder contains 48 gates and is simulated with all gates having a transport delay of 10 time units. Since all circuit input combinations result in an extremely large number, only selected input values will be simulated. Table 4.2 shows several values of inputs to the ripple-carry adder and the expected circuit outputs. Figure 4.10 presents the simulator output. Simulation of these values is performed at the time given in Table 4.2 with the CI_0 input applied at different times for each A and B input combination. These particular values highlight the propagation of the interstage carry values for both the left and right logical adders.

Table 4.2. Selected Inputs and Expected Results for the State Integrated and Multiplexed Ripple-Carry Adder

| <u>A3-A0</u> | <u>B3-B0</u> | <u>C10</u> | <u>S3-S0</u> | <u>C03</u> | <u>TIME</u> |
|--------------|--------------|------------|--------------|------------|-------------|
| 0101 | 0101 | 0 | 1010 | 0 | 1 |
| 0101 | 0101 | 1 | 1011 | 0 | 250 |
| 1010 | 1010 | 0 | 0100 | 1 | 500 |
| 1010 | 1010 | 0 | 0101 | 1 | 750 |
| 0101 | 1010 | 0 | 1111 | 0 | 1000 |
| 0101 | 1010 | 1 | 0000 | 1 | 1250 |
| 0202 | 0202 | 0 | 2020 | 0 | 1500 |
| 0202 | 0202 | 2 | 2022 | 0 | 1750 |
| 2020 | 2020 | 0 | 0200 | 2 | 2000 |
| 2020 | 2020 | 2 | 0202 | 2 | 2250 |
| 0202 | 2020 | 0 | 2222 | 0 | 2500 |
| 0202 | 2020 | 2 | 0000 | 2 | 2750 |
| 0101 | 2020 | 0 | 2121 | 0 | 3000 |
| 0101 | 2020 | 1 | 2131 | 0 | 3250 |
| 1010 | 2020 | 0 | 3030 | 0 | 3500 |
| 1010 | 2020 | 2 | 3032 | 0 | 3750 |
| 3030 | 3030 | 0 | 0300 | 3 | 4000 |
| 3030 | 3030 | 3 | 0303 | 3 | 4250 |
| 0303 | 0303 | 0 | 3030 | 0 | 4500 |
| 0303 | 0303 | 3 | 3033 | 0 | 4750 |

The two previous examples indicate that four-valued Boolean combinational circuits can be derived directly from binary circuits. This ability greatly simplifies the design of state integrated and multiplexed combinational circuits. The behavior of these four-valued circuits, under the assumptions given here, is more complicated than the binary case, but it should be possible, using the quaternary simulator, to predict the circuit response. Combinational circuit behavior is most acute when the circuit is used within a sequential circuit. This topic is covered in the following chapters.

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | A0 | 0 | 1 |
| 1 | B0 | 0 | 1 |
| 1 | A2 | 0 | 1 |
| 1 | B2 | 0 | 1 |
| 21 | S0 | 0 | 1 |
| 21 | S2 | 0 | 1 |
| 31 | S0 | 1 | 0 |
| 31 | S2 | 1 | 0 |
| 41 | S1 | 0 | 1 |
| 41 | S3 | 0 | 1 |
| 250 | C10 | 0 | 1 |
| 270 | S0 | 0 | 1 |
| 500 | A0 | 1 | 0 |
| 500 | B0 | 1 | 0 |
| 500 | C10 | 1 | 0 |
| 500 | A1 | 0 | 1 |
| 500 | B1 | 0 | 1 |
| 500 | A2 | 1 | 0 |
| 500 | B2 | 1 | 0 |
| 500 | A3 | 0 | 1 |
| 500 | B3 | 0 | 1 |
| 520 | S0 | 1 | 0 |
| 520 | C03 | 0 | 1 |
| 540 | S1 | 1 | 0 |
| 540 | S2 | 0 | 1 |
| 540 | S3 | 1 | 0 |
| 750 | C10 | 0 | 1 |
| 770 | S0 | 0 | 1 |
| 1000 | A0 | 0 | 1 |
| 1000 | C10 | 1 | 0 |
| 1000 | A1 | 1 | 0 |
| 1000 | A2 | 0 | 1 |
| 1000 | A3 | 1 | 0 |
| 1020 | S0 | 1 | 0 |
| 1020 | C03 | 1 | 0 |
| 1030 | S0 | 0 | 1 |
| 1030 | S1 | 0 | 1 |
| 1030 | S2 | 1 | 0 |
| 1030 | S3 | 0 | 1 |
| 1040 | C03 | 0 | 1 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDER.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1050 | S2 | 0 | 1 |
| 1050 | S3 | 1 | 0 |
| 1060 | CO3 | 1 | 0 |
| 1070 | S3 | 0 | 1 |
| 1250 | CI0 | 0 | 1 |
| 1280 | S0 | 1 | 0 |
| 1300 | S1 | 1 | 0 |
| 1320 | S2 | 1 | 0 |
| 1330 | CO3 | 0 | 1 |
| 1340 | S3 | 1 | 0 |
| 1500 | A0 | 1 | 2 |
| 1500 | B0 | 0 | 1 |
| 1500 | CI0 | 1 | 0 |
| 1500 | B1 | 1 | 0 |
| 1500 | A2 | 1 | 2 |
| 1500 | B2 | 0 | 1 |
| 1500 | B3 | 1 | 0 |
| 1501 | B0 | 1 | 2 |
| 1501 | B2 | 1 | 2 |
| 1520 | S0 | 0 | 1 |
| 1520 | S2 | 0 | 1 |
| 1520 | CO3 | 1 | 0 |
| 1521 | S0 | 1 | 2 |
| 1521 | S2 | 1 | 2 |
| 1530 | S0 | 2 | 3 |
| 1530 | S1 | 0 | 1 |
| 1530 | S2 | 2 | 3 |
| 1530 | S3 | 0 | 1 |
| 1531 | S0 | 3 | 2 |
| 1531 | S2 | 3 | 2 |
| 1532 | S0 | 2 | 1 |
| 1532 | S2 | 2 | 1 |
| 1533 | S0 | 1 | 0 |
| 1540 | S1 | 1 | 0 |
| 1540 | S2 | 1 | 0 |
| 1541 | S1 | 0 | 1 |
| 1542 | S1 | 1 | 2 |
| 1542 | S3 | 1 | 2 |
| 1750 | CI0 | 0 | 1 |
| 1751 | CI0 | 1 | 2 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1770 | S0 | 0 | 1 |
| 1772 | S0 | 1 | 2 |
| 1791 | S1 | 2 | 3 |
| 1792 | S1 | 3 | 2 |
| 2000 | A0 | 2 | 1 |
| 2000 | B0 | 2 | 1 |
| 2000 | CI0 | 2 | 1 |
| 2000 | A1 | 0 | 1 |
| 2000 | B1 | 0 | 1 |
| 2000 | A2 | 2 | 1 |
| 2000 | B2 | 2 | 1 |
| 2000 | A3 | 0 | 1 |
| 2000 | B3 | 0 | 1 |
| 2001 | A0 | 1 | 0 |
| 2001 | B0 | 1 | 0 |
| 2001 | CI0 | 1 | 0 |
| 2001 | A1 | 1 | 2 |
| 2001 | B1 | 1 | 2 |
| 2001 | A2 | 1 | 0 |
| 2001 | B2 | 1 | 0 |
| 2001 | A3 | 1 | 2 |
| 2001 | B3 | 1 | 2 |
| 2020 | S0 | 2 | 1 |
| 2020 | S1 | 2 | 3 |
| 2020 | S2 | 0 | 1 |
| 2020 | CO3 | 0 | 1 |
| 2020 | S3 | 2 | 3 |
| 2021 | S0 | 1 | 0 |
| 2021 | S2 | 1 | 0 |
| 2021 | CO3 | 1 | 2 |
| 2022 | S1 | 3 | 2 |
| 2022 | S3 | 3 | 2 |
| 2031 | S1 | 2 | 3 |
| 2031 | S3 | 2 | 3 |
| 2032 | S3 | 3 | 2 |
| 2040 | S1 | 2 | 1 |
| 2040 | S2 | 0 | 1 |
| 2040 | CO3 | 2 | 3 |
| 2040 | S3 | 2 | 1 |
| 2041 | S1 | 1 | 0 |
| 2041 | S2 | 1 | 2 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 3

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDER.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 2041 | CO3 | 3 | 2 |
| 2041 | S3 | 1 | 0 |
| 2060 | S2 | 2 | 3 |
| 2061 | S2 | 3 | 2 |
| 2250 | CI0 | 0 | 1 |
| 2251 | CI0 | 1 | 2 |
| 2270 | S0 | 0 | 1 |
| 2271 | S0 | 1 | 2 |
| 2500 | A0 | 0 | 1 |
| 2500 | CI0 | 2 | 1 |
| 2500 | A1 | 2 | 1 |
| 2500 | A2 | 0 | 1 |
| 2500 | A3 | 2 | 1 |
| 2501 | A0 | 1 | 2 |
| 2501 | CI0 | 1 | 0 |
| 2501 | A1 | 1 | 0 |
| 2501 | A2 | 1 | 2 |
| 2501 | A3 | 1 | 0 |
| 2520 | S0 | 2 | 1 |
| 2520 | S1 | 0 | 1 |
| 2520 | S2 | 2 | 3 |
| 2520 | CO3 | 2 | 1 |
| 2520 | S3 | 0 | 1 |
| 2521 | S0 | 1 | 0 |
| 2521 | S1 | 1 | 0 |
| 2521 | S2 | 3 | 2 |
| 2521 | CO3 | 1 | 0 |
| 2521 | S3 | 1 | 0 |
| 2530 | S0 | 0 | 1 |
| 2530 | S1 | 0 | 1 |
| 2530 | S3 | 0 | 1 |
| 2531 | S0 | 1 | 2 |
| 2531 | S1 | 1 | 2 |
| 2531 | S2 | 2 | 1 |
| 2531 | S3 | 1 | 2 |
| 2532 | S2 | 1 | 0 |
| 2540 | S1 | 2 | 3 |
| 2540 | S2 | 0 | 1 |
| 2541 | S1 | 3 | 2 |
| 2541 | S2 | 1 | 0 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 4

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 2541 | S3 | 2 | 3 |
| 2542 | CO3 | 0 | 1 |
| 2542 | S3 | 3 | 2 |
| 2543 | CO3 | 1 | 2 |
| 2550 | S2 | 0 | 1 |
| 2551 | S2 | 1 | 2 |
| 2552 | S3 | 2 | 1 |
| 2553 | S3 | 1 | 0 |
| 2560 | CO3 | 2 | 1 |
| 2560 | S3 | 0 | 1 |
| 2561 | CO3 | 1 | 0 |
| 2561 | S3 | 1 | 0 |
| 2570 | S3 | 0 | 1 |
| 2571 | S3 | 1 | 2 |
| 2750 | CI0 | 0 | 1 |
| 2751 | CI0 | 1 | 2 |
| 2770 | S0 | 2 | 3 |
| 2771 | S0 | 3 | 2 |
| 2781 | S0 | 2 | 1 |
| 2782 | S0 | 1 | 0 |
| 2791 | S1 | 2 | 3 |
| 2792 | S1 | 3 | 2 |
| 2802 | S1 | 2 | 1 |
| 2803 | S1 | 1 | 0 |
| 2812 | S2 | 2 | 3 |
| 2813 | S2 | 3 | 2 |
| 2823 | S2 | 2 | 1 |
| 2824 | S2 | 1 | 0 |
| 2833 | S3 | 2 | 3 |
| 2834 | CO3 | 0 | 1 |
| 2834 | S3 | 3 | 2 |
| 2835 | CO3 | 1 | 2 |
| 2844 | S3 | 2 | 1 |
| 2845 | S3 | 1 | 0 |
| 3000 | A0 | 2 | 1 |
| 3000 | CI0 | 2 | 1 |
| 3000 | A2 | 2 | 1 |
| 3001 | CI0 | 1 | 0 |
| 3020 | S0 | 0 | 1 |
| 3020 | S2 | 0 | 1 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 5

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 3030 | S0 | 1 | 0 |
| 3031 | S2 | 1 | 2 |
| 3031 | S0 | 0 | 1 |
| 3032 | S2 | 2 | 3 |
| 3040 | S1 | 0 | 1 |
| 3040 | CO3 | 2 | 1 |
| 3040 | S3 | 0 | 1 |
| 3041 | CO3 | 1 | 0 |
| 3041 | S3 | 1 | 0 |
| 3041 | S1 | 1 | 0 |
| 3050 | S1 | 0 | 1 |
| 3050 | S3 | 0 | 1 |
| 3051 | S3 | 1 | 2 |
| 3051 | S1 | 1 | 2 |
| 3060 | S2 | 3 | 2 |
| 3061 | S2 | 2 | 1 |
| 3070 | S2 | 1 | 0 |
| 3071 | S2 | 0 | 1 |
| 3080 | S3 | 2 | 3 |
| 3081 | S3 | 3 | 2 |
| 3250 | CI0 | 0 | 1 |
| 3280 | S0 | 1 | 0 |
| 3290 | S1 | 2 | 3 |
| 3500 | A0 | 1 | 0 |
| 3500 | CI0 | 1 | 0 |
| 3500 | A1 | 0 | 1 |
| 3500 | A2 | 1 | 0 |
| 3500 | A3 | 0 | 1 |
| 3520 | S2 | 1 | 0 |
| 3520 | S3 | 2 | 3 |
| 3530 | S1 | 3 | 2 |
| 3540 | S2 | 0 | 1 |
| 3550 | S1 | 2 | 3 |
| 3560 | S2 | 1 | 0 |
| 3750 | CI0 | 0 | 1 |
| 3751 | CI0 | 1 | 2 |
| 3770 | S0 | 0 | 1 |
| 3771 | S0 | 1 | 2 |
| 4000 | CI0 | 2 | 1 |
| 4000 | A1 | 1 | 2 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 6

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 4000 | B1 | 2 | 3 |
| 4000 | A3 | 1 | 2 |
| 4000 | B3 | 2 | 3 |
| 4001 | C10 | 1 | 0 |
| 4001 | A1 | 2 | 3 |
| 4020 | S0 | 2 | 1 |
| 4020 | S1 | 3 | 2 |
| 4020 | S3 | 3 | 2 |
| 4020 | CO3 | 0 | 1 |
| 4021 | S0 | 1 | 0 |
| 4021 | S1 | 2 | 3 |
| 4021 | CO3 | 1 | 2 |
| 4021 | S3 | 2 | 3 |
| 4022 | CO3 | 2 | 3 |
| 4030 | S1 | 3 | 2 |
| 4030 | S3 | 3 | 2 |
| 4031 | S1 | 2 | 1 |
| 4031 | S3 | 2 | 1 |
| 4032 | S1 | 1 | 0 |
| 4032 | S3 | 1 | 0 |
| 4040 | S2 | 0 | 1 |
| 4041 | S2 | 1 | 2 |
| 4042 | S2 | 2 | 3 |
| 4250 | C10 | 0 | 1 |
| 4251 | C10 | 1 | 2 |
| 4252 | C10 | 2 | 3 |
| 4270 | S0 | 0 | 1 |
| 4271 | S0 | 1 | 2 |
| 4272 | S0 | 2 | 3 |
| 4500 | A0 | 0 | 1 |
| 4500 | B0 | 0 | 1 |
| 4500 | C10 | 3 | 2 |
| 4500 | A1 | 3 | 2 |
| 4500 | B1 | 3 | 2 |
| 4500 | A2 | 0 | 1 |
| 4500 | B2 | 0 | 1 |
| 4500 | A3 | 3 | 2 |
| 4500 | B3 | 3 | 2 |
| 4501 | A0 | 1 | 2 |

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 7

QUATERNARY LOGIC SIMULATOR - QLOSIM

RADDR.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 4501 | B0 | 1 | 2 |
| 4501 | C10 | 2 | 1 |
| 4501 | A1 | 2 | 1 |
| 4501 | B1 | 2 | 1 |
| 4501 | A2 | 1 | 2 |
| 4501 | B2 | 1 | 2 |
| 4501 | A3 | 2 | 1 |
| 4501 | B3 | 2 | 1 |
| 4502 | A0 | 2 | 3 |
| 4502 | B0 | 2 | 3 |
| 4502 | C10 | 1 | 0 |
| 4502 | A1 | 1 | 0 |
| 4502 | B1 | 1 | 0 |
| 4502 | A2 | 2 | 3 |
| 4502 | B2 | 2 | 3 |
| 4502 | A3 | 1 | 0 |
| 4502 | B3 | 1 | 0 |
| 4520 | S0 | 3 | 2 |
| 4520 | C03 | 3 | 2 |
| 4521 | S0 | 2 | 1 |
| 4521 | C03 | 2 | 1 |
| 4522 | S0 | 1 | 0 |
| 4522 | C03 | 1 | 0 |
| 4540 | S1 | 0 | 1 |
| 4540 | S2 | 3 | 2 |
| 4540 | S3 | 0 | 1 |
| 4541 | S1 | 1 | 2 |
| 4541 | S2 | 2 | 1 |
| 4541 | S3 | 1 | 2 |
| 4542 | S1 | 2 | 3 |
| 4542 | S2 | 1 | 0 |
| 4542 | S3 | 2 | 3 |
| 4750 | C10 | 0 | 1 |
| 4751 | C10 | 1 | 2 |
| 4752 | C10 | 2 | 3 |
| 4770 | S0 | 0 | 1 |
| 4771 | S0 | 1 | 2 |
| 4772 | S0 | 2 | 3 |

SIMULATION TERMINATED AT TIME 4772

Figure 4.10. Four-Valued State Integrated and Multiplexed
Ripple-Carry Adder Simulation - Part 8

CHAPTER V

MEMORY ELEMENT DEVELOPMENT

Behavior Model

In order to construct state integrated and multiplexed sequential circuits, a memory element capable of simultaneously storing the state for two or more binary machines must be developed. The logical structure of such a device will be investigated. The basic operation of the memory element must allow each machine to set and clear its logical part of the device independently of the other machines. The memory element will be multiple-valued, and previous work dealing with multistables is required for the development of the state integrated memory element.

Wills [19] has proposed a behavioral model for multiple-valued memory elements or r-flops. The model, which is applicable to a wide variety of multistables, depicts the r-flop as a device which:

1. Is defined for any $r \geq 2$;
2. Is capable of remaining in any one of r discrete states when no external stimuli are applied;

3. Can be described by some deterministic next-state and/or next-output function or table;
4. Changes to its proper next-state value in a manner that can be described as either leading edge, level, trailing edge, or master-slave triggered;
5. Presents one or more outputs that change from their present-state to their next-state values in a monotonic, non-increasing (or non-decreasing), non-return-to-zero fashion;
6. Can be driven from any one of its stable states to any other stable state by the application of exactly one input change (and clocking cycle, for synchronous designs);
7. Responds to intermediate values of input and/or clock signals in such a fashion that is rise or fall time independent (within practical limits).

Achieving these characteristics in a device is highly desirable since they are closely related to the behavior of binary flip-flops. As suggested by Wills, the similarity of multiple-valued devices to the binary devices is highly desirable if the circuit design techniques of the binary system can also be extended to the multiple-valued case. We, therefore, wish to maintain this idea in the development for the B_{2m} state integrated memory devices and sequential circuits.

Multiple-Valued Storage Elements

The basic cross-coupled RS r-flop studied by several authors [15,16,19] was given in Chapter III as a simulation example. Figure 5.1

shows the same circuit again along with an alternate design using $\overline{\text{MIN}}$ gates. These circuits conform nicely to the behavioral model, but it should be noted that the circuit is asynchronous, so consequently the R or S inputs must not change until the circuit is stable. This restriction does not disallow multiple-valued changes, but it can be shown for the four-valued case that transient inputs can cause circuit oscillation. This result can be verified by pencil-and-paper or computer simulation with the $\overline{\text{MAX}}$ r-flop reset, $R = S = 0$, and allowing an S transition from 0 to 2 and back to 0 before the circuit stabilizes in state $Q = 2$. This behavior was not previously reported by Wills [19].

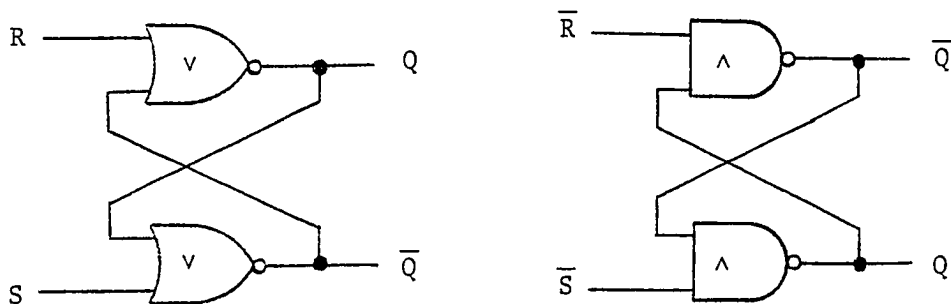
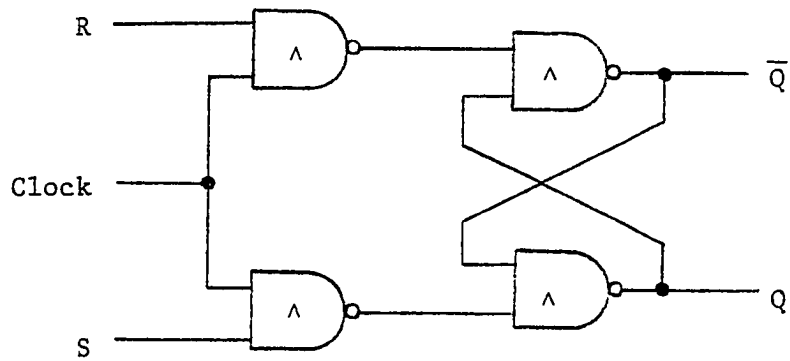
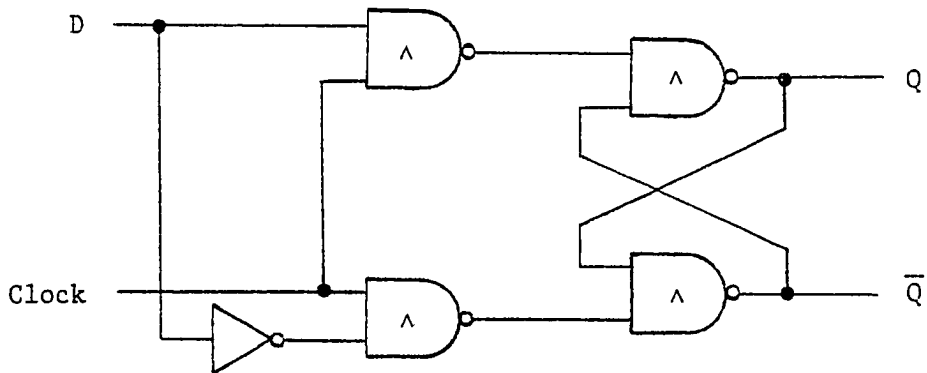


Figure 5.1. RS r-Flop Circuits

The addition of a clock signal allows the construction of the clocked RS r-flop and the D-type r-flop. Both designs are given in Figure 5.2. In general, clocked r-flops operate with a binary clock. For the four-valued case, the clock is active or asserted only at the logic level of 3. The inactive clock level is 0, and the logic levels 1 and 2 should be transition levels only to avoid improper r-flop operation. This action is true for the r-flops shown in Figure 5.2.



(a) RS Type



(b) D Type

Figure 5.2. Clocked Four-Valued r-Flops

Other r-flops have been studied [17,18,19], but those given thus far are sufficient for the purposes here.

State Integrated Storage Elements

The RS r-flop operation necessary for state integrated and multiplexed sequential circuits is very similar to the RS r-flop given above. The next-state table for the four-valued state integrated r-flop, however, differs slightly and is given in Figure 5.3. The four-valued state integrated RS r-flop simultaneously stores the state for both the left and right binary machines and must respond to the R and S inputs accordingly. For convenience, the next-state table uses the B_4 elements, although the action represents the $B_2 \times B_2$ system. Dashed entries represent ambiguous or undesirable input conditions analogous to the binary case. To distinguish the state integrated r-flop from previously studied types, the term SIM-flop will be used.

| | | Q(t+1) | | | | | | | | | | | | | | | |
|------|---|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | RS | | | | RS | | | | RS | | | | RS | | | |
| | | 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
| Q(t) | 0 | 0 | 1 | 2 | 3 | 0 | - | 2 | - | 0 | 1 | - | - | 0 | - | - | - |
| | 1 | 1 | 1 | 3 | 3 | 0 | - | 2 | - | 1 | 1 | - | - | 0 | - | - | - |
| | 2 | 2 | 3 | 2 | 3 | 2 | - | 2 | - | 0 | 1 | - | - | 0 | - | - | - |
| | 3 | 3 | 3 | 3 | 3 | 2 | - | 2 | - | 1 | 1 | - | - | 0 | - | - | - |

Figure 5.3. State Integrated RS SIM-Flop Next-State Table

Analyzation of the SIM-flop next-state table reveals several aspects of the operating behavior required of the circuit implementation. The significant characteristic is that the RS SIM-flop must be settable

and resetable independently for both the left and right machines. For example, if $Q = 0$, $R = S = 0$, and the set input changes from 0 to 2, indicating that the left machine is to be set, the SIM-flop must change to state $Q = 2$. If the S input then returns to 0, the SIM-flop should remain in state $Q = 2$. This action must occur even though the set input passes through the logic value of 1 with at least one gate reacting to this value. Another example of the independence of circuit action occurs when the SIM-flop is in state $Q = 1$ (right machine set) with $R = S = 0$. Assuming the right machine is to be reset and the left machine is to be set, R must change from 0 to 1, and S must change from 0 to 2. Even if the R and S changes occur simultaneously, the final SIM-flop state should be $Q = 2$. Note that the at-rest or no-change state of the inputs occur for $R = S = 0$.

Circuit implementation for the RS SIM-flop requires a basic asynchronous circuit from which other types of SIM-flops can be constructed. Following the binary flip-flop design, a simple cross-coupled quaternary NOR gate circuit is a reasonable specification. Such a circuit is given in Figure 5.4. The next-state table for the circuit was given in Figure 5.3. The next-state equation for the circuit is similar to the binary cross-coupled NOR flip-flop, and is given as

$$Q = R' \dot{x} (q \dot{+} S) ,$$

where Q is the next-state and q is the present-state. Although the circuit is quite simple, its action under the assumptions given in Chapter III are not as expected.

Simulation results for the circuit in Figure 5.4 reveal that the multiple-valued intolerance of the NOR gate produces circuit oscillation for certain multiple-valued input transitions. This circuit action is clearly evident in the QLOSIM simulator tabular report shown in Figure 5.5. In this simulator execution, the circuit is initially in

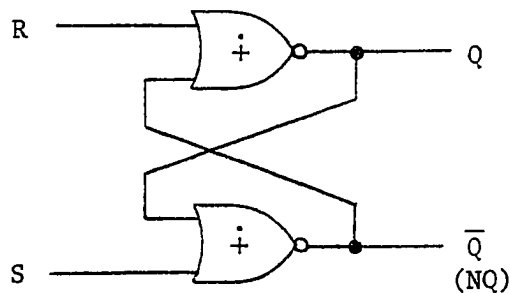


Figure 5.4. Cross-Coupled NOR Gate RS SIM-Flop

state $Q = 0$ with $R = S = 0$. Both NOR gate delays are 10 time units, and the rise/fall time is 1 time unit. When the S input changes from 0 to 2, beginning at time 1, the circuit is thrown into oscillation. The repetitive oscillation patterns are indicated in the figure. In violation of the fundamental mode operation, the S input returns to 0 at time 150, but the circuit continues to oscillate. Of course, this type of operation cannot be tolerated, even though an actual realization of the circuit might actually damp the oscillation and force the SIM-flop into a stable state. However, if damping occurs, the final state is not deterministic and could be different for each physical copy of the circuit.

QUATERNARY LOGIC SIMULATOR - QLOSIM

SRSFF.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE | |
|------|------|-------------|-----------|--------------------------|
| 1 | S | 0 | 1 | |
| 2 | S | 1 | 2 | |
| 11 | NQ | 3 | 2 | |
| 12 | NQ | 2 | 1 | |
| 21 | Q | 0 | 1 | |
| 22 | Q | 1 | 2 | |
| 31 | NQ | 1 | 0 | |
| 32 | NQ | 0 | 1 | |
| 41 | Q | 2 | 3 | } Oscillation Cycle 1 |
| 42 | Q | 3 | 2 | |
| 51 | NQ | 1 | 0 | |
| 52 | NQ | 0 | 1 | } Oscillation Cycle 2 |
| 61 | Q | 2 | 3 | |
| 62 | Q | 3 | 2 | |
| 71 | NQ | 1 | 0 | } . . |
| 72 | NQ | 0 | 1 | |
| 81 | Q | 2 | 3 | |
| 82 | Q | 3 | 2 | } . . |
| 91 | NQ | 1 | 0 | |
| 92 | NQ | 0 | 1 | |
| 101 | Q | 2 | 3 | } . . |
| 102 | Q | 3 | 2 | |
| 111 | NQ | 1 | 0 | |
| 112 | NQ | 0 | 1 | } Oscillation Cycle 5 |
| 121 | Q | 2 | 3 | |
| 122 | Q | 3 | 2 | |
| 131 | NQ | 1 | 0 | |
| 132 | NQ | 0 | 1 | |
| 141 | Q | 2 | 3 | |
| 142 | Q | 3 | 2 | |
| 150 | S | 2 | 1 | |
| 151 | S | 1 | 0 | |
| 151 | NQ | 1 | 0 | |
| 152 | NQ | 0 | 1 | |
| 160 | NQ | 1 | 0 | |
| 161 | NQ | 0 | 1 | |
| 161 | Q | 2 | 3 | |
| 162 | Q | 3 | 2 | |
| 170 | Q | 2 | 3 | |
| 171 | Q | 3 | 2 | |
| 171 | NQ | 1 | 0 | |
| 172 | NQ | 0 | 1 | |
| 180 | NQ | 1 | 0 | |

SIMULATION TERMINATED AT TIME 180

Figure 5.5. Simulation of NOR Gate RS SIM-Flop

Attempts at correcting the circuit behavior by altering and imbalancing the element transport delays have been unsuccessful, due primarily to the fact that each gate continues to react to each logic level. Imbalancing the feedback path delays has also proven unsuccessful. For these reasons, this basic cross-coupled NOR gate arrangement will be abandoned.

As mentioned previously, the next-state table for the four-valued RS SIM-flop is similar to that given for the four-valued r-flop, differing in varying degrees in the columns under the RS = 01, RS = 02, RS = 10, RS = 20, and RS = 11 inputs. See Figures 3.15 and 5.3. The RS = 11 input is actually invalid or undesirable for the four-valued RS SIM-flop but not for the RS r-flop. The undesirable input condition for the SIM-flop does not imply that the circuit has no output, but simply means that the outputs are undefined or meaningless in an operational sense.

These observations suggest that the four-valued RS r-flop may be adapted or modified to give the proper circuit behavior. Indeed, this is the case. The circuit given in Figure 5.6 utilizes the cross-coupled four-valued $\overline{\text{MAX}}$ gates which are modified by the addition of four-valued OR gates, that are coupled into the feedback loops. Although this RS SIM-flop circuit appears different, it is possible to show that the circuit is algebraically equivalent to the previous cross-coupled NOR SIM-flop, that is, the circuit's next-state equation can be reduced to

$$Q = R' \dot{x} (q \dot{+} S) ,$$

where Q is the next-state and q is the present-state.

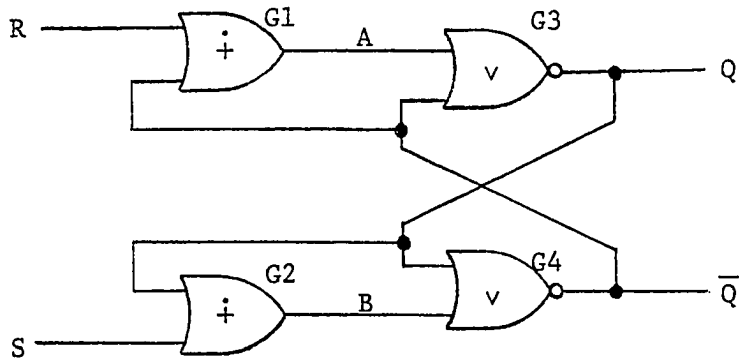


Figure 5.6. Modified RS SIM-Flop Circuit

To show this result, first note that the following relationships hold,

$$\begin{aligned} A &= q' \dot{+} R \\ B &= q \dot{+} S \\ q' &= (q \vee B)' \\ Q &= (q' \vee A)' . \end{aligned}$$

Substituting for A in the last equation,

$$Q = [q' \vee (q' \dot{+} R)]' .$$

From the results of Chapter III, it is known that

$$q' \vee (q' \dot{+} R) = q' \dot{+} R .$$

Applying this result,

$$Q = (q' \dot{+} R)'$$

and substitution for q' gives

$$Q = [(q \vee B)' \dot{+} R]' .$$

Substituting for B,

$$Q = \{[q \vee (q \dot{+} S)]' \dot{+} R\}' .$$

Again, applying the results from Chapter III,

$$Q = [(q \dot{+} S)' \dot{+} R]' .$$

Using DeMorgan's Theorem,

$$Q = (q \dot{+} S) \dot{\times} R'$$

and

$$Q = R' \dot{\times} (q \dot{+} S),$$

which is the same next-state equation as before. Therefore, the two SIM-flop types are algebraically equivalent. It is interesting to note that the same algebraic result holds for the equivalent binary flip-flop in which the $\overline{\text{MAX}}$ becomes a NOR gate. In the binary system the $\overline{\text{MAX}}$ and NOR gates are identical, and the resulting flip-flop circuit is highly redundant but produces the same functional result.

Although equivalent algebraically to the circuit in Figure 5.4, the modified SIM-flop produces superior behavioral characteristics. Simulation results for this SIM-flop show that the previously observed oscillatory conditions are suppressed. Figure 5.7 gives the same simulation conditions as before in which S changes from 0 to 2. Clearly, the modified SIM-flop reaches a stable state $Q = 2$ as required by the next-state table. At time 150, S returns to 0. Except for a final 1-to-0 0-to-1 transition on the \overline{Q} output, the stability of the SIM-flop is evident. The implication of the \overline{Q} behavior will be discussed later.

To give a better idea of the SIM-flop behavior, a more detailed simulation of the circuit is given in Figure 5.8. In this simulation both the R and S inputs are stimulated, sometimes simultaneously for the left and right logical machines. The SIM-flop is initialized in state $Q = 0$ with $R = S = 0$. At Time 1, S changes from 0 to 2 which sets the left machine, $Q = 2$ at Time 42. S then returns to 0. At Time 200, both S and R change values with S going from 0 to 1 and R going from 0 to 2. This input combination represents the right machine being set and the

QUATERNARY LOGIC SIMULATOR - QLOSIM

SRSFF.EX2 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S | 0 | 1 |
| 2 | S | 1 | 2 |
| 21 | NQ | 3 | 2 |
| 22 | NQ | 2 | 1 |
| 41 | Q | 0 | 1 |
| 42 | Q | 1 | 2 |
| 61 | NQ | 1 | 0 |
| 62 | NQ | 0 | 1 |
| 150 | S | 1 | 0 |
| 151 | S | 1 | 0 |
| 170 | NQ | 1 | 0 |
| 171 | NQ | 0 | 1 |

SIMULATION TERMINATED AT TIME 181

Figure 5.7. Simulation of Four-Valued RS SIM-Flop

left machine being reset. The final state should be $Q = 1$, and this, indeed, is the result at Time 262. After R and S return to 0, S changes to a value of 3, leaving the circuit in state $Q = 3$ at Time 442. Finally, R resets both machines, and the simulation terminates at Time 702 with $Q = 0$.

Although favorable operation of the RS SIM-flop is achieved for the simulation above, not all valid input transitions cause acceptable operation. Two particular R and S input transitions cause oscillation to occur when the transport delays of the OR gates are equal. The two cases for which the oscillatory conditions occur are (1) when $Q = 2$ and R changes from 0 to 2 and S changes simultaneously from 2 to 0, and (2) when $Q = 1$ and R changes from 2 to 0 and S changes simultaneously from 0 to 2. Both conditions are correctable by making the OR gate transport delay values unequal, e.g., the G1 Δ_T value can be 11 and the G2 Δ_T value can be 10. When the R and S input changes are separated in time by more than the OR gates' Δ_T value, assuming the Δ_T 's are equal, oscillation will not occur provided the ordering of the input changes is correct. That is, for the first case cited above, R must first change from 0 to 2 followed by the change in S at least $\Delta_T + 1$ time units later, where Δ_T is the transport delay of the OR gates. For the second case, the same relationship is true except that the timing for R and S is interchanged.

The application of the RS SIM-flop in sequential circuits must be examined closely to insure that the oscillation conditions discussed above do not jeopardize normal circuit behavior. The QLOSIM simulator is, of course, available to investigate these possibilities, and, if

QUATERNARY LOGIC SIMULATOR - QLOSIM

SRSFF.EX3 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S | 0 | 1 |
| 2 | S | 1 | 2 |
| 21 | NQ | 3 | 2 |
| 22 | NQ | 2 | 1 |
| 41 | Q | 0 | 1 |
| 42 | Q | 1 | 2 |
| 61 | NQ | 1 | 0 |
| 62 | NQ | 0 | 1 |
| 100 | S | 2 | 1 |
| 101 | S | 1 | 0 |
| 120 | NQ | 1 | 0 |
| 121 | NQ | 0 | 1 |
| 200 | S | 0 | 1 |
| 200 | R | 0 | 1 |
| 201 | R | 1 | 2 |
| 220 | NQ | 1 | 0 |
| 221 | Q | 2 | 1 |
| 222 | Q | 1 | 0 |
| 240 | Q | 0 | 1 |
| 241 | NQ | 0 | 1 |
| 242 | NQ | 1 | 2 |
| 261 | Q | 1 | 0 |
| 262 | Q | 0 | 1 |
| 300 | S | 1 | 0 |
| 300 | R | 2 | 1 |
| 301 | R | 1 | 0 |
| 320 | Q | 1 | 0 |
| 321 | Q | 0 | 1 |
| 400 | S | 0 | 1 |
| 401 | S | 1 | 2 |
| 402 | S | 2 | 3 |
| 421 | NQ | 2 | 1 |
| 422 | NQ | 1 | 0 |
| 441 | Q | 1 | 2 |
| 442 | Q | 2 | 3 |
| 500 | S | 3 | 2 |
| 501 | S | 2 | 1 |
| 502 | S | 1 | 0 |

Figure 5.8. Detailed Simulation of Four-Valued RS SIM-Flop - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

SRSFF.EX3 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 600 | R | 0 | 1 |
| 601 | R | 1 | 2 |
| 602 | R | 2 | 3 |
| 620 | Q | 3 | 2 |
| 621 | Q | 2 | 1 |
| 622 | Q | 1 | 0 |
| 640 | NQ | 0 | 1 |
| 641 | NQ | 1 | 2 |
| 642 | NQ | 2 | 3 |
| 700 | R | 3 | 2 |
| 701 | R | 2 | 1 |
| 702 | R | 1 | 0 |

SIMULATION TERMINATED AT TIME 702

Figure 5.8. Detailed Simulation of Four-Valued RS SIM-Flop - Part 2

necessary, a specific circuit employing the SIM-flop can be "tuned" to provide proper response.

The remaining combinations of RS SIM-flop inputs can be shown to yield proper SIM-flop operations using balanced OR gate transport delays. There are, however, other characteristics of the RS SIM-flop that require investigation. The mechanics of the stability are necessary, and these help to understand why the glitches occur on the Q and/or \bar{Q} outputs. For example, in Figure 5.7 a glitch appeared on the \bar{Q} output as the S input changed from 2 to 0. Knowing that the gates react to all intermediate transitions, the G2 OR gate in Figure 5.6 will react to the intermediate value of 1 on the S input. Since the Q input is 2, the G2 output is temporarily a 3 and causes a change in the output of the G4 $\overline{\text{MAX}}$ gate. The G4 reaction occurs for only one time unit and is a 1-to-0 0-to-1 transition. The SIM-flop remains stable, however, because the G1 OR gate transport delay buffers the \bar{Q} glitch and prevents it from

affecting the Q output. Before the G1 gate reacts to the \bar{Q} input, \bar{Q} has returned to 1 and prevents the G1 output from affecting G3. Thus, the Q side of the RS SIM-flop remains stable. The criteria for stability is that the OR gate transport delay must be greater than the signal rise/fall time. This is a conservative requirement and should be true for most electronic devices.

The tolerance of the RS SIM-flop to short duration input glitches or transients is another interesting characteristic. This type of transient input is in violation of the fundamental mode of operation, but asynchronous sequential circuits may exist in which this action will occur. Although actual physical device characteristics may differ from

QUATERNARY LOGIC SIMULATOR - QLOSIM

SFF.EX4 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S | 0 | 1 |
| 2 | S | 1 | 2 |
| 10 | S | 2 | 1 |
| 11 | S | 1 | 0 |
| 11 | B | 0 | 1 |
| 12 | B | 1 | 2 |
| 20 | B | 2 | 1 |
| 21 | B | 1 | 0 |
| 21 | NQ | 3 | 2 |
| 22 | NQ | 2 | 1 |
| 30 | NQ | 1 | 2 |
| 31 | NQ | 2 | 3 |
| 31 | A | 3 | 2 |
| 32 | A | 2 | 1 |
| 40 | A | 1 | 2 |
| 41 | A | 2 | 3 |

SIMULATION TERMINATED AT TIME 41

Figure 5.9. RS SIM-Flop Transient Behavior Simulation

the assumptions made here, the transient input tolerance of the RS SIM-flop is nevertheless a useful topic and helps to predict its behavior in larger sequential circuits.

Figure 5.9 shows the result of a short simulation in which the S input to the RS SIM-flop is changed from 0 to 2 and back to 0 before the SIM-flop has fully responded to the initial 0 to 2 transition. The SIM-flop is initially in state $Q = 0$ and the rise/fall time delay is one time unit. All gate transport delays are 10 time units. Although the circuit responds to the transient input on node S, the state of the SIM-flop remains unchanged. The ratio of rise/fall time to the transport delay of the G1 OR gate is again a critical aspect of the SIM-flop's stability, i.e., if the S input transient, including rise/fall time, does not exceed the G1 OR gate transport delay, the RS SIM-flop will remain stable.

If the S input transient duration exceeds the G1 OR gate transport delay but is less than the latch time of the circuit, the RS SIM-flop is unstable and will oscillate. Simulation results for this condition can be summarized as follows. Let T_S be the transient duration on input node S. Further, let $\Delta_T(G_i)$ be the transport delay of the i th gate in Figure 5.6 and let Δ_{RFS} be the total rise/fall time of the S input. An unstable condition exists for the RS SIM-flop if

$$\Delta_T(G1) < T_S < \{[\Delta_T(G_2) + \Delta_T(G_4) + \Delta_T(G1)] + \Delta_{RFS} + 1\} .$$

If T_S exceeds the right hand side of the relation above, a stable condition exists and the RS SIM-flop will change state normally.

With this characterization of the RS SIM-flop established, other SIM-flop types can be specified. Using the RS SIM-flop as a

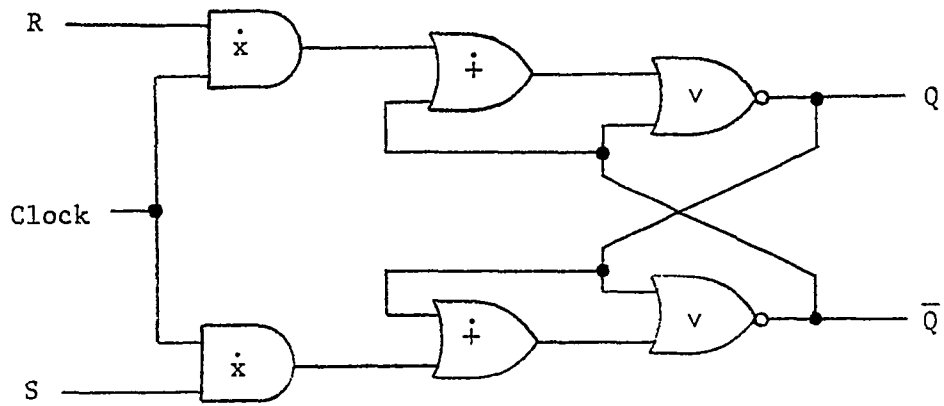


Figure 5.10. Four-Valued Clocked RS SIM-Flop

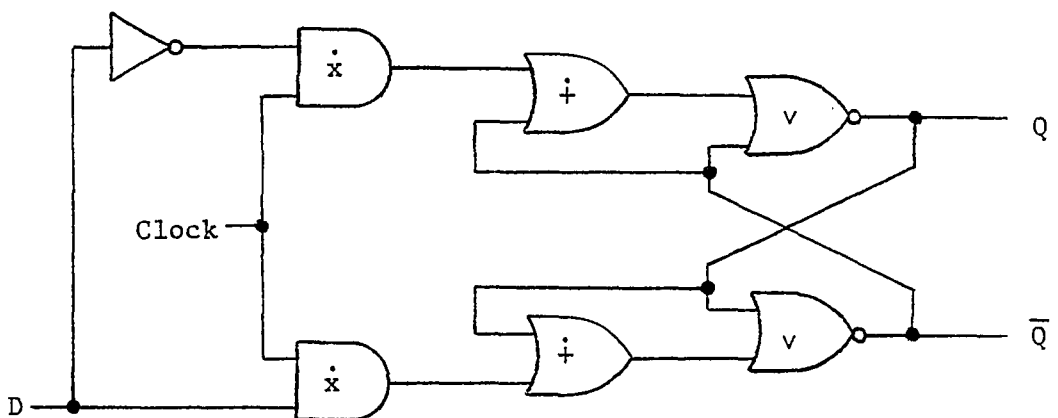


Figure 5.11. Four-Valued D-Type SIM-Flop

basis, the clocked RS and D-type SIM-flops are simple extensions. Circuits for these two SIM-flops are shown in Figure 5.10 and Figure 5.11. Both of these four-valued SIM-flops possess clock inputs that are true multiple-valued clocks. That is, the clock input level selects the machine, left, right or both, for which the input stimulus is to be applied. As an example, suppose the Q output of the D-type SIM-flop is 0, and the clock input is also 0. If the D input is 3 and the clock changes to 1, the Q output will change to $Q = 1$. Thus, only the right machine changes state. This type of behavior is evident in the next-state table of the D-type SIM-flop which is given in Figure 5.12. The clocked RS SIM-flop has a similar mode of operation. Simulation of

| | | Q(t+1) | | | | | | | | | | | | | | | |
|------|---|---------|---|---|---|---------|---|---|---|---------|---|---|---|---------|---|---|---|
| | | CLK = 0 | | | | CLK = 1 | | | | CLK = 2 | | | | CLK = 3 | | | |
| | | D | | | | D | | | | D | | | | D | | | |
| Q(t) | | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 2 | 0 | 1 | 2 | 3 | |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 3 | 3 | 0 | 1 | 2 | 3 | |
| 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 3 | 0 | 0 | 2 | 2 | 0 | 1 | 2 | 3 | |
| 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 | 3 | 1 | 1 | 3 | 3 | 0 | 1 | 2 | 3 | |

Figure 5.12. Four-Valued D-Type SIM-Flop Next-State Table

the clocked RS and D-type SIM-flops will be seen in sequential circuit examples that follow in the next chapter.

The basic design of the RS SIM-flop given in Figure 5.6 can be extended to the three-machines or $B_2 \times B_2 \times B_2$ case. This extension yields an eight-valued logic device that simultaneously stores the state for three binary machines. The algebraic results for the eight-valued case

are the same as were given previously; that is, the next-state equation is

$$Q = R' \dot{x} (q \dot{+} S) .$$

It is difficult to manually evaluate the detailed behavior of the eight-valued RS SIM-flop for all possible input conditions and SIM-flop states. The next-state table contains 512 entries that result from eight SIM-flop states and 64 combinations of the eight-valued R and S inputs. With eight logic values, the analysis problem is exponentially more difficult than the four-valued case. Nevertheless, it is felt that the cross-coupled four-gate SIM-flop structure is reasonable and, hopefully, workable for the three-machine case.

In summary, the work in this chapter has established a logical structure for a state integrated multiple-valued memory element. Detailed functional and behavioral results have been shown for the four-valued RS SIM-flop. In addition, other SIM-flop types were derived from the basic RS device. The capability of these devices to work well in state integrated and multiplexed sequential circuits will be shown in the next chapter. The ability to apply these devices in state integrated sequential circuits with the same ease as one would apply a flip-flop in a binary sequential circuit is a very important result of this work.

CHAPTER VI

STATE INTEGRATED AND MULTIPLEXED SEQUENTIAL CIRCUITS

Circuit Model and Behavior

A sequential circuit is defined [36] as a network in which the output at any instant is dependent not only upon the inputs present at that instant, but also upon the past history (or sequence) of inputs. The past history of inputs must be preserved by the network. For this reason, sequential circuits have memory. The concept which allows information to be preserved is referred to as the internal state, or simply state, of the network. In a sequential circuit realization, the internal state becomes the collection of signals at specified points within the network.

The general sequential circuit model is shown in Figure 6.1. The circuit contains n inputs x_1, x_2, \dots, x_n and m output terminals designated z_1, z_2, \dots, z_m . The combinational circuit portion is the same as described in Chapter IV. The memory portion was discussed in Chapter V and consists of circuit elements or devices arranged to store logic levels. The memory devices may be delay mechanisms or actual memory elements referred to as r-flops (flip-flops in the binary case). The secondary inputs y_1, y_2, \dots, y_p are the present-state terminals, and the

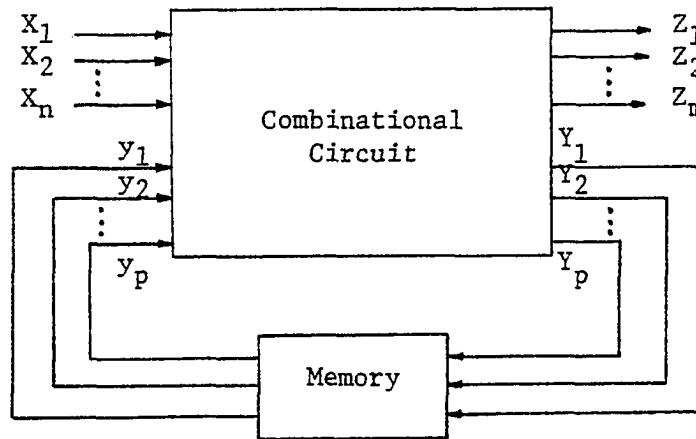


Figure 6.1. General Sequential Circuit Model

secondary outputs are the Y_1, Y_2, \dots, Y_p connections. Note that the model provides feedback paths from the memory elements that are logically combined with the inputs. In general, the output variables are related to the inputs as follows:

$$Z_i = f_i(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_p) \quad i = 1, 2, \dots, m$$

and

$$Y_j = g_j(x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_p) \quad j = 1, 2, \dots, p.$$

When these relationships hold explicitly, the circuit is referred to as a Mealy machine. When the primary outputs are not a function of the primary inputs, i.e., they are a function of only the secondary inputs, y_k , the circuit is a Moore machine.

Sequential circuits are also distinguished by the method of operation, being either synchronous or asynchronous [23]. Synchronous sequential circuits operate in either the clocked or pulse mode. In the clock mode, a clock signal is used on the r-flops to control or synchronize the change of state. In the pulse mode, circuit input pulses control the application of circuit input signals. All internal

states are stable in the synchronous circuit. Only one mode of operation, the fundamental mode, exists for the asynchronous circuit. The basic requirements for fundamental mode operation are that only a single input may change value at a time, and once this change occurs, all other inputs must remain unchanged until the circuit reaches a final stable state. These restrictions are important since the asynchronous circuit can possess intermediate unstable transition states. When an input changes, the state transition may pass through these unstable transition states, and the restriction on input transitions insures that the proper final stable state will be reached.

The description of a sequential circuit can be given as a flow table. The flow table depicts the operation of a machine independent of the manner in which it is ultimately constructed. A general flow table was given in Chapter III. Except for the labels on the inputs, the multiple-valued flow table will be exactly like one for a binary circuit. Given a flow table for an r -valued machine, it can be converted into a reduced flow table by methods given by Givone [36].

Sheafor [15] restricts the construction of the multiple-valued flow table to exclude ambiguous circuit operation. For example, in the asynchronous flow table of Figure 6.2, assume that the machine is in state A with input $X = 0$. For X changing to a value of 2, the operation of the circuit is questionable since it is not clear whether the circuit's final state will be B or C. That is, if the circuit reacts completely to the intermediate value of $X = 1$, the final state of the circuit is B. If the circuit does not react to $X = 1$, the final state is C. As a consequence of this ambiguous action, Sheafor formulates conditions for

flow table construction. Basically, two types of restrictions are asserted. If the flow table is constructed as in Figure 6.2, then the

| | X | | | |
|---|--------|--------|--------|--------|
| | 0 | 1 | 2 | 3 |
| A | (A), 0 | B, 1 | C, 1 | (A), 1 |
| B | A, 0 | (B), 1 | (B), 0 | A, 1 |
| C | A, 0 | (C), 0 | (C), 1 | D, 0 |
| D | A, 0 | B, 1 | C, 1 | (D), 0 |

Figure 6.2. Four-Valued Flow Table

type of input change described above is not allowed. If this type of input change is possible, then the flow table entries in the row which is under a column between the initial stable entry and the final entry must be either stable or a transition to the same row as the final entry.

Flow table construction for the B_{2^m} state integrated and multiplexed digital system will not be restricted as described above. The flow table presented in Chapter III and the one in Figure 6.2 are perfectly valid for the work presented here. The SIM-flops developed in Chapter V tolerate multiple-valued signal transitions and contribute significantly to the proper operation of state integrated and multiplexed sequential circuits.

Circuit Design Methodology

The sequential circuit design philosophy for multiple-valued state integrated and multiplexed digital systems is basically the same as for combinational circuits. That is, the sequential circuit design is

first performed for the binary case and is then projected to the B_{2^m} system by a simple substitution of multiple-valued logic gates. In this section the details of this synthesis procedure for four-valued sequential circuits will be performed by way of example. To keep the circuit simple but still provide feedback connections, a non-reduced flow table will be used. Using the reduced flow table results in a circuit that does not contain feedback and, hence, is not a good circuit for demonstrating the design philosophy.

The non-reduced binary flow table is given in Figure 6.3 for a fundamental mode asynchronous machine. The circuit has one input, labeled X, three internal states, and one output, labeled Z. The circled entries indicate stable states. Since the flow table contains three states, two binary flip-flops are required for the circuit realization. The adjacency diagram is given in Figure 6.4 and clearly shows that states B and C can be reached from state A; however, A cannot be

| | Next State (A,B,C) | | Output (Z) | |
|---|-----------------------|-----|---------------|---|
| | Input (X) | | Input (X) | |
| | 0 | 1 | 0 | 1 |
| A | B | (A) | - | 0 |
| B | (B) | C | 1 | - |
| C | B | (C) | - | 1 |

Figure 6.3. Fundamental Mode Binary Flow Table

reached from B or C. To prevent race conditions, the diagram further implies that the state assignment for B should be adjacent to both A and C, but A and C need not be adjacent states. Therefore, the states are coded with the following y_1y_2 assignments:

A = 00

B = 01

C = 11.

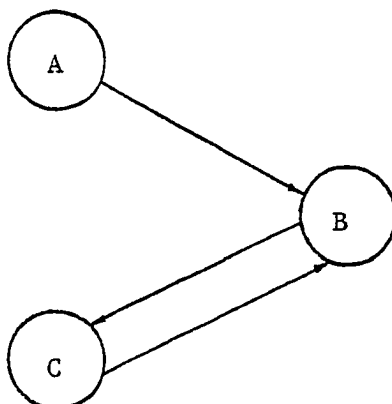


Figure 6.4. State Adjacency Diagram

This state assignment is race free. Using RS flip-flops for the binary memory elements, the excitation table is shown in Figure 6.5. Based on the state assignment given above, this table provides the necessary R and S inputs for the two RS flip-flops. Using the Karnaugh maps in Figure 6.6, the expressions for the R and S inputs and output Z can be constructed. The circuit realization for the binary asynchronous machine is shown in Figure 6.7. To form a four-valued asynchronous state integrated and multiplexed machine, the binary combinational gates are replaced by four-valued Boolean gates, and the binary RS flip-flops are replaced with RS SIM-flops. This projection of the binary circuit into the $B_2 \times B_2$ system forms the four-valued total composite machine (TCM) $M_2^L \times M_2^R$, where M_2^L is equivalent to M_2^R . However, one additional modification is required for the TCM. Since the delay in the feedback paths may

be less than the latch time of the RS SIM-flops, additional gates or buffers are added at the output of the RS SIM-flops. The resulting four-valued TCM circuit is shown in Figure 6.8, where the nodes have been labeled for the simulator evaluation that follows.

| | | MEMORY ($R_1S_1; R_2S_2$) | | OUTPUT (Z) | |
|----------|----|--------------------------------|-------|---------------|---|
| | | Input (X) | | Input (X) | |
| | | 0 | 1 | 0 | 1 |
| y_1y_2 | 00 | --;01 | -0;-0 | 1 | 0 |
| | 01 | -0;-0 | 01;0- | 1 | 1 |
| | 11 | 10;0- | 0-;0- | 1 | 1 |

Figure 6.5. Excitation Table

Before simulating the TCM circuit, an input transition sequence is required to structure the verification process. Although the sequences that verify correct circuit operation for both M_2^L and M_2^R , can be generated from the binary state table, the use of the TCM state table will better illustrate the method. Figure 6.9 shows the TCM state table in which the next-state output subtables are given together and separated by commas.

An input transition sequence for the TCM will verify that each state transition will be made properly. The adjacency diagram for the binary circuit, however, shows that state A is not reachable from states B or C. This means that the transition sequence will consist of several disjoint subsequences.

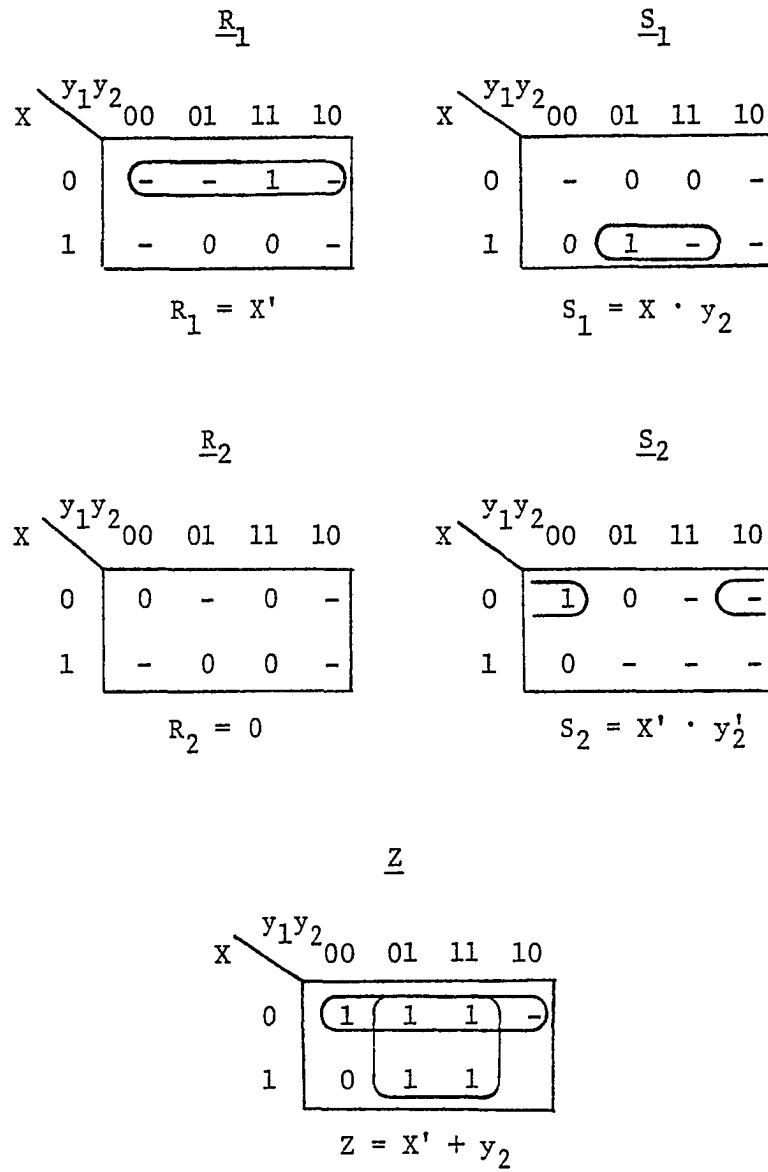


Figure 6.6. Excitation and Output Karnaugh Maps

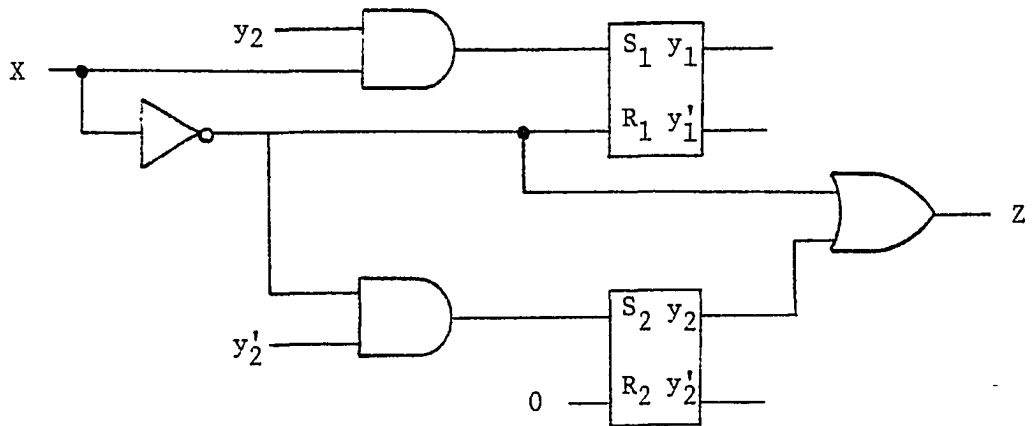


Figure 6.7. Binary Circuit Realization

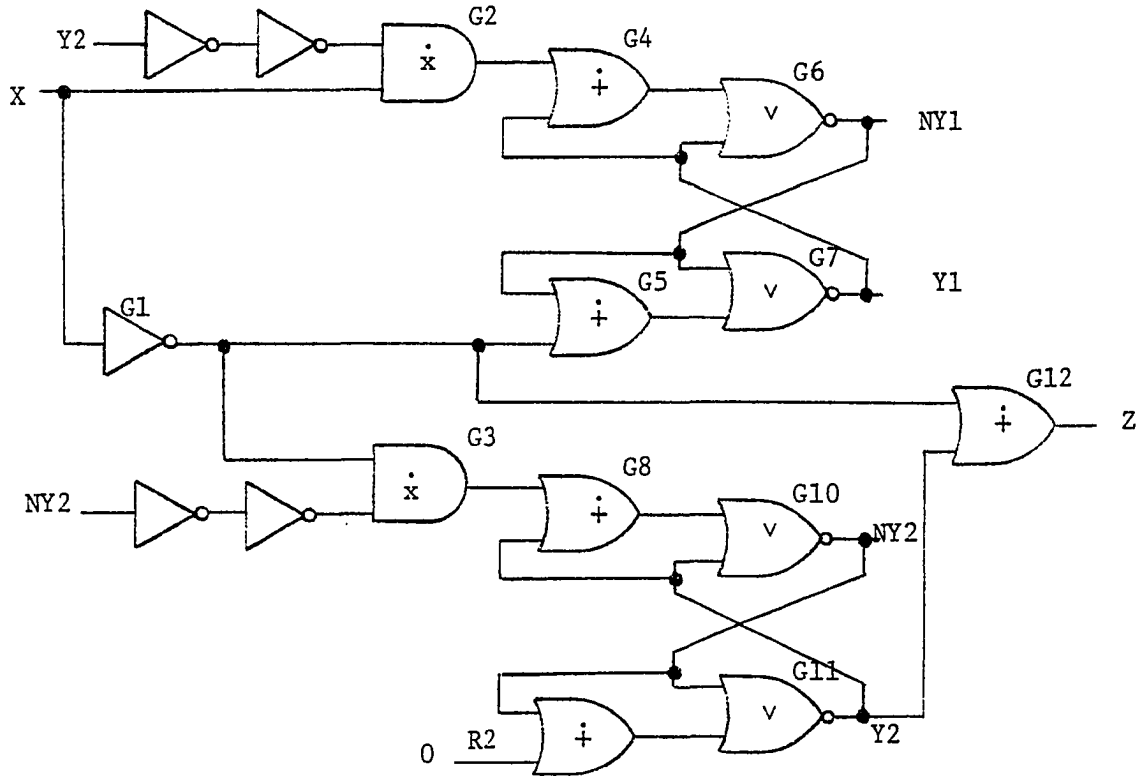


Figure 6.8. Four-Valued TCM Realization

| | | Next State (Y_1Y_2) | | | |
|------------------------------------|----|-------------------------|---------|---------|---------|
| | | Input (X) | | | |
| | | 0 | 1 | 2 | 3 |
| Present State (y_1, y_2) | 00 | 03 , 3 | 02 , 2 | 01 , 1 | ⊙00 , 0 |
| | 01 | 03 , 3 | 13 , 3 | ⊙01 , 1 | 11 , 1 |
| | 11 | 03 , 3 | 13 , 3 | 00 , 1 | ⊙11 , 1 |
| | 02 | 03 , 3 | ⊙02 , 2 | 23 , 3 | 22 , 2 |
| | 03 | ⊙03 , 3 | 13 , 3 | 23 , 3 | 33 , 3 |
| | 13 | 03 , 3 | ⊙13 , 3 | 23 , 3 | 33 , 3 |
| | 22 | 03 , 3 | 02 , 2 | 23 , 3 | ⊙22 , 2 |
| | 23 | 03 , 3 | 13 , 3 | ⊙23 , 3 | 33 , 3 |
| | 33 | 03 , 3 | 13 , 3 | 23 , 3 | ⊙33 , 3 |

Figure 6.9. TCM State Table

The transition subsequences given in Figure 6.10 were constructed from the TCM state table. The state y_1y_2 values are shown in the circles and the value of the X input is given on the arrows. Each subsequence requires a separate execution of the QLOSIM simulator. Many of the transitions are redundant in that each subsequence need not begin at state $y_1y_2 = 00$; however, this initial state selection allows more efficient tracing in the state table and is a convenient starting point for execution.

The sequences have been grouped to reflect the type of operation within the M_2^L and M_2^R machines. In the first subsequence given in Figure 6.10(a), both M_2^L and M_2^R are driven to state B ($y_1y_2 = 03$), and then all TCM B and C state transition combinations follow. In the subsequences of Figure 6.10(b) M_2^R is driven to state B while M_2^L is held in state A, whereas, in Figure 6.10(c) the reverse is done. Once both

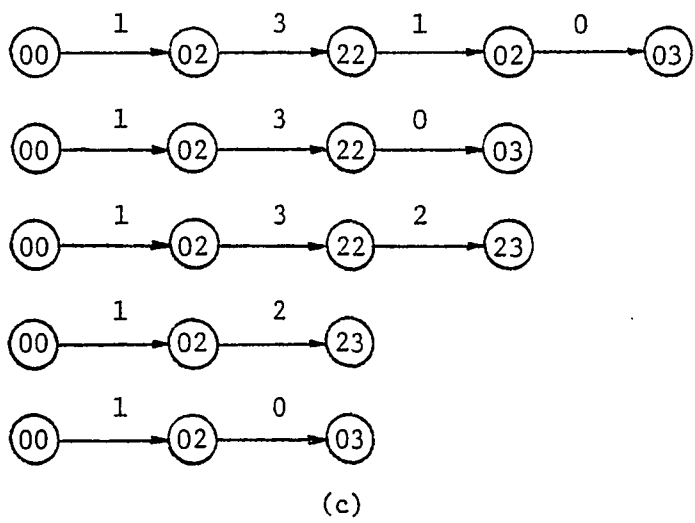
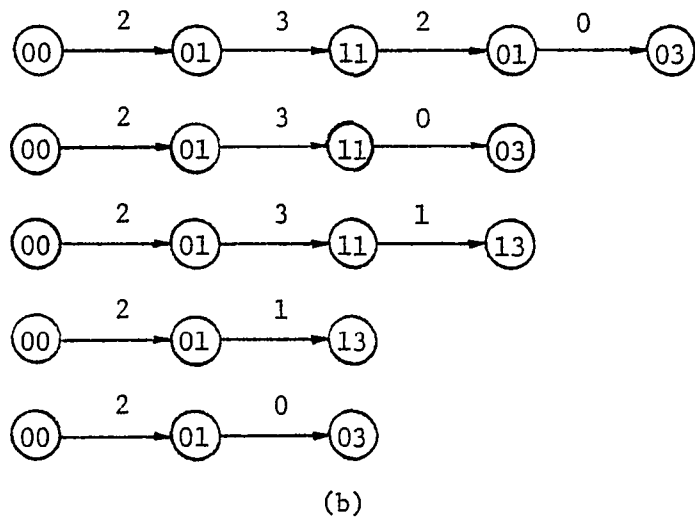
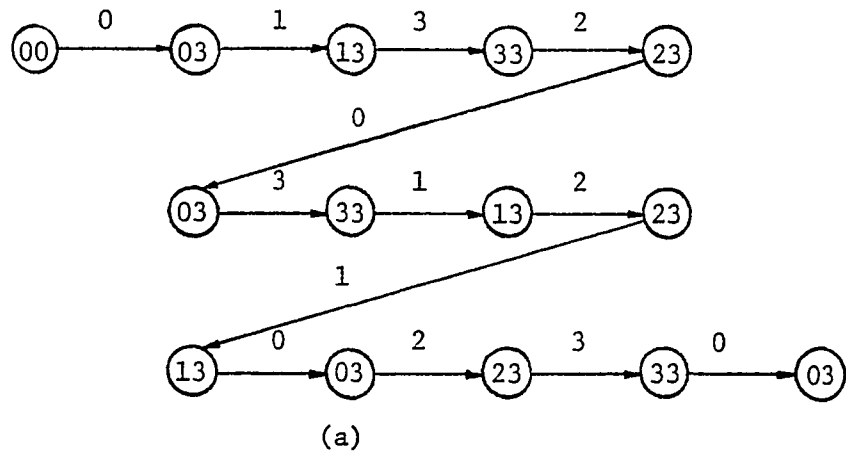


Figure 6.10. TCM Input Transition Sequences

M_2^L and M_2^R enter state B or C concurrently, the subsequences in Figure 6.10(b) and (c) join the subsequence given in Figure 6.10(a).

Simulation of all the subsequences in Figure 6.10 can be performed and shown to work properly; however, only the subsequence in Figure 6.10(a) and the first subsequence in Figure 6.10(b) and (c) are repeated here. These simulations are given, respectively, in Figures 6.11, 6.12 and 6.13. All three simulations were run with the Δ_T delay for the G4 and G8 OR gates (see Figure 6.8) set to 11 time units. All other gate delays have a Δ_T delay of 10 time units. For this particular TCM circuit, the OR gate delay is necessary on gate G4 since the 3 to 1 transition of X at Time 1000 in Figure 6.13 will cause oscillation in the Y1 RS SIM-flop. In this case S1 is changing from 2 to 0, simultaneously, as R changes from 0 to 2. Figure 6.14 shows the oscillatory condition simulated starting with the X transition from 3 to 1 and $y_1y_2 = 22$. All gate transport delays are 10 time units for this simulation. The oscillation is clearly evident on the Y1 output and will continue on in time except that the simulator was forced to stop at Time 1101.

The synchronous or clocked RS SIM-flop can be used in place of the asynchronous RS SIM-flop in the TCM circuit. The clocked RS SIM-flop does not exhibit the oscillatory condition but does require external control of the clock input. Figure 6.15 shows the TCM circuit implemented with clocked RS SIM-flops. The simulation of this circuit using the first subsequence of Figure 6.10(c) is given in Figure 6.16. All gate Δ_T values are 10 time units. Note that no oscillation occurs for this circuit realization. The clock input is asserted after the input has

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 3 | X | 1 | 0 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 23 | Z | 2 | 3 |
| 41 | NY2 | 3 | 2 |
| 42 | NY2 | 2 | 1 |
| 43 | NY2 | 1 | 0 |
| 61 | Y2 | 0 | 1 |
| 62 | Y2 | 1 | 2 |
| 63 | Y2 | 2 | 3 |
| 500 | X | 0 | 1 |
| 551 | Y1 | 0 | 1 |
| 1000 | X | 1 | 2 |
| 1001 | X | 2 | 3 |
| 1030 | Y1 | 1 | 0 |
| 1031 | Y1 | 0 | 1 |
| 1051 | Y1 | 1 | 2 |
| 1052 | Y1 | 2 | 3 |
| 1500 | X | 3 | 2 |
| 1530 | Y1 | 3 | 2 |
| 2000 | X | 2 | 1 |
| 2001 | X | 1 | 0 |
| 2030 | Y1 | 2 | 1 |
| 2031 | Y1 | 1 | 0 |
| 2500 | X | 0 | 1 |
| 2501 | X | 1 | 2 |
| 2502 | X | 2 | 3 |
| 2551 | Y1 | 0 | 1 |
| 2552 | Y1 | 1 | 2 |
| 2553 | Y1 | 2 | 3 |
| 3000 | X | 3 | 2 |
| 3001 | X | 2 | 1 |
| 3030 | Y1 | 3 | 2 |
| 3031 | Y1 | 2 | 1 |
| 3072 | Y1 | 1 | 0 |
| 3073 | Y1 | 0 | 1 |
| 3500 | X | 1 | 2 |
| 3530 | Y1 | 1 | 0 |
| 3551 | Y1 | 0 | 1 |
| 3552 | Y1 | 1 | 2 |
| 4000 | X | 2 | 1 |

Figure 6.11. TCM Subsequence Simulation for Figure 6.10(a) - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 4030 | Y1 | 2 | 1 |
| 4031 | Y1 | 1 | 0 |
| 4051 | Y1 | 0 | 1 |
| 4071 | Y1 | 1 | 0 |
| 4072 | Y1 | 0 | 1 |
| 4500 | X | 1 | 0 |
| 4530 | Y1 | 1 | 0 |
| 5000 | X | 0 | 1 |
| 5001 | X | 1 | 2 |
| 5052 | Y1 | 0 | 1 |
| 5053 | Y1 | 1 | 2 |
| 5500 | X | 2 | 3 |
| 5551 | Y1 | 2 | 3 |
| 6000 | X | 3 | 2 |
| 6001 | X | 2 | 1 |
| 6002 | X | 1 | 0 |
| 6030 | Y1 | 3 | 2 |
| 6031 | Y1 | 2 | 1 |
| 6032 | Y1 | 1 | 0 |

SIMULATION TERMINATED AT TIME 6053

Figure 6.11. TCM Subsequence Simulation for Figure 6.10(a) - Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX2 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 21 | Z | 0 | 1 |
| 41 | NY2 | 3 | 2 |
| 61 | Y2 | 0 | 1 |
| 500 | X | 2 | 3 |
| 551 | Y1 | 0 | 1 |
| 1000 | X | 3 | 2 |
| 1030 | Y1 | 1 | 0 |
| 1500 | X | 2 | 1 |
| 1501 | X | 1 | 0 |
| 1520 | Z | 1 | 2 |
| 1521 | Z | 2 | 3 |
| 1541 | NY2 | 2 | 1 |
| 1542 | NY2 | 1 | 0 |
| 1561 | Y2 | 1 | 2 |
| 1562 | Y2 | 2 | 3 |

SIMULATION TERMINATED AT TIME 1602

Figure 6.12. TCM Subsequence Simulation for Figure 6.10(b)

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX3 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 41 | NY2 | 3 | 2 |
| 42 | NY2 | 2 | 1 |
| 61 | Y2 | 0 | 1 |
| 62 | Y2 | 1 | 2 |
| 71 | Z | 2 | 3 |
| 72 | Z | 3 | 2 |
| 81 | NY2 | 1 | 0 |
| 82 | NY2 | 0 | 1 |
| 112 | NY2 | 1 | 0 |
| 113 | NY2 | 0 | 1 |
| 500 | X | 1 | 2 |
| 501 | X | 2 | 3 |
| 520 | Z | 2 | 3 |
| 521 | Z | 3 | 2 |
| 540 | NY2 | 1 | 0 |
| 541 | NY2 | 0 | 1 |
| 551 | Y1 | 0 | 1 |
| 552 | Y1 | 1 | 2 |
| 1000 | X | 3 | 2 |
| 1001 | X | 2 | 1 |
| 1020 | Z | 2 | 3 |
| 1021 | Z | 3 | 2 |
| 1031 | Y1 | 2 | 1 |
| 1032 | Y1 | 1 | 0 |
| 1040 | NY2 | 1 | 0 |
| 1041 | NY2 | 0 | 1 |
| 1052 | Y1 | 0 | 1 |
| 1053 | Y1 | 1 | 0 |
| 1072 | Y1 | 0 | 1 |
| 1073 | Y1 | 1 | 0 |
| 1500 | X | 1 | 0 |
| 1520 | Z | 2 | 3 |
| 1540 | NY2 | 1 | 0 |
| 1560 | Y2 | 2 | 3 |

SIMULATION TERMINATED AT TIME 1600

Figure 6.13. TCM Subsequence Simulation for Figure 6.10(c)

QUATERNARY LOGIC SIMULATOR -- QLOSIM

TCM.EX4 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 41 | NY2 | 3 | 2 |
| 42 | NY2 | 2 | 1 |
| 61 | Y2 | 0 | 1 |
| 62 | Y2 | 1 | 2 |
| 71 | Z | 2 | 3 |
| 72 | Z | 3 | 2 |
| 81 | NY2 | 1 | 0 |
| 82 | NY2 | 0 | 1 |
| 112 | NY2 | 1 | 0 |
| 113 | NY2 | 0 | 1 |
| 500 | X | 1 | 2 |
| 501 | X | 2 | 3 |
| 520 | Z | 2 | 3 |
| 521 | Z | 3 | 2 |
| 540 | NY2 | 1 | 0 |
| 541 | NY2 | 0 | 1 |
| 550 | Y1 | 0 | 1 |
| 551 | Y1 | 1 | 2 |
| 1000 | X | 3 | 2 |
| 1001 | X | 2 | 1 |
| 1020 | Z | 2 | 3 |
| 1021 | Z | 3 | 2 |
| 1031 | Y1 | 2 | 1 |
| 1032 | Y1 | 1 | 0 |
| 1040 | NY2 | 1 | 0 |
| 1041 | NY2 | 0 | 1 |
| 1051 | Y1 | 0 | 1 |
| 1052 | Y1 | 1 | 0 |
| 1071 | Y1 | 0 | 1 |
| 1072 | Y1 | 1 | 0 |
| 1081 | Y1 | 0 | 1 |
| 1082 | Y1 | 1 | 0 |
| 1091 | Y1 | 0 | 1 |
| 1092 | Y1 | 1 | 0 |
| 1101 | Y1 | 0 | 1 |

SIMULATION TERMINATED AT TIME 1101

Figure 6.14. TCM Simulation Showing Oscillatory Condition

changed and settled. This mode of operation results in smooth circuit response and state transitions.

As previously mentioned, the clock input for the clocked-type SIM-flops is a true four-valued input. In the previous simulation that used clocked RS SIM-flops, the clock was asserted in the binary fashion, i.e., 0 to 3 transitions, thus allowing both machines to change state as necessary. The same result can be obtained by asserting the clock at logic level 2 for the first three transitions of the subsequence in order to change only the left machine. For the final transition the clock is asserted at logic level 1 to allow the right machine transition, the end result being the final $y_1y_2 = 03$ state. The simulation of this TCM subsequence is shown in Figure 6.17.

The versatility of the clock input is better exemplified by using the first transition of the subsequence in Figure 6.10(a). For the transition as given both machines normally enter state $y_1y_2 = 03$ when X changes from 3 to 0. If the clock input is asserted at 2, only the left machine will change state, resulting in $y_1y_2 = 02$. Then if X is held at 1 and the clock does finally change to a value of 3, the TCM circuit will enter state $y_1y_2 = 03$. This action is shown in the simulator output of Figure 6.18. Note that the clock input rises to a value of 2, the machine enters state $y_1y_2 = 02$, the clock input returns to 0, and after the clock rises to a value of 3, the circuit enters state $y_1y_2 = 03$.

From the examples of circuit realization given above, it is seen that the TCM circuit using the RS SIM-flop, clocked or unclocked, will perform as expected. It is, of course, difficult and time consuming

QUATERNARY LOGIC SIMULATOR -- QLOSIM

TCM.EX5 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 100 | CLK | 0 | 1 |
| 101 | CLK | 1 | 2 |
| 102 | CLK | 2 | 3 |
| 131 | NY2 | 3 | 2 |
| 132 | NY2 | 2 | 1 |
| 151 | Y2 | 0 | 1 |
| 152 | Y2 | 1 | 2 |
| 161 | Z | 2 | 3 |
| 162 | Z | 3 | 2 |
| 171 | NY2 | 1 | 0 |
| 172 | NY2 | 0 | 1 |
| 212 | NY2 | 1 | 0 |
| 213 | NY2 | 0 | 1 |
| 400 | CLK | 3 | 2 |
| 401 | CLK | 2 | 1 |
| 402 | CLK | 1 | 0 |
| 500 | X | 1 | 2 |
| 501 | X | 2 | 3 |
| 520 | Z | 2 | 3 |
| 521 | Z | 3 | 2 |
| 600 | CLK | 0 | 1 |
| 601 | CLK | 1 | 2 |
| 602 | CLK | 2 | 3 |
| 651 | Y1 | 0 | 1 |
| 652 | Y1 | 1 | 2 |
| 900 | CLK | 3 | 2 |
| 901 | CLK | 2 | 1 |
| 902 | CLK | 1 | 0 |
| 1000 | X | 3 | 2 |
| 1001 | X | 2 | 1 |
| 1020 | Z | 2 | 3 |
| 1021 | Z | 3 | 2 |
| 1100 | CLK | 0 | 1 |
| 1101 | CLK | 1 | 2 |
| 1102 | CLK | 2 | 3 |
| 1132 | Y1 | 2 | 1 |
| 1133 | Y1 | 1 | 0 |
| 1400 | CLK | 3 | 2 |
| 1401 | CLK | 2 | 1 |

Figure 6.16. TCM Simulation Using Clocked RS SIM-Flops -- Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX5 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1402 | CLK | 1 | 0 |
| 1500 | X | 1 | 0 |
| 1520 | Z | 2 | 3 |
| 1600 | CLK | 0 | 1 |
| 1601 | CLK | 1 | 2 |
| 1602 | CLK | 2 | 3 |
| 1630 | NY2 | 1 | 0 |
| 1631 | NY2 | 0 | 1 |
| 1632 | NY2 | 1 | 0 |
| 1650 | Y2 | 2 | 3 |
| 1651 | Y2 | 3 | 2 |
| 1652 | Y2 | 2 | 3 |
| 1900 | CLK | 3 | 2 |
| 1901 | CLK | 2 | 1 |
| 1902 | CLK | 1 | 0 |

SIMULATION TERMINATED AT TIME 1912

Figure 6.16. TCM Simulation Using Clocked RS SIM-Flops - Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX6

SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 100 | CLK | 0 | 1 |
| 101 | CLK | 1 | 2 |
| 131 | NY2 | 3 | 2 |
| 132 | NY2 | 2 | 1 |
| 151 | Y2 | 0 | 1 |
| 152 | Y2 | 1 | 2 |
| 161 | Z | 2 | 3 |
| 162 | Z | 3 | 2 |
| 171 | NY2 | 1 | 0 |
| 172 | NY2 | 0 | 1 |
| 212 | NY2 | 1 | 0 |
| 213 | NY2 | 0 | 1 |
| 400 | CLK | 2 | 1 |
| 401 | CLK | 1 | 0 |
| 500 | X | 1 | 2 |
| 520 | Z | 2 | 3 |
| 521 | Z | 3 | 2 |
| 600 | CLK | 0 | 1 |
| 601 | CLK | L | 2 |
| 651 | Y1 | 0 | 1 |
| 652 | Y1 | 1 | 2 |
| 900 | CLK | 2 | 1 |
| 901 | CLK | 1 | 0 |
| 1000 | X | 3 | 2 |
| 1001 | X | 2 | 1 |
| 1020 | Z | 2 | 3 |
| 1021 | Z | 3 | 2 |
| 1100 | CLK | 0 | 1 |
| 1101 | CLK | 1 | 2 |
| 1132 | Y1 | 2 | 1 |
| 1133 | Y1 | 1 | 0 |
| 1400 | CLK | 2 | 1 |
| 1401 | CLK | 1 | 0 |
| 1500 | X | 1 | 0 |
| 1520 | Z | 2 | 3 |
| 1600 | CLK | 0 | 1 |
| 1601 | CLK | 1 | 2 |
| 1602 | CLK | 2 | 3 |

Figure 6.17. TCM Simulation of Figure 6.22(c) Subsequence
using Controlled Four-Valued Clock - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX6 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1630 | NY2 | 1 | 0 |
| 1631 | NY2 | 0 | 1 |
| 1650 | Y2 | 2 | 3 |
| 1651 | Y2 | 3 | 2 |
| 1652 | Y2 | 2 | 3 |
| 1900 | CLK | 3 | 2 |
| 1901 | CLK | 2 | 1 |
| 1902 | CLK | 1 | 0 |

SIMULATION TERMINATED AT TIME 1912

Figure 6.17. TCM Simulation of Figure 6.22(c) Subsequence
using Controlled Four-Valued Clock - Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIM

TCM.EX7 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | X | 3 | 2 |
| 2 | X | 2 | 1 |
| 3 | X | 1 | 0 |
| 21 | Z | 0 | 1 |
| 22 | Z | 1 | 2 |
| 23 | Z | 2 | 3 |
| 100 | CLK | 0 | 1 |
| 101 | CLK | 1 | 2 |
| 130 | NY2 | 3 | 2 |
| 131 | NY2 | 2 | 1 |
| 150 | Y2 | 0 | 1 |
| 151 | Y2 | 1 | 2 |
| 170 | NY2 | 1 | 0 |
| 171 | NY2 | 0 | 1 |
| 211 | NY2 | 1 | 0 |
| 212 | NY2 | 0 | 1 |
| 500 | CLK | 2 | 1 |
| 501 | CLK | 1 | 0 |
| 530 | NY2 | 1 | 0 |
| 531 | NY2 | 0 | 1 |
| 1000 | CLK | 0 | 1 |
| 1001 | CLK | 1 | 2 |
| 1002 | CLK | 2 | 3 |
| 1030 | NY2 | 1 | 0 |
| 1031 | NY2 | 0 | 1 |
| 1032 | NY2 | 1 | 0 |
| 1050 | Y2 | 2 | 3 |
| 1051 | Y2 | 3 | 2 |
| 1052 | Y2 | 2 | 3 |
| 1500 | CLK | 3 | 2 |
| 1501 | CLK | 2 | 1 |
| 1502 | CLK | 1 | 0 |

SIMULATION TERMINATED AT TIME 1512

Figure 6.18. TCM Simulation of Figure 6.10(a) Subsequence Transition Using Controlled Four-Valued Clock

to show all possible variations in input timing and gate delays, but for the conditions given, it is felt that sequential circuits can be successfully constructed using the methods given here. The design method for state integrated sequential circuits is basically a projection of a binary circuit design, new or existing, into a B_{2^m} realization. It is clear that the m machines within a B_{2^m} realization function independently of each other and realize m parallel binary machines.

The Register Transfer Circuit

The register transfer circuit is a general type of digital logic circuit used in central processing units [38]. This circuit was discussed in Chapter II and represents the mechanism by which digital information is moved from register to register. When the register circuit is supplemented with combinational and sequential circuits, the necessary building blocks are available for designing a computer processor. Because of this significance, the state integrated and multiplexed register transfer circuit will be investigated as a special type of sequential circuit. It will be seen that the state integrated property of simultaneous data storage within the circuit's SIM-flops is highly visible. To simplify the analysis of the circuit behavior, the two-machine or four-valued case is emphasized.

Figure 6.19 shows one method for implementing a simple state integrated and multiplexed register transfer circuit. Consider the circuit to be four-valued. This OR-Bus form of the circuit is quite similar to the corresponding binary version. Four two-bit registers are given where R1 and R2 are the source registers while R3 and R4 are the destination register. Conceivably, any source register can be a

destination register, and vice versa. In general, any register can be arranged to transfer information to any other register, including itself; however, to simplify the circuit, a distinction is made between the source and destination registers.

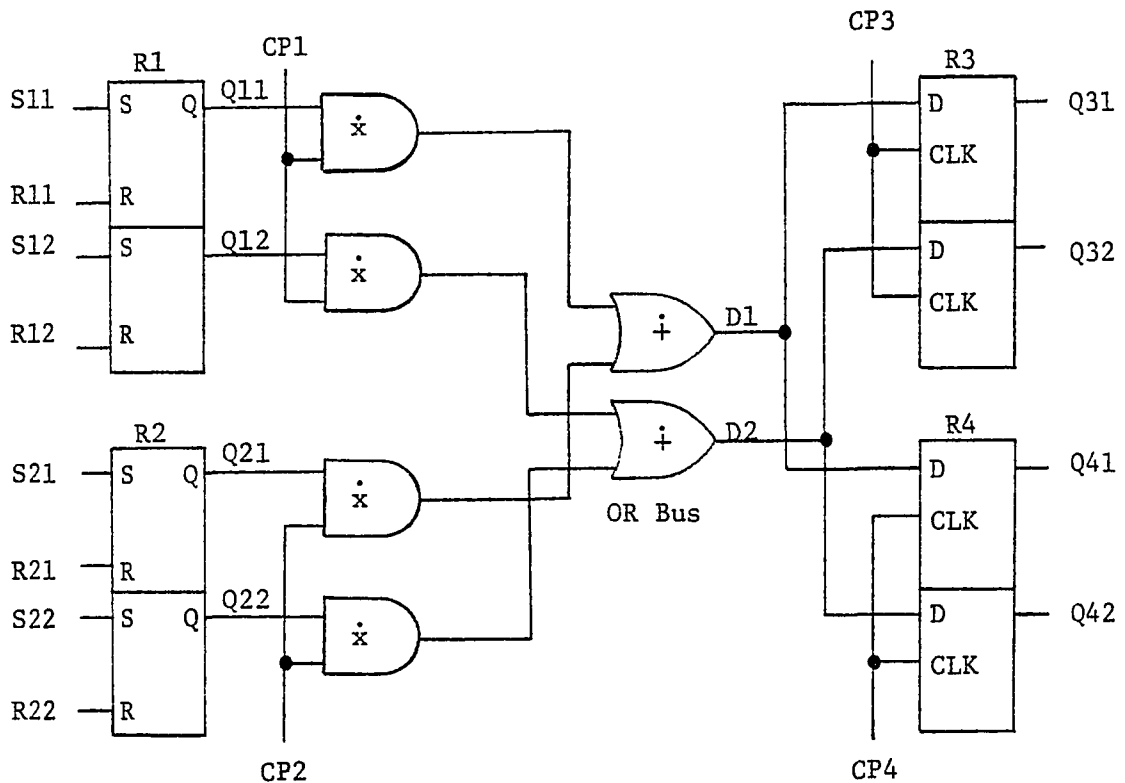


Figure 6.19. Four-Valued State Integrated and Multiplexed Register Transfer Circuit

The circuit operates by selecting a source register using either CP1 or CP2 which gates data from either R1 or R2, respectively, to the OR gates. The data is then strobed into a destination register, R3 or R4, by asserting either CP3 or CP4, respectively. For this

particular implementation, the CP3 and CP4 control signals can be tied directly to the clock inputs of the D-type SIM-flops. In the four-valued case, it is possible to transfer the contents of R1 or R2 to R3 or R4 for either the left or right machine. The left and right machine data transfers can occur simultaneously with differing source and destination registers. For example, data can transfer from R1 to R3 for the left machine while a simultaneous transfer occurs from R2 to R4 for the right machine. Data in the unaffected machine half of the destination registers remains unchanged by the operations. Note that the control pulses are multiplexed four-valued signals similar to the true four-valued clock signals previously described for the clocked SIM-flops. For example, to gate data into R3 for the right machine, CP3 has a logic value of one. To gate data for the left machine, CP3 has a logic value of 2, and to gate data for both machines, CP3 has a logic value of 3.

Table 6.1 summarizes the allowable transfer combinations for the register transfer circuit of Figure 6.19. The superscripts, L and R, represent the left and right machines, respectively. When only one "FROM" column entry appears in the table, for a given row, the "TO" column entries represent an inclusive OR transfer combination. For example, it is possible to transfer R_1^L to either R_3^L or R_4^L , or to both R_3^L and R_4^L . The other transfer combinations are given explicitly.

The four-valued register transfer circuit in Figure 6.19 is comprised of 50 gates and is simulated for the transfer combinations shown in Table 6.2. Successive rows of the table are cumulative and reflect the action of previous rows. Figure 6.20 gives the simulator output for these transfer combinations.

Table 6.1. Four-Valued Register Transfer Combinations

| <u>FROM</u> | <u>FROM</u> | <u>TO</u> | <u>TO</u> |
|-------------|-------------|------------|------------|
| R_1^L | - | R_3^L | R_4^L |
| R_1^R | - | R_3^R | R_4^R |
| R_2^L | - | R_3^L | R_4^L |
| R_2^R | - | R_3^R | R_4^R |
| R_1^L | R_1^R | R_3^L | R_4^R |
| R_1^L | R_1^R | R_4^L | R_3^R |
| R_2^L | R_2^R | R_3^L | R_4^R |
| R_2^L | R_2^R | R_4^L | R_3^R |
| R_1^L | R_2^R | R_3^L | R_4^R |
| R_1^L | R_2^R | R_4^L | R_3^R |
| R_1^R | R_2^L | R_3^R | R_4^L |
| R_1^R | R_2^L | R_4^R | R_3^L |
| R_1^{LR} | - | R_3^{LR} | R_4^{LR} |
| R_2^{LR} | - | R_3^{LR} | R_4^{LR} |

In row one of Table 6.2, R1 is set to 33 and the contents of the right side of R1 are transferred to R3. Figure 6.20 shows this result at Time 301. For this initial transfer, observe that control signal CP1 holds the source data active while CP3 is used as a strobe signal to clock the data into the destination register. Thus, CP1 is held active throughout the strobe time of CP3. All transfers are performed by using the control pulses in this manner.

Table 6.2. Simulated Transfers for the Four-Valued Register Transfer Circuit

| <u>Transfer No.</u> | <u>Source Registers</u> | | <u>Control Pulses</u> | | | | <u>Destination Registers</u> | |
|---------------------|-------------------------|-----------|-----------------------|------------|------------|------------|------------------------------|-----------|
| | <u>R1</u> | <u>R2</u> | <u>CP1</u> | <u>CP2</u> | <u>CP3</u> | <u>CP4</u> | <u>R3</u> | <u>R4</u> |
| 1 | 33 | 00 | 1 | 0 | 1 | 0 | 11 | 00 |
| 2 | 33 | 00 | 2 | 0 | 0 | 2 | 11 | 22 |
| 3 | 33 | 22 | 0 | 2 | 2 | 0 | 33 | 22 |
| 4 | 00 | 22 | 3 | 0 | 3 | 0 | 00 | 22 |
| 5 | 11 | 22 | 1 | 0 | 1 | 1 | 11 | 33 |
| 6 | 00 | 00 | 3 | 0 | 3 | 3 | 00 | 00 |
| 7 | 33 | 00 | 3 | 0 | 2 | 1 | 22 | 11 |
| 8 | 33 | 00 | 0 | 3 | 3 | 3 | 00 | 00 |
| 9 | 33 | 33 | 2 | 1 | 1 | 2 | 11 | 22 |

The next data transfer moves the data for the left side of R1 to the right side of R4 and is essentially completed at Time 603. Prior to the third transfer, register R2 is set to a value of 22. R2 is then transferred to the left side of R3 giving a value of R3 = 33 at Time 1123. Note that the right side of R3 was not affected by this data movement. For this particular transfer, the R3 D-type SIM-flop is not well behaved and actually results in oscillation when the gate transport delays are equal. This is a similar condition as was previously discussed for the RS SIM-flop. The instability is cured, as before, by unbalancing the gate delays, although the resulting behavior is still slightly erratic.

The next transfer is row four of Table 6.2 and shows that both sides of R1 are cleared, followed by a transfer of this result to R3. This is shown in the simulator output at Time 1583. The fifth transfer is performed by setting the right side of R1 and then simultaneously transferring the right side of R1 to both R3 and R4. This gives R3 = 11 and R4 = 33 at Time 2101. Again, note that the left side of R4 was unchanged by the right side transfer. The sixth transfer clears R1 and R2 and moves R1 to both R3 and R4, giving a result of 00 for both at Time 2583.

The last three data transfers clearly show the multiplexing capability of the circuit. The seventh row of the table shows that the left side of R1 is transferred to R3 and, simultaneously, the right side of R1 is transferred to R4. R3 and R4 reach their values of 22 and 11 at Times 3101 and 3102, respectively. In the eighth transfer the 00 value of R2 is used to clear both R3 and R4 simultaneously. R3 and R4 are clear at Time 2583. The final data movement begins by setting both R1 and R2 to a value of 33. The left side of R1 is transferred to R4, and at the same time the right side of R2 is transferred to R3. R3 and R4 are equal to 11 and 22 at Times 3901 and 3902.

These results strongly imply that the state integrated and multiplexed register transfer circuit is logically sound and suggest that circuits of this type can be used in a computer central processing unit to achieve simultaneous movement of data for two independent machines operating within the same logic circuitry. Functionally, these results can be extended to the case of m binary machines operating in parallel. Again, no detailed predictions can be made on the specific

operation of the circuit, but it should be possible to choose $m = 3$ and realize a three-machine register transfer circuit much the same way, if not exactly the same way, as for $m = 2$.

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S11 | 0 | 1 |
| 1 | S12 | 0 | 1 |
| 2 | S11 | 1 | 2 |
| 2 | S12 | 1 | 2 |
| 3 | S11 | 2 | 3 |
| 3 | S12 | 2 | 3 |
| 41 | Q11 | 0 | 1 |
| 41 | Q12 | 0 | 1 |
| 42 | Q11 | 1 | 2 |
| 42 | Q12 | 1 | 2 |
| 43 | Q11 | 2 | 3 |
| 43 | Q12 | 2 | 3 |
| 100 | S11 | 3 | 2 |
| 100 | S12 | 3 | 2 |
| 101 | S11 | 2 | 1 |
| 101 | S12 | 2 | 1 |
| 102 | S11 | 1 | 0 |
| 102 | S12 | 1 | 0 |
| 200 | CP1 | 0 | 1 |
| 220 | D2 | 0 | 1 |
| 220 | D1 | 0 | 1 |
| 250 | CP3 | 0 | 1 |
| 301 | Q32 | 0 | 1 |
| 301 | Q31 | 0 | 1 |
| 350 | CP3 | 1 | 0 |
| 400 | CP1 | 1 | 0 |
| 420 | D2 | 1 | 0 |
| 420 | D1 | 1 | 0 |
| 500 | CP1 | 0 | 1 |
| 501 | CP1 | 1 | 2 |
| 520 | D2 | 0 | 1 |
| 520 | D1 | 0 | 1 |
| 521 | D2 | 1 | 2 |
| 521 | D1 | 1 | 2 |
| 550 | CP4 | 0 | 1 |
| 551 | CP4 | 1 | 2 |
| 602 | Q42 | 0 | 1 |
| 602 | Q41 | 0 | 1 |
| 603 | Q42 | 1 | 2 |
| 603 | Q41 | 1 | 2 |
| 650 | CP4 | 2 | 1 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 651 | CP4 | 1 | 0 |
| 700 | CP1 | 2 | 1 |
| 701 | CP1 | 1 | 0 |
| 720 | D2 | 2 | 1 |
| 720 | D1 | 2 | 1 |
| 721 | D2 | 1 | 0 |
| 721 | D1 | 1 | 0 |
| 800 | S21 | 0 | 1 |
| 800 | S22 | 0 | 1 |
| 801 | S21 | 1 | 2 |
| 801 | S22 | 1 | 2 |
| 830 | Q21 | 0 | 1 |
| 831 | Q21 | 1 | 2 |
| 840 | Q22 | 0 | 1 |
| 841 | Q22 | 1 | 2 |
| 900 | S21 | 2 | 1 |
| 900 | S22 | 2 | 1 |
| 901 | S21 | 1 | 0 |
| 901 | S22 | 1 | 0 |
| 1000 | CP2 | 0 | 1 |
| 1001 | CP2 | 1 | 2 |
| 1021 | D2 | 0 | 1 |
| 1021 | D1 | 0 | 1 |
| 1022 | D2 | 1 | 2 |
| 1022 | D1 | 1 | 2 |
| 1050 | CP3 | 0 | 1 |
| 1051 | CP3 | 1 | 2 |
| 1081 | Q32 | 1 | 0 |
| 1081 | Q31 | 1 | 0 |
| 1082 | Q32 | 0 | 1 |
| 1082 | Q31 | 0 | 1 |
| 1103 | Q32 | 1 | 2 |
| 1103 | Q31 | 1 | 2 |
| 1104 | Q32 | 2 | 3 |
| 1104 | Q31 | 2 | 3 |
| 1111 | Q32 | 3 | 2 |
| 1111 | Q31 | 3 | 2 |
| 1112 | Q32 | 2 | 3 |
| 1112 | Q31 | 2 | 3 |
| 1122 | Q32 | 3 | 2 |
| 1122 | Q31 | 3 | 2 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1123 | Q32 | 2 | 3 |
| 1123 | Q31 | 2 | 3 |
| 1150 | CP3 | 2 | 1 |
| 1151 | CP3 | 1 | 0 |
| 1181 | Q32 | 3 | 2 |
| 1181 | Q31 | 3 | 2 |
| 1182 | Q32 | 2 | 3 |
| 1182 | Q31 | 2 | 3 |
| 1200 | CP2 | 2 | 1 |
| 1201 | CP2 | 1 | 0 |
| 1220 | D2 | 2 | 1 |
| 1220 | D1 | 2 | 1 |
| 1221 | D2 | 1 | 0 |
| 1221 | D1 | 1 | 0 |
| 1300 | R11 | 0 | 1 |
| 1300 | R12 | 0 | 1 |
| 1301 | R11 | 1 | 2 |
| 1301 | R12 | 1 | 2 |
| 1302 | R11 | 2 | 3 |
| 1302 | R12 | 2 | 3 |
| 1320 | Q11 | 3 | 2 |
| 1320 | Q12 | 3 | 2 |
| 1321 | Q11 | 2 | 1 |
| 1321 | Q12 | 2 | 1 |
| 1322 | Q11 | 1 | 0 |
| 1322 | Q12 | 1 | 0 |
| 1400 | R11 | 3 | 2 |
| 1400 | R12 | 3 | 2 |
| 1401 | R11 | 2 | 1 |
| 1401 | R12 | 2 | 1 |
| 1402 | R11 | 1 | 0 |
| 1402 | R12 | 1 | 0 |
| 1500 | CP1 | 0 | 1 |
| 1501 | CP1 | 1 | 2 |
| 1502 | CP1 | 2 | 3 |
| 1550 | CP3 | 0 | 1 |
| 1551 | CP3 | 1 | 2 |
| 1552 | CP3 | 2 | 3 |
| 1581 | Q32 | 3 | 2 |
| 1581 | Q31 | 3 | 2 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 3

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1582 | Q32 | 2 | 1 |
| 1582 | Q31 | 2 | 1 |
| 1583 | Q32 | 1 | 0 |
| 1583 | Q31 | 1 | 0 |
| 1650 | CP3 | 3 | 2 |
| 1651 | CP3 | 2 | 1 |
| 1652 | CP3 | 1 | 0 |
| 1700 | CP1 | 3 | 2 |
| 1701 | CP1 | 2 | 1 |
| 1702 | CP1 | 1 | 0 |
| 1800 | S11 | 0 | 1 |
| 1800 | S12 | 0 | 1 |
| 1840 | Q11 | 0 | 1 |
| 1840 | Q12 | 0 | 1 |
| 1900 | S11 | 1 | 0 |
| 1900 | S12 | 1 | 0 |
| 2000 | CP1 | 0 | 1 |
| 2020 | D2 | 0 | 1 |
| 2020 | D1 | 0 | 1 |
| 2050 | CP3 | 0 | 1 |
| 2050 | CP4 | 0 | 1 |
| 2101 | Q32 | 0 | 1 |
| 2101 | Q31 | 0 | 1 |
| 2101 | Q42 | 2 | 3 |
| 2101 | Q41 | 2 | 3 |
| 2150 | CP3 | 1 | 0 |
| 2150 | CP4 | 1 | 0 |
| 2200 | CP1 | 1 | 0 |
| 2220 | D2 | 1 | 0 |
| 2220 | D1 | 1 | 0 |
| 2300 | R11 | 0 | 1 |
| 2300 | R12 | 0 | 1 |
| 2300 | R21 | 0 | 1 |
| 2300 | R22 | 0 | 1 |
| 2301 | R21 | 1 | 2 |
| 2301 | R22 | 1 | 2 |
| 2320 | Q11 | 1 | 0 |
| 2320 | Q12 | 1 | 0 |
| 2321 | Q21 | 2 | 1 |
| 2321 | Q22 | 2 | 1 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 4

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 2322 | Q21 | 1 | 0 |
| 2322 | Q22 | 1 | 0 |
| 2400 | R11 | 1 | 0 |
| 2400 | R12 | 1 | 0 |
| 2400 | R21 | 2 | 1 |
| 2400 | R22 | 2 | 1 |
| 2401 | R21 | 1 | 0 |
| 2401 | R22 | 1 | 0 |
| 2500 | CP1 | 0 | 1 |
| 2501 | CP1 | 1 | 2 |
| 2502 | CP1 | 2 | 3 |
| 2550 | CP3 | 0 | 1 |
| 2550 | CP4 | 0 | 1 |
| 2551 | CP3 | 1 | 2 |
| 2551 | CP4 | 1 | 2 |
| 2552 | CP3 | 2 | 3 |
| 2552 | CP4 | 2 | 3 |
| 2581 | Q32 | 1 | 0 |
| 2581 | Q31 | 1 | 0 |
| 2581 | Q42 | 3 | 2 |
| 2581 | Q41 | 3 | 2 |
| 2582 | Q32 | 0 | 1 |
| 2582 | Q31 | 0 | 1 |
| 2582 | Q42 | 2 | 1 |
| 2582 | Q41 | 2 | 1 |
| 2583 | Q32 | 1 | 0 |
| 2583 | Q31 | 1 | 0 |
| 2583 | Q42 | 1 | 0 |
| 2583 | Q41 | 1 | 0 |
| 2650 | CP3 | 3 | 2 |
| 2650 | CP4 | 3 | 2 |
| 2651 | CP3 | 2 | 1 |
| 2651 | CP4 | 2 | 1 |
| 2652 | CP3 | 1 | 0 |
| 2652 | CP4 | 1 | 0 |
| 2700 | CP1 | 3 | 2 |
| 2701 | CP1 | 2 | 1 |
| 2702 | CP1 | 1 | 0 |
| 2800 | S11 | 0 | 1 |
| 2800 | S12 | 0 | 1 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 5

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 2801 | S11 | 1 | 2 |
| 2801 | S12 | 1 | 2 |
| 2802 | S11 | 2 | 3 |
| 2802 | S12 | 2 | 3 |
| 2840 | Q11 | 0 | 1 |
| 2840 | Q12 | 0 | 1 |
| 2841 | Q11 | 1 | 2 |
| 2841 | Q12 | 1 | 2 |
| 2842 | Q11 | 2 | 3 |
| 2842 | Q12 | 2 | 3 |
| 2900 | S11 | 3 | 2 |
| 2900 | S12 | 3 | 2 |
| 2901 | S11 | 2 | 1 |
| 2901 | S12 | 2 | 1 |
| 2902 | S11 | 1 | 0 |
| 3000 | CP1 | 0 | 1 |
| 3001 | CP1 | 1 | 2 |
| 3002 | CP1 | 2 | 3 |
| 3020 | D2 | 0 | 1 |
| 3020 | D1 | 0 | 1 |
| 3021 | D2 | 1 | 2 |
| 3021 | D1 | 1 | 2 |
| 3022 | D2 | 2 | 3 |
| 3022 | D1 | 2 | 3 |
| 3050 | CP3 | 0 | 1 |
| 3050 | CP4 | 0 | 1 |
| 3051 | CP3 | 1 | 2 |
| 3101 | Q32 | 0 | 1 |
| 3101 | Q31 | 0 | 1 |
| 3101 | Q42 | 0 | 1 |
| 3101 | Q41 | 0 | 1 |
| 3102 | Q32 | 1 | 2 |
| 3102 | Q31 | 1 | 2 |
| 3150 | CP3 | 2 | 1 |
| 3150 | CP4 | 1 | 0 |
| 3151 | CP3 | 1 | 0 |
| 3200 | CP1 | 3 | 2 |
| 3201 | CP1 | 1 | 0 |
| 3202 | CP1 | 1 | 0 |
| 3220 | D2 | 3 | 2 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 6

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 3220 | D1 | 3 | 2 |
| 3221 | D2 | 2 | 1 |
| 3221 | D1 | 2 | 1 |
| 3222 | D2 | 1 | 0 |
| 3222 | D1 | 1 | 0 |
| 3300 | CP2 | 0 | 1 |
| 3301 | CP2 | 1 | 2 |
| 3302 | CP2 | 2 | 3 |
| 3350 | CP3 | 0 | 1 |
| 3350 | CP4 | 0 | 1 |
| 3351 | CP3 | 1 | 2 |
| 3351 | CP4 | 1 | 2 |
| 3352 | CP3 | 2 | 3 |
| 3352 | CP4 | 2 | 3 |
| 3381 | Q42 | 1 | 0 |
| 3381 | Q41 | 1 | 0 |
| 3382 | Q32 | 2 | 1 |
| 3382 | Q31 | 2 | 1 |
| 3382 | Q42 | 0 | 1 |
| 3382 | Q41 | 0 | 1 |
| 3383 | Q32 | 1 | 0 |
| 3383 | Q31 | 1 | 0 |
| 3383 | Q42 | 1 | 0 |
| 3383 | Q41 | 1 | 0 |
| 3450 | CP3 | 3 | 2 |
| 3450 | CP4 | 3 | 2 |
| 3451 | CP3 | 2 | 1 |
| 3451 | CP4 | 2 | 1 |
| 3452 | CP3 | 1 | 0 |
| 3452 | CP4 | 1 | 0 |
| 3500 | CP2 | 3 | 2 |
| 3501 | CP2 | 2 | 1 |
| 3502 | CP2 | 1 | 0 |
| 3600 | S21 | 0 | 1 |
| 3600 | S22 | 0 | 1 |
| 3601 | S21 | 1 | 2 |
| 3601 | S22 | 1 | 2 |
| 3602 | S21 | 2 | 3 |
| 3602 | S22 | 2 | 3 |
| 3640 | Q21 | 0 | 1 |
| 3640 | Q22 | 0 | 1 |

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 7

QUATERNARY LOGIC SIMULATOR - QLOSIM

RICKT.EX1 SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 3641 | Q21 | 1 | 2 |
| 3641 | Q22 | 1 | 2 |
| 3642 | Q21 | 2 | 3 |
| 3642 | Q22 | 2 | 3 |
| 3700 | S21 | 3 | 2 |
| 3700 | S22 | 3 | 2 |
| 3701 | S21 | 2 | 1 |
| 3701 | S22 | 2 | 1 |
| 3702 | S21 | 1 | 0 |
| 3702 | S22 | 1 | 0 |
| 3800 | CP1 | 0 | 1 |
| 3800 | CP2 | 0 | 1 |
| 3801 | CP1 | 1 | 2 |
| 3820 | D2 | 0 | 1 |
| 3820 | D1 | 0 | 1 |
| 3821 | D2 | 1 | 2 |
| 3821 | D1 | 1 | 2 |
| 3822 | D2 | 2 | 3 |
| 3822 | D1 | 2 | 3 |
| 3850 | CP3 | 0 | 1 |
| 3850 | CP4 | 0 | 1 |
| 3851 | CP4 | 1 | 2 |
| 3901 | Q32 | 0 | 1 |
| 3901 | Q31 | 0 | 1 |
| 3901 | Q42 | 0 | 1 |
| 3901 | Q41 | 0 | 1 |
| 3902 | Q42 | 1 | 2 |
| 3902 | Q41 | 1 | 2 |
| 3950 | CP3 | 1 | 0 |
| 3950 | CP4 | 2 | 1 |
| 3951 | CP4 | 1 | 0 |
| 4000 | CP1 | 2 | 1 |
| 4000 | CP2 | 1 | 0 |
| 4001 | CP1 | 1 | 0 |
| 4020 | D2 | 3 | 2 |
| 4020 | D1 | 3 | 2 |
| 4021 | D2 | 2 | 1 |
| 4021 | D1 | 2 | 1 |
| 4022 | D2 | 1 | 0 |
| 4022 | D1 | 1 | 0 |

SIMULATION TERMINATED AT TIME 4032

Figure 6.20. Four-Valued Register Transfer Circuit Simulation
Part 8

CHAPTER VII

CONCLUSION

Summary of Results

This thesis introduces and examines the concept of state integrated and multiplexed multiple-valued digital systems. The logic circuit design methods for the multiple-valued B_{2^m} , $m \geq 2$, cases were considered, and $B_2 \times B_2$ was studied in detail. An important result is that the four-valued $B_2 \times B_2$ circuits are realizable directly from the binary design techniques. The actual behavior of the four-valued circuits was studied by the application of the quaternary logic simulator, QLOSIM. This simulator was developed for purposes of this thesis but can be applied to other studies of four-valued digital systems.

The circuit operating characteristics assumed in the thesis are believed to be reasonable predictions of how actual gates would perform. Hence, the overall results obtained from the logic circuits considered herein suggest that the state integrated and multiplexed concept is feasible.

The design of four-valued state integrated and multiplexed combinational circuits is relatively straightforward with some circuit

behavior phenomena being quite similar to the binary case. For example, static hazards are eliminated by the same methods used for binary circuits. The same result was found to be true for sequential circuits except that the sequential circuit problem necessitated the development of a new memory element, the SIM-flop. The stability of the four-valued SIM-flop was studied for various conditions. Three SIM-flop types, the RS, clocked RS, and D-type, were applied in sequential circuits and shown via simulation to perform adequately. The state integrated and multiplexed register transfer circuit operation was also examined, and quite good results were obtained from it. In total these findings give a complete and detailed account of the logic circuit operation for a two-machine state integrated and multiplexed digital system. Of particular importance is that all of the circuits investigated can be generated directly from the binary circuit design, and the resulting state integrated and multiplexed circuit will replace two or more copies of the binary circuit.

Suggestions for Further Research

To provide an adequate foundation for state integrated and multiplexed digital systems, this thesis necessarily covers a wide range of topics in the area of multiple-valued digital design. Additional work can extend the basic concepts presented here. Using tools such as the quaternary logic simulator, further work can be done in the system design area toward the specification of a machine organization based on this concept. For example, by constructing a computer control unit from four-valued TCM circuits and supplementing this with $B_2 \times B_2$

arithmetic and register transfer circuits, the specification of a fairly complete state integrated and multiplexed central processing unit is possible. Such a machine actually represents two independent binary machines and would be capable of executing two software programs simultaneously. The architectural and system level advantages gained from this capability deserve further attention. In addition, this work represents the first step toward the design of a dual radix hardware computer system.

A more detailed investigation of B_{2^m} combinational and sequential circuits can be undertaken. Further work on the various SIM-flops and their characterization should be considered. However, the need for actual gate hardware realization is significant. I^2L circuit designs were given for the AND and OR gates, but more work is necessary to generate results for these or functionally equivalent circuits.

The quaternary logic simulator provided valuable insights into the possible behavior of the state integrated and multiplexed digital circuits. In addition, the simulator was used to study four-valued algebraic relationships. It is expected that the simulator will likewise prove valuable for other work, but since the total needs of these studies are not known presently, the need for additional software enhancement is likely.

Certainly, this thesis provides many of the answers for the design and analysis of state integrated and multiplexed digital systems. Strong evidence that such systems will perform is presented, but many questions still remain. Lest we continue to pose them all, humbly note that,

"The outcome of any serious research can only be to make two questions grow where only one grew before." - Thorstein Veblen.

LIST OF REFERENCES

1. Thurber, K. J., "Parallel Processor Architectures - Part 1: General Purpose Systems," Computer Design, Vol. 18, pp. 89-97, Jan. 1979.
2. Bouknight, W. J., Denenburg, S. A., Randall, J. M., Sameh, A. H., and Slotnick, D. L., "The Illiac IV System," Proc. IEEE, Vol. 60, pp. 369-388, Apr. 1972.
3. Dao, T. T., "Threshold I^2L and its Applications to Binary Symmetric Functions and Multivalued Logic," IEEE J. of Solid State Circuits, Vol. SC-12, pp. 463-472, October 1977.
4. Dao, T. T., McCluskey, E. J., and Russell, L. K., "Multivalued Integrated Injection Logic," IEEE Trans. Comput., Vol. C-26, pp. 1233-1241, Dec. 1977.
5. Smith, K. C., "Circuits for Multiple Valued Logic-- A Tutorial and Appreciation," Proc. Sixth Int. Symp. on Multiple-Valued Logic, Utah State Univ., Logan, Utah, 1976, pp. 30-43.
6. Post, E. L., "Introduction to a General Theory of Elementary Propositions," American J. Math, Vol. 43, pp. 163-165, 1921.
7. Allen, C. M. and Givone, "A Minimization Technique for Multiple-Valued Logic Systems," IEEE Trans. Comput., Vol. C-17 pp. 182-184, Feb. 1968.
8. Su, S. Y. H. and Cheung, P. T., "Computer Simplification of Multi-Valued Switching Functions," Computer Science and Multiple-Valued Logic, Chapter 7, D. C. Rine (Ed.), North-Holland, 1977.

9. Smith, W. R., "Minimization of Multivalued Functions," Computer Science and Multiple-Valued Logic, Chapter 8, D. C. Rine (Ed.), North-Holland, 1977.
10. Allen, C. M. and Givone, D. D., "The Allen-Givone Implementation Oriented Algebra," Computer Science and Multiple-Valued Logic, Chapter 9, D. C. Rine (Ed.), North-Holland, 1977.
11. McCluskey, E. J., "Logic Design of Multi-Valued I²L Logic Circuits," Proc. Eighth Int. Symp. on Multiple-Valued Logic, Chicago, Ill., 1978, pp. 14-22.
12. McCluskey, E. J., "Logic Design of Multivalued I²L Logic Circuits," IEEE Trans. Comput., Vol. C-28, pp. 546-559, Aug. 1979.
13. Wojcik, A. S., "Multi-Valued Asynchronous Circuits," Proc. 1973 Int. Symp. on Multiple-Valued Logic, Univ. of Toronto, pp. 217-227.
14. Wojcik, A. S., "Multi-Valued Asynchronous Sequential Circuits," Proc. 1974 Int. Symp. on Multiple-Valued Logic, West Virginia Univ., pp. 401-411.
15. Sheafor, S. J., The Design of Multiple-Valued Asynchronous Sequential Circuits, Ph.D. Thesis, Univ. of Illinois, Urbana, 1974.
16. Irving, T. A. and Nagle, H. T., "An Approach to Multi-Valued Sequential Logic," Proc. 1973 Int. Symp. on Multiple-Valued Logic, Univ. of Toronto, pp. 89-105.
17. Sintonen, L., "A Clocked Multivalued Flip-Flop," IEEE Trans. Comput., Vol. C-26, pp. 292-294, Mar. 1977.
18. Huertas, J. J., Acha, J. I. and Gomez, G. S., "Theory and Design of Multivalued Memory Elements," Proc. Eighth Int. Symp. on Multiple-Valued Logic, Chicago, Ill., 1978, pp. 213-220.
19. Wills, M. S., "A Behavior Model and Triggering Modes for MVL R-Flops," Proc. Eighth Int. Symp. on Multiple-Valued Logic, Chicago, Ill., 1978, pp. 226-234.
20. Moraga, C. and Guitierrez, J., "Critical Races in Fundamental Mode Ternary Sequential Machines," Proc. 1974 Int. Symp. on Multiple-Valued Logic, West Virginia Univ., pp. 401-411.

21. Wojcik, A. S., "An Analysis of Some Relationships Between Post and Boolean Algebras," J. of ACM, Vol. 21, pp. 680-696, Oct. 1974.
22. Wojcik, A. S. and Metze, G., "An Analysis of Some Relationships Between Post and Boolean Algebras with Application to the Minimization of Higher-Order Boolean Functions," Report R-541, Coordinated Science Laboratory, Univ. of Illinois, Urbana, Dec. 1971.
23. Lee, S. C., Modern Switching Theory and Digital Design, Prentice-Hall, Englewood Cliffs, NJ., 1978.
24. Breuer, M. A. and Friedman, A. D., Diagnosis & Reliable Design of Digital Systems, Computer Science Press, Woodland Hills, CA, 1976.
25. Chappell, S. G., Elmendorf, C. H., and Schmidt., L. D., "LAMP: Logic Circuit Simulators," Bell System Tech. J., Vol. 53, pp. 1451-1476, Oct. 1974.
26. Harris, J. A. and Smith, D. R., Computer Structure Language (CSL), State Univ. of New York at Stony Brook, 1975.
27. Moraga, C., "CARMEN-I: An Approach to Simulation of Ternary Digital Systems," CAD Conference, Sussex, 1979.
28. Rammig, F. J., Vogel, A. T., and Moraga, C., "Six-Valued Simulation of Ternary Switching Circuits," Proc. Seventh Int. Symp. on Multiple-Valued Logic, Univ. of North Carolina, Charlotte, 1977, pp. 51-54.
29. MCS-85 Handbook, Intel Corp., Santa Clara, CA, 1977.
30. Edwards, C. R., "I²L Threshold Circuits for Binary-Quaternary Encoding and Decoding," Int. J. of Electronics, Vol. 44, pp. 445-448, Apr. 1978.
31. Adkins, G. W. and Pooch, U. W., "Computer Simulation: A Tutorial," Computer, Vol. 10, pp. 12-17, Apr. 1977.
32. Singh, A. D., Armstrong, J. R., and Gray, F. G., "Combinational and Sequential Multivalued Logic Design Using Universal Iterative Tree Structures," Proc. Ninth Int. Symp. on Multiple-Valued Logic, Bath, England, 1979, pp. 182-189.

33. Tolstova, Y. M., "Modeling L-Valued Logic in K-Valued Logic ($K > L$)," (Russian) Problemy Kibernet, 18, pp. 67-82, 1966, English Translation - Systems Theory Research, 19, pp. 62-77, 1968, Mathematical Reviews, 41, No. 41.
34. Su, S. Y., and Sarris, A. A., "The Relationship Between Multivalued Switching Algebra and Boolean Algebra Under Different Definitions of Complement," IEEE Trans. Comput., Vol. C-21, May 1972, pp. 479-485.
35. Smith, E. J., and Kohavi, Z., "Synthesis of Multiple Sequential Machines," Proc. of Seventh Annual Symp. on Switching and Automata Theory, 1966, pp. 160-171.
36. Givone, D. D., Introduction to Switching Circuit Theory, McGraw-Hill, New York, N.Y., 1970.
37. The TTL Data Book for Design Engineers, Second Edition, Texas Instruments, Inc., 1976.
38. Hill, F. J. and Peterson, G. R., Digital Systems: Hardware Organization and Design, John Wiley and Sons, New York, N.Y., 1973.

APPENDIX A

SIMULATOR USER'S GUIDE

Simulator Capabilities

The quaternary logic simulator, QLOSIM, is a gate level simulator designed to simulate four-valued logic circuits exclusively. The software described here was designed to operate on a DECSys-10 computer, using the TOPS-10 operating system.

Several gate types comprise the simulator's library. Each gate type used in a circuit possesses a unique identification. Gates such as MIN, MAX, AND and OR require more than one input, and QLOSIM permits two, three, or four inputs for these gates. Table A.1 lists each type of gate supported. The user should consult thesis Chapter III for specific function definitions.

The version of QLOSIM given in Appendix B limits the size of the simulated circuit to 100 gates or 200 nodes. Neither constraint can be violated. Every circuit node must possess a unique identification. The node identification is any one to five character string selected by the user. Each circuit must be initialized, using the input file, by specifying logic values for all circuit input nodes. If the circuit

Table A.1. QLOSIM Gate Types

| <u>ID</u> | <u>Function</u> |
|-----------|-------------------------------|
| MAX2 | Two input MAX |
| MAX3 | Three input MAX |
| MAX4 | Four input MAX |
| MIN2 | Two input MIN |
| MIN3 | Three input MIN |
| MIN4 | Four input MIN |
| COMP | Complement or strong negation |
| AND2 | Two input AND |
| AND3 | Three input AND |
| AND4 | Four input AND |
| OR2 | Two input OR |
| OR3 | Three input OR |
| OR4 | Four input OR |
| CYCL1 | Cycle (a = 1) |
| CYCL2 | Cycle (a = 2) |
| CYCL3 | Cycle (a = 3) |
| PLUS | Two input <u>PLUS</u> |
| NMAX2 | Two input <u>MAX</u> |
| NMAX3 | Three input <u>MAX</u> |
| NMAX4 | Four input <u>MAX</u> |
| NMIN2 | Two input <u>MIN</u> |
| NMIN3 | Three input <u>MIN</u> |
| NMIN4 | Four input <u>MIN</u> |
| NAND2 | Two input NAND |
| NAND3 | Three input NAND |
| NAND4 | Four input NAND |
| NOR2 | Two input NOR |
| NOR3 | Three input NOR |
| NOR4 | Four input <u>NOR</u> |
| NPLUS | Two input <u>PLUS</u> |
| 1LT00 | 1X ^{0,0} Literal |
| 1LT11 | 1X ^{1,1} Literal |
| 1LT22 | 1X ^{2,2} Literal |
| 1LT33 | 1X ^{3,3} Literal |
| 1LT01 | 1X ^{0,1} Literal |
| 1LT02 | 1X ^{0,2} Literal |

Table A.1. QLOSIM Gate Types (continued)

| <u>ID</u> | <u>Function</u> |
|-----------|-----------------------------|
| 1LT12 | 1X ^{1,2} Literal |
| 1LT13 | 1X ^{1,3} Literal |
| 1LT23 | 1X ^{2,3} Literal |
| 2LT00 | 2X ^{0,0} Literal |
| 2LT11 | 2X ^{1,1} Literal |
| 2LT22 | 2X ^{2,2} Literal |
| 2LT33 | 2X ^{3,3} Literal |
| 2LT01 | 2X ^{0,1} Literal |
| 2LT02 | 2X ^{0,2} Literal |
| 2LT12 | 2X ^{1,2} Literal |
| 2LT13 | 2X ^{1,3} Literal |
| 2LT23 | 2X ^{2,3} Literal |
| 3LT00 | 3X ^{0,0} Literal |
| 3LT11 | 3X ^{1,1} Literal |
| 3LT22 | 3X ^{2,2} Literal |
| 3LT33 | 3X ^{3,3} Literal |
| 3LT01 | 3X ^{0,1} Literal |
| 3LT02 | 3X ^{0,2} Literal |
| 3LT12 | 3X ^{1,2} Literal |
| 3LT13 | 3X ^{1,3} Literal |
| 3LT23 | 3X ^{2,3} Literal |
| 1INH2 | Two input INHIBIT (B = 1) |
| 1INH3 | Three input INHIBIT (B = 1) |
| 1INH4 | Four input INHIBIT (B = 1) |
| 2INH2 | Two input INHIBIT (B = 2) |
| 2INH3 | Three input INHIBIT (B = 2) |
| 2INH4 | Four input INHIBIT (B = 2) |
| 3INH2 | Two input INHIBIT (B = 3) |
| 3INH3 | Three input INHIBIT (B = 3) |
| 3INH4 | Four input INHIBIT (B = 3) |
| 1UNIV | Universal gate (B = 1) |
| 2UNIV | Universal gate (B = 2) |
| 3UNIV | Universal gate (B = 3) |
| MUX41 | Four-to-one Multiplexer |

contains feedback nodes, these must also be initialized. If the initialization is not performed, a simulation failure will occur.

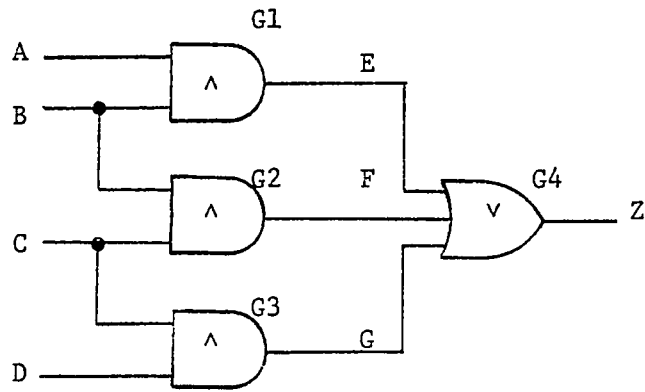
QLOSIM will initialize the circuit at Time 0 by propagating the initial circuit conditions through the circuit. Once this process is successfully completed, the dynamic input node changes are applied to the circuit and the results are reported as selected by the user.

The simulator always scans the input data for accuracy. Several errors can be detected by the program, but the logical correctness of the circuit data is the user's responsibility. Error messages are summarized in a later section.

Data Preparation

The circuit input data file is prepared prior to simulator execution and is generally done using a text editor. The input file consists of four parts, (1) the circuit topology, (2) the initial node conditions, (3) the dynamic input node changes, and (4) the list of circuit nodes to be reported. The input file is constructed using the normal rules for FORTRAN list directed I/O.

The preparation of a data file is best illustrated using an example. Figure A.1 shows a four-valued logic circuit and the resulting input data file. The circuit is labeled with all information necessary for generating the circuit topology data. The circuit topology records for the example circuit are given as the first four lines of the data file. Each of the first four lines consists of several fields. The first field is the gate identification. The second field is the gate type and must be one of the types given in Table A.1.



(a)

```

'G1' 'MIN2' 10 'A' 'B' 'E' /
'G2' 'MIN2' 10 'B' 'C' 'F' /
'G3' 'MIN2' 10 'C' 'D' 'G' /
'G4' 'MAX3' 10 'E' 'F' 'G' 'Z' /
'$$' /
'A' 0
'B' 0
'C' 0
'D' 0
'$$' /
'A' 1,2 400,3 /
'B' 100,1 200,2 300,3 /
'C' 300,3 400,2 500,1 600,0 700,2
'$$' /
'A' 'B' 'C' 'D' 'Z' /
'$$' /

```

(b)

Figure A.1. Example Circuit and Input Data

The third field is a numeric entry that gives the gate transport delay. The remaining fields are the input and output nodes for the gate. The gate's output node is always specified as the last node field, and the total quantity of node fields must correspond to the particular gate type. Up to seven node fields are allowed, but only the universal gate utilizes seven nodes. For all other gates the "/" (forward slash) is used to terminate the circuit topology record. The space between each field is necessary as a field separator. The carriage return or new line characters appearing at the end of records are ignored. The "'\$\$' /" record terminates the circuit topology data.

The second group of data records provide the circuit initialization information. Each record consists of a single circuit node and its associated initial logic value. These records are terminated by the "'\$\$' /".

The third portion of the input data file is the dynamic input node information. Only one node can be given per record. Each record gives the circuit input node and up to five numeric time-value/logic-value pairs. The commas are optional and, if omitted, a space must be used as the field delimiter. When fewer than five pairs are given on a line, the record is terminated with a "/". When a node requires more than five entries, as is the case for node C in Figure A.1(b), additional records are entered. The time values entered can be either absolute time or they can be time values relative to the previous time value. Either method can be selected, but only one method can be used for a given data file. As before, the "'\$\$' /" terminator denotes the end of the dynamic node change information.

The last group of data records specifies the nodes to be reported in the simulator output. A maximum of ten nodes can be specified per record. The final "'\$\$' /" is required.

Program Execution

It is assumed in the following that the input data file has been prepared prior to the execution of the simulator. The notation for the dialogue below has all user responses underlined. All responses are terminated by typing a new line character. The simulator prompts the user for parameter information by way of the timesharing terminal. The simulation report, however, is sent to the line printer. If the user desires the output report on the terminal, the ASSIGN command should be used to redirect the output.

The simulator is invoked with the RUN command by typing RU QLOSIM . The program types out

QUATERNARY LOGIC SIMULATOR - QLOSIM

ENTER FILE NAME CONTAINING CIRCUIT DATA:

The user responds by entering the file name of the input data file that was prepared earlier. The next prompt requests the method of treating the time values contained in the dynamic node change records. The request is

HOW ARE TIME VALUES GIVEN FOR DYNAMIC LOGIC CHANGES?

1 = ABSOLUTE

2 = RELATIVE TO PREVIOUS CHANGE

The user responds by entering 1 or 2 depending on how the data was prepared. Next, the method of reporting is requested by

ENTER REPORT FORMAT STYLE:

1 = EACH REQUESTED NODE CHANGE REPORTED

2 = ALL NODES REPORTED AT SPECIFIED TIME INCREMENTS

If the response is 1, the simulator will print node changes as they occur for those nodes to be reported. A response of 2 means that the simulator will print values for all reportable nodes at successive time values. If 2 is entered, the program will follow with a request to enter the time value. For example, if 100 is entered, the simulator will report node logic values at every 100 time units. Actually, the report will occur at the time of the next logic change which is greater than or equal to a multiple of 100 time units. The next input request is for the rise/fall time delay between successive logic values and is requested by

ENTER RISE/FALL TIME DELAY PER LOGIC LEVEL CHANGE:

The user response is a numeric time delay value greater than zero. The stop condition is selected next by

ENTER STOP CONDITION:

1 = UNTIL ALL VALUES HAVE PROPAGATED

2 = SPECIFIED TIME LIMIT

An entry of 1 will allow the simulator to propagate all input conditions until no further circuit changes occur. If 2 is entered, the program will prompt for a time value entry that gives the termination time. The second option is especially good if the user is uncertain of the stability of the circuit and needs to control the duration of the simulator execution. The final input parameter is the selection of the timing

diagram option. The five possibilities are requested by

ENTER TIMING DIAGRAM OPTION:

0 = NO DIAGRAM GENERATED

1 = 80 COLUMN PAGED OUTPUT DIAGRAM

2 = 80 COLUMN CONTINUOUS OUTPUT DIAGRAM

3 = 132 COLUMN PAGED OUTPUT DIAGRAM

4 = 132 COLUMN CONTINUOUS OUTPUT DIAGRAM

The options are fairly self-explanatory. The paged output diagram option will place a page heading on each page of the timing diagram, whereas the continuous option will not. The 80 column report places up to six nodes across the report while the 132 column can print up to ten nodes across the report page. The choices, of course, give the user flexibility with regard to his particular printer or terminal.

Figure A.2 gives a complete execution output for a sample circuit. This output was obtained by assigning the printer output to the user's terminal. Parts 2 through 5 would normally comprise the portion of the report sent to the line printer.

Error Condition Summary

Table A.2 gives a tabulation of the simulator error messages and a brief supplemental description for some. Unless otherwise specified, the errors are fatal and cause immediate termination of the simulator execution. Normally, the location of the error message in the output report gives an indication of the invalid condition. Some of the error conditions are the result of internal checks that are used for debugging purposes. These error messages should not occur and are so labeled.

RU QLOSIH

QUATERNARY LOGIC SIMULATOR - QLOSIH

ENTER FILE NAME CONTAINING CIRCUIT DATA:

SRSFF7.DAT

HOW ARE TIME VALUES GIVEN FOR DYNAMIC LOGIC CHANGES ?

1 = ABSOLUTE

2 = RELATIVE TO PREVIOUS CHANGE

1

ENTER REPORT FORMAT STYLE:

1 = EACH REQUESTED NODE CHANGE REPORTED

2 = ALL NODES REPORTED AT SPECIFIED TIME INCREMENTS

1

ENTER RISE/FALL TIME DELAY PER LOGIC LEVEL CHANGE:

1

ENTER STOP CONDITION:

1 = UNTIL ALL VALUES HAVE PROPAGATED

2 = SPECIFIED TIME LIMIT

1

ENTER TIMING DIAGRAM OPTION:

0 = NO DIAGRAM GENERATED

1 = 80 COLUMN PAGED OUTPUT DIAGRAM

2 = 80 COLUMN CONTINUOUS OUTPUT DIAGRAM

3 = 132 COLUMN PAGED OUTPUT DIAGRAM

4 = 132 COLUMN CONTINUOUS OUTPUT DIAGRAM

1

Figure A.2. Illustration of a Complete Simulator Output - Part 1

QUATERNARY LOGIC SIMULATOR - QLOSIM

QLOSIM INPUT DATA

FILE = SRSFF7.DAT

| GATE ID | GATE TYPE | GATE DELAY | GATE NODES | | | |
|---------|-----------|------------|------------|----|----|--|
| G1 | OR2 | 10 | R | NQ | A | |
| G2 | OR2 | 10 | S | Q | B | |
| G3 | NMAX2 | 10 | A | NQ | Q | |
| G4 | NMAX2 | 10 | B | Q | NQ | |

INITIAL NODE CONDITIONS:

| | |
|---------|---------------|
| NODE R | LOGIC VALUE 0 |
| NODE S | LOGIC VALUE 0 |
| NODE Q | LOGIC VALUE 1 |
| NODE NQ | LOGIC VALUE 2 |

DYNAMIC NODE CHANGES:

| NODE ID | TIME | VALUE | TIME | VALUE | TIME | VALUE | TIME | VALUE | TIME | VALUE |
|---------|------|-------|------|-------|------|-------|------|-------|------|-------|
| S | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

NODES REPORTED:

S R Q NQ A B

INPUT PARAMETER SUMMARY:

DYNAMIC CHANGE TIME VALUES ARE ABSOLUTE

ALL (REQUESTED) NODE CHANGES REPORTED

RISE/FALL TIME PER LOGIC LEVEL CHANGE = 1

SIMULATION STOPS WHEN ALL CHANGES HAVE PROPAGATED

TIMING DIAGRAM GENERATED

END INPUT DATA REPORT

Figure A.2. Illustration of a Complete Simulator Output - Part 2

QUATERNARY LOGIC SIMULATOR - QLOSIH

SRSFF7.DAT INITIAL CIRCUIT CONDITIONS:

NODE R INITIAL VALUE = 0
 NODE NQ INITIAL VALUE = 2
 NODE A INITIAL VALUE = 2
 NODE S INITIAL VALUE = 0
 NODE Q INITIAL VALUE = 1
 NODE B INITIAL VALUE = 1

QUATERNARY LOGIC SIMULATOR - QLOSIH

SRSFF7.DAT SIMULATION RESULTS:

| TIME | NODE | PREV. VALUE | NEW VALUE |
|------|------|-------------|-----------|
| 1 | S | 0 | 1 |
| 1 | R | 0 | 1 |
| 2 | S | 1 | 2 |
| 11 | A | 2 | 3 |
| 12 | B | 1 | 2 |
| 13 | B | 2 | 3 |
| 21 | Q | 1 | 0 |
| 22 | NQ | 2 | 1 |
| 23 | NQ | 1 | 0 |
| 31 | B | 3 | 2 |
| 32 | A | 3 | 2 |
| 33 | A | 2 | 1 |
| 41 | NQ | 0 | 1 |
| 42 | Q | 0 | 1 |
| 43 | Q | 1 | 2 |
| 52 | B | 2 | 3 |
| 53 | B | 3 | 2 |
| 62 | NQ | 1 | 0 |
| 63 | NQ | 0 | 1 |

SIMULATION TERMINATED AT TIME 63

Figure A.2. Illustration of a Complete Simulator Output - Part 3

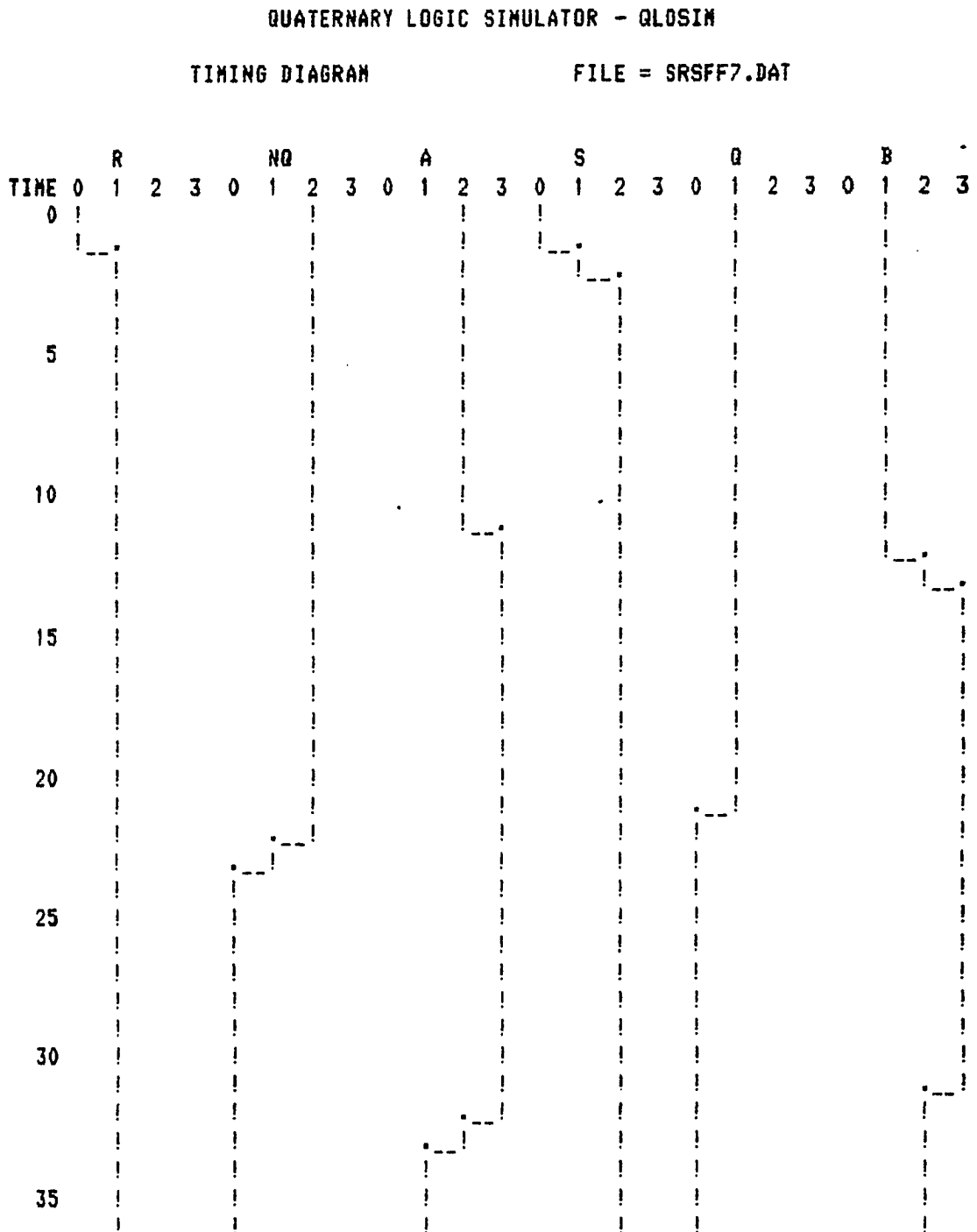
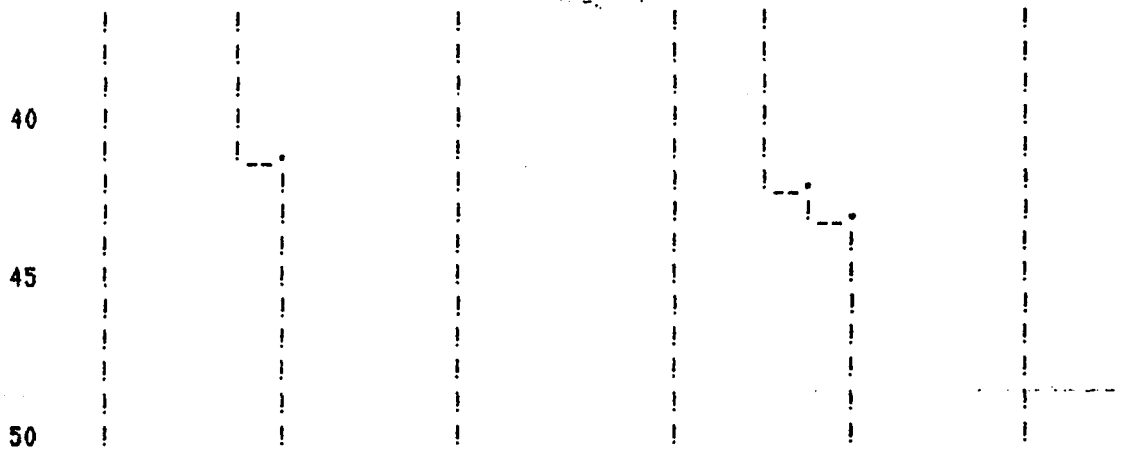


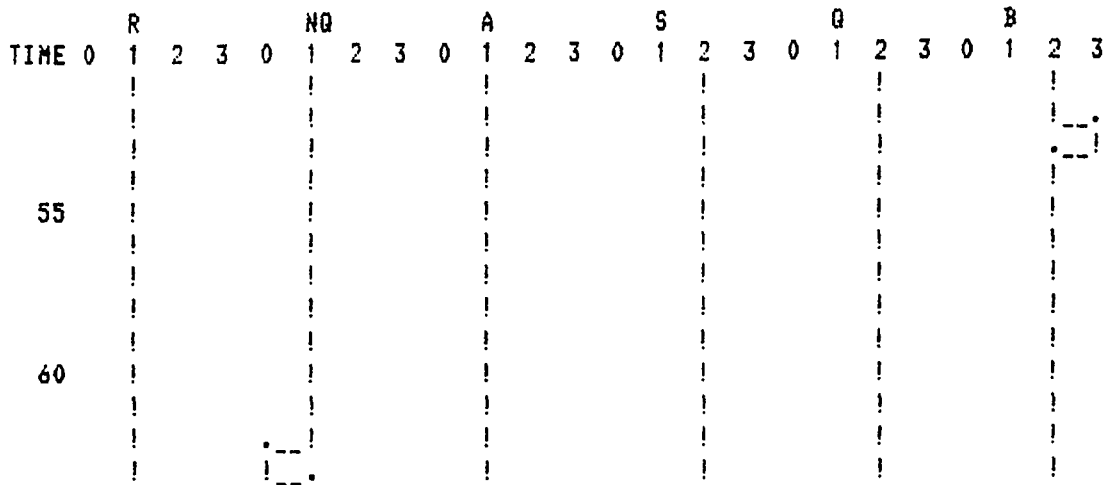
Figure A.2. Illustration of a Complete Simulator Output - Part 4



QUATERNARY LOGIC SIMULATOR - QLOSIM

TIMING DIAGRAM

FILE = SRSFF7.DAT



END OF EXECUTION

CPU TIME: 1.00 ELAPSED TIME: 5:22.65

EXIT

Figure A.2. Illustration of a Complete Simulator Output - Part 5

Table A.2. Error Message Summary

| <u>Error No.</u> | <u>Message Text</u> | <u>Comment</u> |
|------------------|---------------------------------------|--|
| 1 | MISSING GATE ID, TYPE, OR DELAY | Check input data. |
| 2 | INVALID GATE TYPE | Type is not one of those given in Table A.1. |
| 3 | INVALID NODE ID OR NODE QTY | Check node fields. |
| 4 | CIRCUIT ARRAYS CAPACITY EXCEEDED | More than 100 gates or 200 nodes. |
| 6 | INITIAL CONDITION NODE INVALID | Check node ID. |
| 7 | INITIAL CONDITION LOGIC VALUE INVALID | |
| 8 | TIME-EVENT QUEUE OVERFLOW | |
| 9 | DYNAMIC NODE ID INVALID | Check node ID. |
| 10 | DYNAMIC CHANGE LOGIC VALUE INVALID | |
| 11 | DYNAMIC NODE NOT INITIALIZED | Initialize the node. |
| 12 | REPORT NODE INVALID - SKIPPED | Warning. |
| 13 | PREMATURE END OF INPUT DATA | |
| 15 | INITIALIZATION SIMULATION FAILURE | Check node initialization. |
| 16 | INVALID GATE INDEX | Should not occur. |
| 17 | TIME-EVENT QUEUE STRUCTURE PROBLEM | Should not occur. |
| 18 | TIME-EVENT QUEUE OUT OF SEQUENCE | Should not occur. |
| 19 | LINK ERROR BETWEEN NODE & T-E LISTS | Should not occur. |
| 21 | LOGIC CHANGE GENERATED > 4 | Should not occur. |
| 22 | GATE OUTPUT EVALUATION INVALID | Should not occur. |

APPENDIX B

SIMULATOR SOURCE LISTING

The QLOSIM simulator FORTRAN source listing is provided on the following pages. Note that the software is copyrighted and is the property of the Western Electric Company, Incorporated. Permission to use the software for research or other academic purposes is freely granted.

```

C
C
C QUATERNARY LOGIC SIMULATOR - QLOSIM
C DECSYSTEM-10 VERSION
C ISSUE 1.    MARCH, 1979
C ISSUE 1.1  MAY, 1979 - ADD TIMING DIAGRAM GENERATOR.
C AUTHOR HP TULL
C COPYRIGHT APRIL 1,1979 MONTE P. TULL
C ALL RIGHTS TO THIS PROGRAM AND ASSOCIATED SUBROUTINES
C BELONG TO WESTERN ELECTRIC CO, INC.
C
      IMPLICIT INTEGER (A-Z)
      REAL*8 FILNAM
C NODE LIST ARRAYS
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      1NLTEPT(200)
C GATE LIST ARRAYS
      COMMON /GLIST/GLGATE(100),GLTYPE(100),GLDLAY(100),
      1GLINPT(100),GLONPT(100)
C TIME-EVENT LIST - DOUBLY LINKED
      COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
      1TENCEL(500),TENCNS(500)
C INPUT NODE LIST
      COMMON /INLIST/INPTR(300),INNGPT(300)
C MISCELLANEDUS COMMON AREAS
      COMMON /AREA1/NLNUH,GLNUH,INNUH,ERR,RPTSW,RPTIME,
      1DMVTIM,STOPSW,STOPTM,GATEMX,NLMAX,GLMAX,INMAX
      COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TINTYP
C
      DIMENSION GTPTAB(70),GNQTY(70)
C
C GATE TYPE TABLE
      DATA GTPTAB /'MAX2','MAX3','MAX4','MIN2','MIN3','MIN4',
      1'COMP','AND2','AND3','AND4','OR2','OR3','OR4','CYCL1',
      2'CYCL2','CYCL3','PLUS','NMAX2','NMAX3','NMAX4','NMIN2',
      3'NMIN3','NMIN4','NAND2','NAND3','NAND4','NOR2','NOR3',
      4'NOR4','NPLUS','1LT00','1LT11','1LT22','1LT33','1LT01',
      5'1LT02','1LT12','1LT13','1LT23','2LT00','2LT11','2LT22',
      6'2LT33','2LT01','2LT02','2LT12','2LT13','2LT23','3LT00',
      7'3LT11','3LT22','3LT33','3LT01','3LT02','3LT12','3LT13',
      8'3LT23','1INH2','1INH3','1INH4','2INH2','2INH3','2INH4',
      9'3INH2','3INH3','3INH4','1UNIV','2UNIV','3UNIV','MUX41'/
C GATE NODE QUANTITY TABLE
      DATA GNQTY/3,4,5,3,4,5,2,3,4,5,3,4,5,2,2,2,3,
      13,4,5,3,4,5,3,4,5,3,4,5,3,2,2,2,2,2,2,2,2,2,
      22,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
      33,4,5,3,4,5,3,4,5,7,7,7,6/
C
C INITIAL DIALOGUE - DETERMINE SOURCE OF DATA

```



```

C
100 WRITE(5,1)
1   FORMAT(10X,'QUATERNARY LOGIC SIMULATOR - QLOSIH',/)
   WRITE(5,2)
2   FORMAT(' ENTER FILE NAME CONTAINING CIRCUIT DATA:',/)
   READ(5,3)FILNAM
3   FORMAT(A10)
C ENTER RUN TIME PARAMETERS
101 WRITE(5,11)
11  FORMAT(' HOW ARE TIME VALUES GIVEN FOR DYNAMIC LOGIC',
          1' CHANGES ?',/,5X,'1 = ABSOLUTE',/,
          25X,'2 = RELATIVE TO PREVIOUS CHANGE',/)
   READ(5,*)TINTYP
   IF(TINTYP.NE.1 .AND. TINTYP.NE.2)GO TO 101
102 WRITE(5,4)
4   FORMAT(' ENTER REPORT FORMAT STYLE:',/,5X,
          1'1 = EACH REQUESTED NODE CHANGE REPORTED',/,5X,
          2'2 = ALL NODES REPORTED AT SPECIFIED TIME INCREMENTS',/)
   READ(5,*)RPTSW
   IF(RPTSW.EQ.1)GO TO 110
   IF(RPTSW.NE.2)GO TO 102
112 WRITE(5,6)
6   FORMAT(' ENTER TIME INCREMENT:',/)
   READ(5,*)RPTIME
   IF(RPTIME.GT.0)GO TO 110
   WRITE(5,15)
15  FORMAT(' VALUE MUST BE POSITIVE')
   GO TO 112
110 WRITE(5,8)
8   FORMAT(' ENTER RISE/FALL TIME DELAY PER LOGIC LEVEL',
          1' CHANGE:',/)
   READ(5,*)DMVTIM
   IF(DMVTIM.GE.0)GO TO 111
   WRITE(5,15)
   GO TO 110
111 WRITE(5,9)
9   FORMAT(' ENTER STOP CONDITION:',/,5X,
          1'1 = UNTIL ALL VALUES HAVE PROPAGATED',/,5X,
          2'2 = SPECIFIED TIME LIMIT',/)
   READ(5,*)STOPSW
   IF(STOPSW.EQ.1)GO TO 120
   IF(STOPSW.NE.2)GO TO 111
   WRITE(5,10)
10  FORMAT(' ENTER STOP TIME:',/)
   READ(5,*)STOPTM
C
120 WRITE(5,16)
16  FORMAT(' ENTER TIMING DIAGRAM OPTION:',/,5X,
          1'0 = NO DIAGRAM GENERATED',/,5X,
          2'1 = 80 COLUMN PAGED OUTPUT DIAGRAM',/,5X,
          3'2 = 80 COLUMN CONTINUOUS OUTPUT DIAGRAM',/,5X,
          .....

```

```

4'3 = 132 COLUMN PAGED OUTPUT DIAGRAM',/,5X,
5'4 = 132 COLUMN CONTINUOUS OUTPUT DIAGRAM',/)
READ(5,*)FRMAT
IF(FRMAT.LT. 0 .OR. FRMAT.GT.4) GO TO 120
IF(FRMAT.EQ.0) GO TO 126
C DELETE OLD TIMING DATA FILE
OPEN(UNIT=21,DEVICE='DSK',ACCESS='SEQIN',
FILE='TIMING.DAT',ERR=121)
CLOSE(UNIT=21,DISPOSE='DELETE')
121 OPEN(UNIT=21,DEVICE='DSK',ACCESS='SEQOUT',FILE='TIMING.DAT')
C
C INITIALIZE NODE, GATE, & TIME-EVENT LISTS
126 GATEMX=70
NLNUM=0
NLMAX=200
GLNUM=0
GLMAX=100
INNUM=0
INMAX=300
TENUM=0
TEMAX=500
TELOW=1
TEHIGH=1
TEFREE=1
ERR=1
CALL LINIT
C
C READ IN CIRCUIT DATA & LOAD NODE & GATE LISTS
CALL INDATA(FILNAM,GTPTAB,GNQTY,FRMAT)
C
C INITIALIZE THE CIRCUIT
CALL CINIT
C
13 FORMAT(5X,A10,' INITIAL CIRCUIT CONDITIONS:',/)
LINE=99
TIME=0
DO 130 J=1,NLNUM
IF(LINE.LE.50)GO TO 125
WRITE(3,14)
14 FORMAT(1H1,22X,'QUATERNARY LOGIC SIMULATOR - QLOSIH',/)
WRITE(3,13)FILNAM
LINE=0
125 LVAL=MOD(NLLVAL(J),256)
WRITE(3,12)NLNODE(J),LVAL
12 FORMAT(5X,'NODE ',A5,' INITIAL VALUE = ',I1)
LINE = LINE + 1
IF(LINE.LE.50)GO TO 135
WRITE(3,14)
WRITE(3,13)FILNAM
LINE = 0
135 IF(FRMAT.EQ.0) GO TO 130

```

```

        WRITE(21,17)TIME,NLNODE(J),LVAL
17      FORMAT(I7,A5,I1)
130     CONTINUE
C
C SIMULATE THE CIRCUIT
      CALL SIMUL(FILNAM,FRMAT)
C
      IF(FRMAT.EQ.0) GO TO 500
      CLOSE(UNIT=21)
      CALL TIMGEN(FILNAM,FRMAT)
C
500     CALL EXIT
      END
C
C
C LINIT ROUTINE INITIALIZES THE TIME-EVENT DATA STRUCTURE.
C THE TIME-EVENT LIST IS DOUBLY LINKED. -1 IS KEPT IN
C UNALLOCATED T-E TIME CELLS.
C
      SUBROUTINE LINIT
      IMPLICIT INTEGER (A-Z)
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      1NLTEPT(200)
      COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
      1TENCEL(500),TENCASN(500)
C
      TENCSN(1)=0 ; TETIME(1)=-1
      TENCEL(1)=2
      TENCEL(500)=0 ; TETIME(500)=-1
      TENCASN(500)=499
      DO 100 J=2,499
      TETIME(J)=-1
      TENCEL(J)=J+1
      TENCSN(J)=0
100     CONTINUE
C CLEAR NODE LIST LOGIC VALUE ARRAY
      DO 110 J=1,200
      NLTEPT(J)=-1
      NLLVAL(J)=0
110     CONTINUE
      RETURN
      END
C
C
C INDATA ROUTINE READS THE CIRCUIT DESCRIPTION DATA & STORES
C THE DATA IN THE VARIOUS STRUCTURES. ALL DATA IS REPORTED.
C ALL INITIAL CONDITIONS ARE PROPAGATED THRU THE CIRCUIT TO
C ESTABLISH INITIAL STATIC NODE VALUES.
C
      SUBROUTINE INDATA(FILNAM,GTPTAB,GNQTY,FRMAT)
      IMPLICIT INTEGER (A-Z)

```



```

      IF(NODE(M+1).NE.' ')GO TO 420
C LOAD GATE ARRAY
131  GLNUM=GLNUM+1
      IF(GLNUM.GT.GLMAX)GO TO 430
      GLGATE(GLNUM)=GATE
      GLTYPE(GLNUM)=J
      GLDLAY(GLNUM)=GLLAY
C LOCATE NODES IN NODE LIST - K COUNTS NUM OF NODES IN RECORD
      K = 1
135  CALL FNDNOD(NODE(K),L)
      IF (L.NE.0)GO TO 150
C NODE NOT FOUND IN NODE LIST - ADD IT
      NLNUM=NLNUM+1
      IF(NLNUM.GT.NLMAX)GO TO 430
      NLNODE(NLNUM)=NODE(K)
      NLLVAL(NLNUM)=99
      NLIGPT(NLNUM)=0
      IF(K.EQ.M)GO TO 145
C SAVE INPUT NODE DATA IN THE INPUT NODE LIST
      NLIGPT(NLNUM)=GLNUM
      INNUM=INNUM+1
      IF(INNUM.GT.INMAX)GO TO 430
      INPTR(INNUM)=NLNUM
      INNGPT(INNUM)=0
      K=K+1
      GO TO 135
C FINISH UPDATE OF GATE LIST FOR OUTPUT NODES & INLIST POINTERS
145  GLONPT(GLNUM)=NLNUM
      GO TO 147
146  GLONPT(GLNUM)=L
C PACK INPUT NODE QTY WITH INPUT NODE LIST POINTER
147  GLINPT(GLNUM)=((INNUM - (M-2)) * 256) + (M-1)
      GO TO 100
C
C NODE WAS FOUND IN NODE LIST - UPDATE INPUT NODE LIST &
C CHANGE POINTER TO PREVIOUS GATE ON THIS NODE.
C IF K = M, THIS IS AN OUTPUT NODE.
150  IF(K.EQ.M)GO TO 146
      INNUM=INNUM+1
      IF(INNUM.GT.INMAX)GO TO 430
      INPTR(INNUM)=L
      INNGPT(INNUM)=NLIGPT(L)
      NLIGPT(L)=GLNUM
      K=K+1
      GO TO 135
C
C INPUT INITIAL CONDITIONS
200  LINE=LINE+3
      IF(LINE.LE.50)GO TO 205
      LINE=3
      WRITE(3,5)FILNAM

```

```

5      FORMAT(1H1,22X,'QUATERNARY LOGIC SIMULATOR - QLOSIM',//,
        118X,'QLOSIM INPUT DATA',10X,'FILE = ',A10)
205     WRITE(3,6)
6      FORMAT(1H0,' INITIAL NODE CONDITIONS:',/)
210     READ(20,*,END=999)NID,LVAL
        IF(NID.EQ.'$$')GO TO 230
        LINE=LINE+1
        IF(LINE.LE.50)GO TO 211
        LINE = 0
        WRITE(3,5)FILNAM
        WRITE(3,6)
211     WRITE(3,8)NID,LVAL
8      FORMAT(10X,'NODE ',A5,5X,'LOGIC VALUE ',I1)
        IF(LVAL.LT.0 .OR. LVAL.GT.3)GO TO 460
C SCAN NODE LIST FOR VALID NODE
        CALL FNDNOD(NID,J)
        IF(J.EQ.0)GO TO 450
C
C ADD INITIAL VALUE CONDITION TO NODE LIST
220     NLLVAL(J)=LVAL
        GO TO 210
C
C INPUT DYNAMIC LOGIC CHANGES. UPDATE TIME-EVENT QUEUE
C FOR EACH CHANGE. ERROR EXISTS IF A DYNAMIC CHANGE
C IS SPECIFIED FOR A NON-INITIALIZED NODE.
230     LINE=LINE+6
        IF(LINE.LE.50)GO TO 235
        LINE=3
        WRITE(3,5)FILNAM
235     WRITE(3,9)
9      FORMAT(1H0,' DYNAMIC NODE CHANGES:',//,
        12X,'NODE ID',5(2X,'TIME VALUE'))
C CLEAR PREVIOUS NODE VARIABLE - PNID
        PNID=' '
C CLEAR INPUT TIME & LOGIC VALUE LISTS
240     DO 241 J=1,5
        DTIME(J)=0
        DVAL(J)=0
241     CONTINUE
        READ(20,*,END=999)NID,(DTIME(J),DVAL(J),J=1,5)
        IF(NID.EQ.'$$')GO TO 300
        LINE=LINE+1
        IF(LINE.LE.50)GO TO 245
        LINE=4
        WRITE(3,5)FILNAM
        WRITE(3,9)
245     WRITE(3,11)NID,(DTIME(J),DVAL(J),J=1,5)
11     FORMAT(3X,A5,3X,5(I4,5X,I1,4X))
C VERIFY INPUT DATA - K COUNTS QTY OF LOGIC CHANGE ENTRIES
        K=0
        DO 250 J=1,5

```

```

        IF(DVAL(J).LT.0 .OR. DVAL(J).GT.3)GO TO 490
        IF(DTIME(J).LE.0)GO TO 260
        K=K+1
250    CONTINUE
C FIND NODE IN NODE LIST
260    CALL FNDNOD(NID,J)
        IF(J.EQ.0)GO TO 480
C NODE WAS FOUND - CHECK THAT IT WAS INITIALIZED
        IF(NLLVAL(J).EQ.99)GO TO 500
C
C INSERT DYNAMIC CHANGES IN T-E QUEUE. MULTIVALUED
C CHANGES ARE BROKEN DOWN INTO INDIVIDUAL T-E ENTRIES.
C T-E QUEUE INSERT SUBROUTINE PERFORMS THE QUEUE INSERTION.
280    IF(K.EQ.0)GO TO 240
C CHECK IF PREVIOUS NODE IS SAME AS CURRENT NODE
        IF(PNID.EQ.NID)GO TO 281
        ITIME=0
        PVAL=NLLVAL(J)
281    DO 290 L=1,K
        IF(TIMTYP.EQ.1)GO TO 283
        ITIME=ITIME+DTIME(L)
        GO TO 284
283    ITIME=DTIME(L)
C COMPUTE MVAL = NUMBER OF LOGIC LEVEL CHANGES
284    MVAL=ABS(PVAL - DVAL(L))
        IF(MVAL.EQ.0)GO TO 290
        DO 285 M=1,MVAL
C DETERMINE DIRECTION OF CHANGE
        IF(PVAL.GT.DVAL(L))GO TO 286
        NVAL=PVAL+M
        GO TO 287
286    NVAL=PVAL-M
287    CALL TEINST(J,ITIME,NVAL)
        ITIME=ITIME+DMVTIM
285    CONTINUE
        PVAL=DVAL(L)
290    CONTINUE
        PNID=NID
        GO TO 240
C
C READ IN NODES TO BE REPORTED
300    IF(LINE.LE.35)GO TO 310
        WRITE(3,5)FILNAM
310    WRITE(3,12)
12    FORMAT(1H0," NODES REPORTED:",/)
C CLEAR INPUT NODE LIST
350    DO 315 L=1,10
        NIDLST(L)=" "
315    CONTINUE
        READ(20,*,END=360)(NIDLST(J),J=1,10)
        IF(NIDLST(1).EQ.'$$ ')GO TO 360

```

```

DO 320 L=1,10
IF(NIDLST(L).EQ.' ')GO TO 320
CALL FNDNOD(NIDLST(L),J)
IF(J.NE.0)GO TO 325
C ERROR 12 - REPORT NODE INVALID - NON-FATAL
ERR=0
ERRNO=12
CALL ERROR(ERRNO,'REPORT NODE INVALID - SKIPPED ')
ERR=1
GO TO 320
C SET REPORT FLAG IN NODE LIST
325 NLLVAL(J)=NLLVAL(J) + 256
320 CONTINUE
WRITE(3,15)(NIDLST(J),J=1,10)
15 FORMAT(5X,10(A5,2X))
GO TO 350
C
C FINISH PARAMETER REPORT
360 WRITE(3,25)
25 FORMAT(1H0,' INPUT PARAMETER SUMMARY:')
IF(TIMYP.EQ.1)GO TO 361
WRITE(3,22)
22 FORMAT(1H0,5X,'DYNAMIC CHANGE TIME VALUES ARE RELATIVE')
GO TO 362
361 WRITE(3,23)
23 FORMAT(1H0,5X,'DYNAMIC CHANGE TIME VALUES ARE ABSOLUTE')
362 IF(RPTSW.EQ.1)GO TO 365
WRITE(3,16)RPTIME
16 FORMAT(1H0,5X,'REPORT GENERATED EACH ',I4,' TIME UNITS')
GO TO 370
365 WRITE(3,17)
17 FORMAT(1H0,5X,'ALL (REQUESTED) NODE CHANGES REPORTED')
370 WRITE(3,18)DHVTIM
18 FORMAT(1H0,5X,'RISE/FALL TIME PER LOGIC LEVEL CHANGE = ',I3)
IF (STOPSW.EQ.1)GO TO 380
WRITE(3,19)STOPTM
19 FORMAT(1H0,5X,'SIMULATION STOPS AT TIME = ',I7)
GO TO 390
380 WRITE(3,20)
20 FORMAT(1H0,5X,'SIMULATION STOPS WHEN ALL CHANGES HAVE',
1' PROPAGATED')
390 IF(FRMT.EQ.0) GO TO 391
WRITE(3,26)
26 FORMAT(1H0,5X,'TIMING DIAGRAM GENERATED')
GO TO 395
391 WRITE(3,27)
27 FORMAT(1H0,5X,'NO TIMING DIAGRAM GENERATED')
395 WRITE(3,21)
21 FORMAT(1H0,20X,'END INPUT DATA REPORT')
C
RETURN

```



```

C
C
C ERROR CONDITIONS
C
C ERROR 1 - INVALID GATE ID, GATE TYPE, OR DELAY VALUE - FATAL
400   ERRNO=1
      CALL ERROR(ERRNO,'MISSING GATE ID, TYPE, OR DELAY      ')
C ERROR 2 - INVALID GATE TYPE - FATAL
410   ERRNO=2
      CALL ERROR(ERRNO,'INVALID GATE TYPE                    ')
C ERROR 3 - INVALID NODE ID - FATAL
420   ERRNO=3
      CALL ERROR(ERRNO,'INVALID NODE ID OR NODE QTY         ')
C ERROR 4 - CIRCUIT ARRAYS CAPACITY EXCEEDED - FATAL
430   ERRNO=4
      CALL ERROR(ERRNO,'CIRCUIT ARRAYS CAPACITY EXCEEDED    ')
C ERROR 6 - INITIAL CONDITION NODE INVALID - FATAL
450   ERRNO=6
      CALL ERROR(ERRNO,'INITIAL CONDITION NODE INVALID      ')
C ERROR 7 - INITIAL CONDITION LOGIC VALUE INVALID - FATAL
460   ERRNO=7
      CALL ERROR(ERRNO,'INITIAL CONDITION LOGIC VALUE INVALID ')
C ERROR 9 - DYNAMIC NODE ID INVALID - FATAL
480   ERRNO=9
      CALL ERROR(ERRNO,'DYNAMIC NODE ID INVALID              ')
C ERROR 10 - DYNAMIC CHANGE LOGIC VALUE INVALID - FATAL
490   ERRNO=10
      CALL ERROR(ERRNO,'DYNAMIC CHANGE LOGIC VALUE INVALID   ')
C ERROR 11 - DYNAMIC CHANGE NODE NOT INITIALIZED - FATAL
500   ERRNO=11
      CALL ERROR(ERRNO,'DYNAMIC NODE NOT INITIALIZED         ')
C ERROR 13 - PREMATURE END OF INPUT DATA - FATAL
999   ERRNO=13
      CALL ERROR(ERRNO,'PREMATURE END OF INPUT DATA        ')
      END

```

```

C
C
C FIND NODE SUBROUTINE - FNDNOD.
C SCANS NODE LIST SEARCHING FOR SNODE.
C RETURNS K=0 IF NOT FOUND, ELSE K POINTS
C NODE LIST CELL FOUND.

```

```

C
      SUBROUTINE FNDNOD(SNODE,K)
      IMPLICIT INTEGER (A-Z)
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      INLTEPT(200)
      COMMON /AREA1/NLNUM

```

```

C
      K=0
      DO 100 J=1,NLNUM
      IF(SNODE.EQ.NLNODE(J))GO TO 110

```

```

100    CONTINUE
      GO TO 120
110    K=J
120    RETURN
      END

C
C
C CIRCUIT INITIALIZATION ROUTINE - CINIT.
C PROPAGATES INITIAL LOGIC VALUES THRU THE CIRCUIT
C TO PRODUCE INITIAL STATIC CIRCUIT OUTPUTS.
      SUBROUTINE CINIT
      IMPLICIT INTEGER (A-Z)
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      INLTEPT(200)
      COMMON /GLIST/GLGATE(100),GLTYPE(100),GLDLAY(100),
      IGLINPT(100),GLONPT(100)
      COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
      ITENCEL(500),TENCNS(500)
      COMMON /INLIST/INPTR(300),INNGPT(300)
      COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR,RPTSW,RPTIME,
      IDMVTIM,STOPSW,STOPTM,GATEMX,NLMAX,GLMAX,INMAX
      COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIHTYP
      DIMENSION NVAL(6)

C
C TO INITIALIZE CIRCUIT, SCAN GATE LIST AND SIMULATE UNTIL
C ALL GATES HAVE ESTABLISHED AN OUTPUT VALUE.
C
100    FOUND=0
      ADDED=0
      DO 200 GCTR=1,GLNUM
      NPTR=GLONPT(GCTR)
      M=MOD(NLLVAL(NPTR),256)
      IF(M.NE.99)GO TO 200
      FOUND=1
C OUTPUT NODE HAS NO ESTABLISHED VALUE, FIND INPUT NODES.
C UNPACK INPUT NODE POINTER & INPUT NODE QTY. (MODULO 256)
      K=GLINPT(GCTR)
      INLPTR=K / 256
      INLQTY=MOD(K,256)
C CHECK IF INPUT NODES HAVE ESTABLISHED VALUES & LOAD
C VALUES IN NVAL ARRAY.
      DO 120 K=1,INLQTY
      NPTR1=INPTR(INLPTR+(K-1))
      M=MOD(NLLVAL(NPTR1),256)
      IF(M.EQ.99)GO TO 200
      NVAL(K)=M
120    CONTINUE
C ALL INPUTS HAVE A VALUE. COMPUTE GATE FUNCTION.
      CALL GEVAL(GLTYPE(GCTR),NVAL,LVAL)
C UPDATE OUTPUT NODE VALUE. PRESERVE REPORT FLAG.
      NLLVAL(NPTR)=((NLLVAL(NPTR) / 256) * 256) + LVAL

```

```

        ADDED=1
200    CONTINUE
C CHECK IF ANY NEW NODES FOUND & ANY NEW NODES ADDED.
      IF(FOUND.EQ.0)GO TO 250
      IF(ADDED.EQ.0)GO TO 210
C MAKE ANOTHER PASS
      GO TO 100
C
C ERROR 15 - NODE WAS FOUND BUT NO NODES ADDED - FATAL
210    ERRNO=15
      CALL 'ERROR(ERRNO,'INITIALIZATION SIMULATION FAILURE
C
C
250    RETURN
      END
C
C
C GATE EVALUATION ROUTINE - GEVAL.
C EVALUATES THE OUTPUT FUNCTION FOR EACH TYPE OF GATE.
C
      SUBROUTINE GEVAL(GTYPE,NVAL,LVAL)
      IMPLICIT INTEGER (A-Z)
      COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR
      DIMENSION NVAL(6),MAXTBL(4,4),MINTBL(4,4),
      1ANDTBL(4,4),ORTBL(4,4)
C
      DATA MAXTBL/0,1,2,3,1,1,2,3,2,2,2,3,3,3,3,3/
      DATA MINTBL/0,0,0,0,0,1,1,1,0,1,2,2,0,1,2,3/
      DATA ANDTBL/0,0,0,0,0,1,0,1,0,0,2,2,0,1,2,3/
      DATA ORTBL /0,1,2,3,1,1,3,3,2,3,2,3,3,3,3,3/
C
C INCREMENT EACH LOGIC VALUE PASSED. THEY WILL BE IN A RANGE
C FROM 1 - 4 AFTER INCREMENTING TO USE FOR TABLE SUBSCRIPTS.
      DO 110 J=1,6
      NVAL(J)=NVAL(J)+1
110    CONTINUE
      LVAL=0
C BRANCH TO INDIVIDUAL GATE ROUTINES
      GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
      116,17,18,19,20,21,22,23,24,25,26,27,28,29,
      230,31,32,33,34,35,36,37,38,39,
      331,32,33,34,35,36,37,38,39,
      431,32,33,34,35,36,37,38,39,
      540,41,42,40,41,42,40,41,42,43,43,43,44),GTYPE
      GO TO 500
C 2-INPUT MAX GATE
1      LVAL=MAXTBL(NVAL(1),NVAL(2))
      GO TO 100
C 3-INPUT MAX GATE
2      J=MAXTBL(NVAL(1),NVAL(2)) + 1
      LVAL=MAXTBL(J,NVAL(3))
      GO TO 100

```

```

C 4-INPUT MAX GATE
3      J=MAXTBL(NVAL(1),NVAL(2)) + 1
      K=MAXTBL(NVAL(3),NVAL(4)) + 1
      LVAL=MAXTBL(J,K)
      GO TO 100
C 2-INPUT MIN GATE
4      LVAL=MINTBL(NVAL(1),NVAL(2))
      GO TO 100
C 3-INPUT MIN GATE
5      J=MINTBL(NVAL(1),NVAL(2)) + 1
      LVAL=MINTBL(J,NVAL(3))
      GO TO 100
C 4-INPUT MIN GATE
6      J=MINTBL(NVAL(1),NVAL(2)) + 1
      K=MINTBL(NVAL(3),NVAL(4)) + 1
      LVAL=MINTBL(J,K)
      GO TO 100
C PSEUDO-COMPLEMENT GATE
7      LVAL=3 - (NVAL(1) - 1)
      GO TO 100
C 2-INPUT AND GATE
8      LVAL=ANDTBL(NVAL(1),NVAL(2))
      GO TO 100
C 3-INPUT AND GATE
9      J=ANDTBL(NVAL(1),NVAL(2)) + 1
      LVAL=ANDTBL(J,NVAL(3))
      GO TO 100
C 4-INPUT AND GATE
10     J=ANDTBL(NVAL(1),NVAL(2)) + 1
      K=ANDTBL(NVAL(3),NVAL(4)) + 1
      LVAL=ANDTBL(J,K)
      GO TO 100
C 2-INPUT OR GATE
11     LVAL=ORTBL(NVAL(1),NVAL(2))
      GO TO 100
C 3-INPUT OR GATE
12     J=ORTBL(NVAL(1),NVAL(2)) + 1
      LVAL=ORTBL(J,NVAL(3))
      GO TO 100
C 4-INPUT OR GATE
13     J=ORTBL(NVAL(1),NVAL(2)) + 1
      K=ORTBL(NVAL(3),NVAL(4)) + 1
      LVAL=ORTBL(J,K)
      GO TO 100
C CYCLE 1 GATE
14     LVAL=MOD((NVAL(1)-1)+1,4)
      GO TO 100
C CYCLE 2 GATE
15     LVAL=MOD((NVAL(1)-1)+2,4)
      GO TO 100
C CYCLE 3 GATE

```

```

16   LVAL=MOD((NVAL(1)-1)+3,4)
    GO TO 100
C PLUS GATE
17   LVAL=(NVAL(1) - 1) + (NVAL(2) - 1)
    IF(LVAL.GT.3) LVAL=3
    GO TO 100
C 2-INPUT NOT MAX GATE
18   LVAL=3 - (MAXTBL(NVAL(1),NVAL(2)))
    GO TO 100
C 3-INPUT NOT MAX GATE
19   J=MAXTBL(NVAL(1),NVAL(2)) + 1
    LVAL=3 - (MAXTBL(J,NVAL(3)))
    GO TO 100
C 4-INPUT NOT MAX GATE
20   J=MAXTBL(NVAL(1),NVAL(2)) + 1
    K=MAXTBL(NVAL(3),NVAL(4)) + 1
    LVAL=3 - (MAXTBL(J,K))
    GO TO 100
C 2-INPUT NOT MIN GATE
21   LVAL= 3 - (MINTBL(NVAL(1),NVAL(2)))
    GO TO 100
C 3-INPUT NOT MIN GATE
22   J=MINTBL(NVAL(1),NVAL(2)) + 1
    LVAL= 3 - (MINTBL(J,NVAL(3)))
    GO TO 100
C 4-INPUT NOT MIN GATE
23   J=MINTBL(NVAL(1),NVAL(2)) + 1
    K=MINTBL(NVAL(3),NVAL(4)) + 1
    LVAL= 3 - (MINTBL(J,K))
    GO TO 100
C 2-INPUT NAND GATE
24   LVAL= 3 - (ANDTBL(NVAL(1),NVAL(2)))
    GO TO 100
C 3-INPUT NAND GATE
25   J=ANDTBL(NVAL(1),NVAL(2)) + 1
    LVAL= 3 - (ANDTBL(J,NVAL(3)))
    GO TO 100
C 4-INPUT NAND GATE
26   J=ANDTBL(NVAL(1),NVAL(2)) + 1
    K=ANDTBL(NVAL(3),NVAL(4)) + 1
    LVAL= 3 - (ANDTBL(J,K))
    GO TO 100
C 2-INPUT NOR GATE
27   LVAL= 3 - (ORTBL(NVAL(1),NVAL(2)))
    GO TO 100
C 3-INPUT NOR GATE
28   J=ORTBL(NVAL(1),NVAL(2)) + 1
    LVAL= 3 - (ORTBL(J,NVAL(3)))
    GO TO 100
C 4-INPUT NOR GATE
29   J=ORTBL(NVAL(1),NVAL(2)) + 1

```

```

      K=UNIBL(NVAL(3),NVAL(4)) + 1
      LVAL= 3 - (ORTBL(J,K))
      GO TO 100
C NOT PLUS
30    LVAL = (NVAL(1) - 1) + (NVAL(2) - 1)
      IF(LVAL.GT.3) LVAL=3
      LVAL = 3 - LVAL
      GO TO 100

C
C LITERAL GATES - RECALL THAT NVAL WAS INCREMENTED BY 1
C AT START OF SUBROUTINE.
C
C LITERAL 0,0
31    IF(NVAL(1).EQ.1)GO TO 100
      GO TO 200
C LITERAL 1,1
32    IF(NVAL(1).EQ.2)GO TO 100
      GO TO 200
C LITERAL 2,2
33    IF(NVAL(1).EQ.3)GO TO 100
      GO TO 200
C LITERAL 3,3
34    IF(NVAL(1).EQ.4)GO TO 100
      GO TO 200
C LITERAL 0,1
35    IF(NVAL(1).EQ.1 .OR. NVAL(1).EQ.2)GO TO 100
      GO TO 200
C LITERAL 0,2
36    IF(NVAL(1).NE.4)GO TO 100
      GO TO 200
C LITERAL 1,2
37    IF(NVAL(1).EQ.2 .OR. NVAL(1).EQ.3)GO TO 100
      GO TO 200
C LITERAL 1,3
38    IF(NVAL(1).NE.1)GO TO 100
      GO TO 200
C LITERAL 2,3
39    IF(NVAL(1).EQ.3 .OR. NVAL(1).EQ.4)GO TO 100
      GO TO 200

C
C SET LITERAL OUTPUT VALUE. ACCOMPLISHED BY KNOWING THE
C LITERAL'S ARRAY POSITION RELATIVE TO ARRAY POSITION 31.
200   K = (( GTYPE - 31 ) / 9 ) + 2
      LVAL = MINTBL(K,4)
      GO TO 100

C
C 2-INPUT INHIBIT GATE
40    IF(NVAL(1).GT.1 .OR. NVAL(2).GT.1)GO TO 100
      GO TO 300
C 3-INPUT INHIBIT GATE
41    IF(NVAL(1).GT.1 .OR. NVAL(2).GT.1)GO TO 100

```

```

        IF(NVAL(3).GT.1)GO TO 100
        GO TO 300
C 4-INPUT INHIBIT GATE
42     IF(NVAL(1).GT.1 .OR. NVAL(2).GT.1)GO TO 100
        IF(NVAL(3).GT.1 .OR. NVAL(4).GT.1)GO TO 100
        GO TO 300
C SET INIHIRIT GATE OUTPUT VALUE. ACCOMPLISHED BY
C KNOWING THE VALUE OF GTYPE. 1ST INHIBIT IS IN ARRAY
C POSITION 58.
300    K = ((GTYPE -58) / 3) + 2
        LVAL = MINTBL (K,4)
        GO TO 100

C
C UNIVERSAL GATE
43     IF(NVAL(5).GT.1 .OR. NVAL(6).GT.1)GO TO 100
        K = MAXTBL (NVAL(1),NVAL(2))
        J = (NVAL(3)-1) + (NVAL(4)-1)
        IF(J.GT.3) J=3
        LVAL = J + K
        IF(LVAL.GT.3) LVAL=3
C SET UNIVERSAL GATE OUTPUT VALUE
        IF(GTYPE.EQ.67)L=1
        IF(GTYPE.EQ.68)L=2
        IF(GTYPE.EQ.69)L=3
        IF(L.LT.LVAL)GO TO 100
        LVAL = L - LVAL
        GO TO 100

C
C 4-TO-1 MUX GATE
44     IF(NVAL(5).EQ.1)LVAL=NVAL(1)-1
        IF(NVAL(5).EQ.2)LVAL=NVAL(2)-1
        IF(NVAL(5).EQ.3)LVAL=NVAL(3)-1
        IF(NVAL(5).EQ.4) LVAL=NVAL(4) - 1
        GO TO 100

C
C ERROR 16 - INVALID GATE INDEX - FATAL
500    ERRNO=16
        CALL ERROR(ERRNO,'INVALID GATE INDEX      ')

C
C ERROR 22 - GATE OUTPUT INVALID - FATAL
510    ERRNO = 22
        CALL ERROR(ERRNO,'GATE OUTPUT EVALUATION INVALID  ')

100    IF(LVAL.LT.0 .OR. LVAL.GT.3) GO TO 510
C
        RETURN
        END

C
C
C ERROR MESSAGE ROUTINE - ERROR
C IF ERR = 0 A NON-FATAL ERROR HAS OCCURRED: OTHERWISE, THE
C PGM IS ABORTED.

```

```
C      SUBROUTINE ERROR(ERRNO,MSG)
      IMPLICIT INTEGER (A-Z)
      COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR
      DIMENSION MSG(8)

C
      IF(ERR.EQ.0)GO TO 100
      WRITE(5,1)ERRNO,MSG
1     FORMAT(1H0,'FATAL ERROR ',I2,' - ',8A5)
      WRITE(3,1)ERRNO,MSG
      CALL EXIT
100    WRITE(5,2)ERRNO,MSG
2     FORMAT(1X,'ERROR ',I2,' - ',8A5)
      WRITE(3,2)ERRNO,MSG
      RETURN
      END
```



```

C
C TIME-EVENT QUEUE INSERT ROUTINE - TEINST.
C UPDATES THE T-E QUEUE BASED ON THE NODE, TIME & LOGIC
C VALUE PASSED.  QUEUE IS ORDERED BY ASCENDING TIME VALUE.
C
      SUBROUTINE TEINST(NODPTR,ITIME,IVAL)
      IMPLICIT INTEGER (A-Z)
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      1NLTEPT(200)
      COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
      1TENCEL(500),TENCSN(500)
      COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR
      COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIMTYP
C
C GET A FREE T-E LIST CELL.  RETURNED IN J
      CALL @TECEL(J)
      IF(TENUM.EQ.0)GO TO 110
C
      IF(ITIME.LT.TETIME(TELOW))GO TO 120
      H=TELOW
100   IF(TENCEL(H).EQ.0)GO TO 140
      IF(ITIME.LT.TETIME(TENCEL(H)))GO TO 130
      H=TENCEL(H)
      GO TO 100
C INSERT THE FIRST T-E NODE
110   TENCEL(J)=0
      TELOW=J
      TEHIGH=J
      GO TO 150
C NEW TELOW CELL
120   TENCEL(J)=TELOW
      TELOW=J
      GO TO 150
C NORMAL T-E INSERT
130   TENCEL(J)=TENCEL(H)
      TENCEL(H)=J
      GO TO 150
C NEW HIGH CELL
140   TENCEL(J)=0
      TEHIGH=J
      TENCEL(H)=J
C FINISH NEW CELL UPDATE
150   TENUM=TENUM+1
      TETIME(J)=ITIME
      TENPTR(J)=NODPTR
      TELVAL(J)=IVAL
C LINK NLIST TO TELIST.  IF LINK EXISTS ALREADY, UPDATE
C THE LINKS FOR THE NEW T-E ENTRY.
      IF(NLTEPT(NODPTR).EQ.-1)GO TO 200

```

```

TENCSN(J)=NLTEPT(NODPTR)
NLTEPT(NODPTR)=J
GO TO 500
200 NLTEPT(NODPTR)=J
TENCSN(J)=0
C
500 RETURN
END
C
C
C GET TIME-EVENT CELL ROUTINE - GTECEL
C SECURES THE NEXT T-E FREE CELL, IF AVAILABLE,
C AND RETURNS CELL POINTER IN J.
C
SUBROUTINE GTECEL(J)
IMPLICIT INTEGER (A-Z)
COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
1TENCEL(500),TENCSN(500)
COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR
COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIMTYP
C
IF(TEFREE.EQ.0)GO TO 100
IF(TETIME(TEFREE).NE.-1)GO TO 110
J=TEFREE
TEFREE=TENCEL(J)
RETURN
C
C ERROR 8 - TIME-EVENT QUEUE OVERFLOW - FATAL
100 ERRNO=8
CALL ERROR(ERRNO,'TIME-EVENT QUEUE OVERFLOW')
C ERROR 17 - TIME-EVENT QUEUE STRUCTURE PROBLEM - FATAL
110 ERRNO=17
CALL ERROR(ERRNO,'TIME-EVENT QUEUE STRUCTURE PROBLEM')
END
C
C
C TIME-EVENT QUEUE RETURN CELL ROUTINE - RTECEL.
C RETURNS CELL J TO THE T-E FREE LIST.
C
SUBROUTINE RTECEL(J)
IMPLICIT INTEGER (A-Z)
COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
1TENCEL(500),TENCSN(500)
COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIMTYP
C
IF(J.EQ.0)GO TO 100
TENCEL(J)=TEFREE
TETIME(J)=-1
TEFREE=J
100 RETURN
END

```

```

C
C SIMULATE SUBROUTINE - SIMUL.
C PERFORMS THE SIMULATION PROCESS, PRINTS RESULTS, ETC.
C
      SUBROUTINE SIMUL(FILNAM,FRMAT)
      IMPLICIT INTEGER (A-Z)
      REAL*8 FILNAM
      COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
      1NLTEPT(200)
      COMMON /GLIST/GLGATE(100),GLTYPE(100),GLDLAY(100),
      1GLINPT(100),GLONPT(100)
      COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
      1TENCEL(500),TENCNS(500)
      COMMON /INLIST/INPTR(300),INNGPT(300)
      COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR,RPTSW,RPTIME,
      1DMVTIM,STOPSW,STOPTM,GATEMX,NLMAX,GLMAX,INMAX
      COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIMTYP
      DIMENSION CHGLST(200),NVAL(6)

C
C INITIALIZE SIMULATION VARIABLES
      RPCNT=0
      LINE=99
      ERR=1
      TCNT=0

C
C EXAMINE TIME-EVENT QUEUE TO FIND ALL NODES TO BE UPDATED AT
C THE CURRENT LOWEST TIME (TETIME(TELOW)). CONSTRUCT A LIST
C OF THESE NODES IN CHGLST WHICH CONSISTS OF POINTERS TO THE
C SELECTED T-E QUEUE CELL(S). IF MORE THAN ONE ENTRY EXISTS
C FOR THE SAME NODE, THE NEAREST LOGIC VALUE TO THE CURRENT
C NODE VALUE IS SELECTED.
C
100      IF(TENUM.EQ.0)GO TO 460
          TIME=TETIME(TELOW)
          TEPTR=TELOW
          CHGNUM=0
110      IF(TETIME(TEPTR).GT.TIME)GO TO 200
          IF(TETIME(TEPTR).LT.TIME)GO TO 470
          IF(CHGNUM.EQ.0)GO TO 130
C CHECK IF NODE ALREADY IN CHGLST
          DO 120 J=1,CHGNUM
              K=TENPTR(CHGLST(J))
              IF(K.EQ.TENPTR(TEPTR))GO TO 140
120      CONTINUE
C NOT FOUND IN CHGLST - ADD IT
130      CHGNUM=CHGNUM+1
          CHGLST(CHGNUM)=TEPTR
C SET DELETE FLAG ON T-E CELL
135      TELVAL(TEPTR)=TELVAL(TEPTR) + 256
          TEPTR=TENCEL(TEPTR)
C CHECK END-OF-LIST
      -----

```

```

      IF(IEPIK.EQ.0) GO TO 200
      GO TO 110
C
C T-E NODE WAS FOUND IN CHGLST. DETERMINE IF CHGLST
C ENTRY SHOULD BE SUPERCEDED. K POINTS TO NODE.
140   DVAL1=ABS(NLLVAL(K) - TELVAL(CHGLST(J)))
      DVAL2=ABS(NLLVAL(TENPTR(TEPTR)) - TELVAL(TEPTR))
      IF(DVAL1.LE.DVAL2)GO TO 135
C SUPERCEED CURRENT CHGLST VALUE.
      CHGLST(J)=TEPTR
      GO TO 135
C
C SIMULATE THE NODE CHANGES THAT WERE LOADED INTO CHGLST.
C TO EVALUATE THE AFFECTED GATES, THE CHGLST IS CONSULTED
C FOR ALL INPUT NODES TO SEE IF A SIMULTANEOUS LOGIC CHANGE
C IS PENDING. IF SO, THE NEW VALUE IS USED. IF NOT FOUND
C IN CHGLST, THEN THE CURRENT NLLVAL VALUE IS USED.
C
200   DO 400 J=1,CHGNUM
      PRTSW = 0
      NXTGTE=0
      NPTR=TEMPTR(CHGLST(J))
C UPDATE NODE VALUE. PRESERVE REPORT FLAG.
      OLDVAL=MOD(NLLVAL(NPTR),256)
      NLLVAL(NPTR)={(NLLVAL(NPTR)/256)*256} + MOD(TELVAL(CHGLST(J)),256)
C IF THIS NODE IS REPORTABLE, CHECK FOR TIMING DIAGRAM OUTPUT.
      RFLAG = NLLVAL(NPTR) / 256
      IF(RFLAG.EQ.0) GO TO 205
      IF(FRMAT.EQ.0) GO TO 205
      NEWVAL = MOD(NLLVAL(NPTR),256)
      WRITE(21,8)TIME,NLNODE(NPTR),NEWVAL
8     FORMAT(I7,A5,I1)
C SET GPTR TO FIRST GATE ON NODE.
205   GPTR=NLIGPT(NPTR)
C IS THIS AN OUTPUT-ONLY NODE? IF SO, GPTR WILL BE ZERO.
      IF(GPTR.EQ.0)GO TO 250
C GET INLIST POINTER & QTY
210   NLIST=GLINPT(GPTR) / 256
      NQTY=MOD(GLINPT(GPTR),256)
C M COUNTS THE INPUT NODE QTY. AT END OF LOOP NXTGTE WILL
C POINT TO NEXT GATE ON THIS NODE.
      M=0
      NSTOP=NLIST + NQTY - 1
      DO 240 K=NLIST,NSTOP
C CHECK IF THIS INPUT LIST NODE IS EQUAL TO THE CURRENT NODE BEING
C UPDATED. IF SO, THEN SET NXTGTE.
      IF(INPTR(K).NE.TENPTR(CHGLST(J)))GO TO 220
      NXTGTE=INNGPT(K)
220   M=M+1
C SEE IF THIS INPUT LIST NODE IS PENDING UPDATE AT THIS TIME.
C ARE THE POINTERS EQUAL?

```

```

      DO 230 N=1,CHGNUM
      IF(INPTR(K).NE.TENPTR(CHGLST(N)))GO TO 230
      NVAL(M)=MOD(TELVAL(CHGLST(N)),256)
      GO TO 240
230   CONTINUE
C NO UPDATE FOR INPUT NODE PENDING - SELECT CURRENT LOGIC VALUE.
      NVAL(M)=MOD(NLLVAL(INPTR(K)),256)
240   CONTINUE
C
C EVALUATE GATE FUNCTION.
      CALL GEVAL(GLTYPE(GPTR),NVAL,LVAL)
C REPORT CHECK
250   IF(RPTSW.EQ.2)GO TO 315
C CHECK IF RESULTS HAVE BEEN PRINTED. IF SO, PRTSW = 1.
      IF(PRTSW.EQ.1) GO TO 315
      PRTSW = 1
      RFLAG=NLLVAL(NPTR) / 256
      IF(RFLAG.EQ.0)GO TO 315
      IF(LINE.LE.50)GO TO 260
      WRITE(3,1)FILNAM
1     FORMAT(1H1,22X,'QUATERNARY LOGIC SIMULATOR - QLOSIM',//,
           15X,A10,' SIMULATION RESULTS:',/)
      WRITE(3,2)
2     FORMAT(15X,'TIME   NODE   PREV. VALUE   NEW VALUE')
      LINE=0
260   LINE=LINE+1
      NEWVAL=MOD(NLLVAL(NPTR),256)
      WRITE(3,3)TIME,HLNODE(NPTR),OLDVAL,NEWVAL
3     FORMAT(14X,I5,3X,A5,8X,I1,12X,I1)
C
C SCHEDULE THE T-E ENTRIES FOR THE GATE OUTPUT
C NODE AT TIME = (TIME + GLDLAY).
C CHECK AGAIN IF THIS IS AN OUTPUT ONLY NODE.
315   IF(GPTR.EQ.0)GO TO 400
      DNPTR=GLONPT(GPTR)
      TEPTR=NLTEPT(DNPTR)
      IF(TEPTR.EQ.-1)GO TO 330
C FOR THE OUTPUT NODE, CHECK FOR T-E ENTRIES WITH A TIME
C >= (CURRENT TIME + GLDLAY). UNSCHEDULE THESE ENTRIES BY SETTING
C THEIR DELETE FLAG. ALSO FIND THE LOGIC VALUE FOR THE T-E
C ENTRY WITH THE LARGEST TIME < CURRENT TIME, IF ANY.
C VARIABLE HIT DETERMINES IF ANY VALID (UNDELETED)
C ENTRIES ARE FOUND. IF SO, HIT IS > 0.
      HIT = 0
      LTIME = 0
      TEPTR1=TEPTR
      ITIME = TIME + GLDLAY(GPTR)
      N = TEPTR
      GO TO 323
320   N=TENCSN(TEPTR)
      IF(N.EQ.0)GO TO 340

```

```

323   HIT = HIT + 1
      IF(TETIME(N).LT.ITIME) GO TO 321
      TELVAL(N) = TELVAL(N) + 256
      HIT = HIT - 1
      GO TO 325
321   IF(TETIME(N).LE.LTIME) GO TO 325
      LTIME=TETIME(N)
      TEPTR1=N
325   TEPTR=N
      GO TO 320
C NO T-E ENTRIES EXIST FOR NODE - USE CURRENT LOGIC VALUE.
330   KVAL=MOD(NLLVAL(ONPTR),256)
      GO TO 345
C USE T-E PENDING LOGIC VALUE FOR NODE VALUE, IF HIT > 0.
340   IF(HIT.LE.0) GO TO 330
      IF(LTIME.EQ.0) GO TO 330
      KVAL=MOD(TELVAL(TEPTR1),256)
C COMPUTE NUMBER OF LOGIC LEVEL CHANGES.
345   MVAL=ABS(KVAL-LVAL)
      IF(MVAL.EQ.0)GO TO 380
      IF(MVAL.GT.4)GO TO 480
      ITIME=TIME + GLDLAY(GPTR)
      DO 370 M=1,MVAL
C DETERMINE DIRECTION OF CHANGE.
      IF(KVAL.GT.LVAL)GO TO 350
      JVAL=KVAL + M
      GO TO 360
350   JVAL=KVAL - M
360   CALL TEINST(ONPTR,ITIME,JVAL)
      ITIME=ITIME + DMVTIM
370   CONTINUE
C
C CHECK NEXT GATE TO BE SIMULATED
380   IF(NXTGTE.EQ.0)GO TO 400
      GPTR=NXTGTE
      GO TO 210
400   CONTINUE
C
C DELETE T-E CELLS THAT WERE FLAGGED
      CALL TEPURG
C
      IF(RPTSW.EQ.2)GO TO 270
C
C CHECK FOR STOP TIME.
425   IF(STOPSW.EQ.1)GO TO 450
      IF(TIME.GE.STOPTM)GO TO 460
C
C CHECK EACH 5000 TIME UNITS TO INSURE NO CIRCUIT
C OSCILLATION EXISTS.
450   TNUM=TIME / 5000
      IF(TNUM.LE.TCNT)GO TO 100

```

```

        TCNT=TNUM
C ASK FOR CONFIRMATION TO CONTINUE SIMULATION.
        WRITE(5,6)
4        FORMAT(5X,'5000 TIME UNITS HAVE ELAPSED - ENTER 0 TO CONTINUE',/)
        READ(5,*)CONTSW
        IF(CONTSW.EQ.0)GO TO 100
460      WRITE(3,7)TIME
7        FORMAT(1H0,15X,'SIMULATION TERMINATED AT TIME ',I7)
        GO TO 500

C
C REPORT TO BE MADE AT TIME INTERVALS - CHECK THE TIME.
270      NRPCNT=TIME / RPTIME
        IF(NRPCNT.LE.RPCNT)GO TO 425
        RPCNT=NRPCNT
        IF(LINE.LE.50)GO TO 280
        LINE=0
        WRITE(3,1)FILNAM
        WRITE(3,4)TIME
4        FORMAT(1H0,10X,'LOGIC VALUES AT TIME ',I7,/,
        115X,'NODE    VALUE',/)
        GO TO 290
280      WRITE(3,4)TIME
290      DO 310 N=1,NLNUM
        RFLAG=NLLVAL(N) / 256
        IF(RFLAG.EQ.0)GO TO 310
        IF(LINE.LE.50)GO TO 300
        LINE=0
        WRITE(3,1)FILNAM
        WRITE(3,4)TIME
300      LINE=LINE + 1
        NEWVAL=MOD(NLLVAL(N),256)
        WRITE(3,5)NLNODE(N),NEWVAL
5        FORMAT(15X,A5,5X,I1)
310      CONTINUE
        GO TO 425

C
C ERROR 18 - TIME-EVENT QUEUE OUT OF SEQUENCE - FATAL
470      ERRNO=18
        CALL ERROR(ERRNO,'TIME-EVENT QUEUE OUT OF SEQUENCE    ')

C
C ERROR 21 - LOGIC CHANGE GENERATED > 4 - FATAL
480      ERRNO=21
        CALL ERROR(ERRNO,'LOGIC CHANGE GENERATED > 4    ')

C
500      RETURN
        END

C
C DELETE FLAGGED TIME-EVENT QUEUE CELLS - TEPURG
C
        SUBROUTINE TEPURG
        IMPLICIT INTEGER (A-Z)

```

```

COMMON /NLIST/NLNODE(200),NLIGPT(200),NLLVAL(200),
1NLTEPT(200)
COMMON /GLIST/GLGATE(100),GLTYPE(100),GLDLAY(100),
1GLINPT(100),GLONPT(100)
COMMON /TELIST/TETIME(500),TENPTR(500),TELVAL(500),
1TENCEL(500),TENCNS(500)
COMMON /INLIST/INPTR(300),INNGPT(300)
COMMON /AREA1/NLNUM,GLNUM,INNUM,ERR,RPTSW,RPTIME,
1DMVTIM,STOPSW,STOPTM,GATENX,NLMAX,GLMAX,INMAX
COMMON /AREA2/TENUM,TELOW,TEMAX,TEHIGH,TEFREE,TIMTYF

C
TEPTR=TELOW
PRVCEL = TEPTR
100 DFLAG=TELVAL(TEPTR) / 256
IF(DFLAG.EQ.0)GO TO 250
C T-E CELL MARKED FOR DELETION - SECURE ALL LINKS IN NLIST
C & T-E QUEUE LISTS.
NPTR=TENPTR(TEPTR)
C ALTER NLIST T-E POINTER VALUE.
IF(NLTEPT(NPTR).EQ.-1)GO TO 400
IF(NLTEPT(NPTR).EQ.TEPTR)GO TO 150
C NLIST T-E POINTER POINTS ANOTHER T-E CELL. SCAN T-E LIST
C TO FIND CORRECT CELL.
TECELL=NLTEPT(NPTR)
TENEXT=TENCNS(TECELL)
130 IF(TENEXT.EQ.TEPTR)GO TO 140
IF(TENEXT.EQ.0)GO TO 400
TECELL=TENEXT
TENEXT=TENCNS(TENEXT)
GO TO 130
C CELL WAS FOUND - UPDATE TENCNS LINKS
140 TENCNS(TECELL)=TENCNS(TENEXT)
GO TO 160
C NLTEPT POINTED TO T-E CELL BEING DELETED.
150 NLTEPT(NPTR) = TENCNS(TEPTR)
IF(TENCNS(TEPTR).EQ.0) NLTEPT(NPTR) = -1
C DELETE & RETURN THE T-E CELL TO THE FREE POOL.
160 IF(TELOW.EQ.TEPTR) GO TO 170
C UPDATE PREVIOUS T-E CELL'S NEXT CEL POINTER
TENCEL(PRVCEL) = TENCEL(TEPTR)
GO TO 180
C SET NEW TELOW VALUE
170 TELOW = TENCEL(TEPTR)
180 TELVAL(TEPTR)=0
C SAVE CURRENT VALUE OF TENCEL(TEPTR); IT WILL BE ALTERED BY
C RTECEL. NXTCEL HOLDS THE VALUE.
NXTCEL=TENCEL(TEPTR)
CALL RTECEL(TEPTR)
TENUM=TENUM - 1

C
C CHECK FOR NEXT CELL OR END-OF-LIST.

```



```

200   IF(NXTCEL.EQ.0)GO TO 500
      TEPTR=NXTCEL
      GO TO 100
C
250   IF(TENCEL(TEPTR).EQ.0)GO TO 500
      PRVCEL = TEPTR
      TEPTR=TENCEL(TEPTR)
      GO TO 100
C
C ERROR 19 - LINK ERROR IN NODE & TIME-EVENT LISTS - FATAL
400   ERRNO=19
      CALL ERROR(ERRNO,'LINK ERROR BETWEEN NODE & T-E LISTS  ')
C
500   RETURN
      END
C
C TIMING DIAGRAM GENERATOR - TIMGEN.
C GENERATES A FOUR-LEVEL TIMING DIAGRAM FOR
C CIRCUIT NODES SELECTED FOR REPORTING.
C 80 OR 132 COLUMN, PAGED OR CONTINUOUS FORMAT
C SELECTED BY VARIABLE FRMAT.
C
      SUBROUTINE TIMGEN(FILNAM,FRMAT)
      IMPLICIT INTEGER (A-Z)
      REAL*8 FILNAM
      REAL*8 HEAD1(10),HEAD2(10)
C
      DIMENSION RPTLST(100),LVAL(10),CVAL(10),PVAL(2,10)
C
      DATA HEAD1/10 * '0 1' //
      DATA HEAD2/10 * ' 2 3' //
C OPEN TIMING DATA FILE
      OPEN(UNIT=21,DEVICE='DSK',ACCESS='SEQIN',FILE='TIMING.DAT')
C
C INITIALIZE VARIABLES & LISTS
      HDONE=0
      PTIME=0
      LQTY=0
      LSTQTY=0
      RESTRT=0
      LINE=99
      GROUP=0
      HIT=0
      DO 100 J=1,100
      RPTLST(J) = ' '
100   CONTINUE
102   DO 101 J=1,10
      LVAL(J) = 0
      CVAL(J) = 0
101   CONTINUE
      IF(FRMAT.LT.3) GO TO 105

```

```

      IF(FRMAT.EQ.4) LINE = -1
      MODQTY=10
      HEAD = (GROUP * 10) + 1
      TAIL = HEAD + 9
      IF(LINE.NE.-1)GO TO 110
      WRITE(3,5)FILNAM
5      FORMAT(1H1,22X,'QUATERNARY LOGIC SIMULATOR - QLOSIM',//,
      117X,'TIMING DIAGRAM',16X,'FILE = ',A10,/)
6      FORMAT(4X,10(7X,A5))
16     FORMAT(3X,'TIME ',10(2A5,2X))
      GO TO 110
105    MODQTY=6
      IF(FRMAT.EQ.2) LINE = -1
      HEAD = (GROUP * 6) + 1
      TAIL = HEAD + 5
      IF(LINE.NE.-1)GO TO 110
      WRITE(3,5)FILNAM
7      FORMAT(4X,6(7X,A5))
17     FORMAT(3X,'TIME ',6(2A5,2X))
C
110    EOF = 0
      READ(21,1,END=500)TIME,NODE,NLVAL
1      FORMAT(17,A5,I1)
      TTIME = TIME
      GO TO 115
111    READ(21,1,END=500)TIME,NODE,NLVAL
C CHECK FOR NODE IN LIST BEING REPORTED
115    DO 120 J=HEAD,TAIL
      IF(NODE.EQ.RPTLST(J)) GO TO 200
120    CONTINUE
C NOT FOUND - CHECK IF THIS IS FIRST PASS THRU FILE.
      IF(GROUP.EQ.0) GO TO 125
      GO TO 111
C CHECK IF NODE IS IN RPTLST. IF NOT, ADD IT.
125    IF(LSTQTY.LT.TAIL) GO TO 145
      K = TAIL + 1
      DO 130 M=K,LSTQTY
      IF(NODE.EQ.RPTLST(M)) GO TO 111
130    CONTINUE
C NOT FOUND IN RPTLST
140    LSTQTY = LSTQTY + 1
      RPTLST(LSTQTY) = NODE
      GO TO 111
145    LQTY = LQTY + 1
      LSTQTY = LSTQTY + 1
      RPTLST(LSTQTY) = NODE
      LVAL(LQTY) = NLVAL
      GO TO 111
C
C SAVE LOGIC CHANGE DIRECTION IN CVAL LIST.
C IF THIS IS AN INITIALIZATION RECORD (TIME = 0),

```

```

C UPDATE LVAL LIST ONLY.
200   SAVEJ = J - 1
      IF(TIME.GT.TTIME) GO TO 300
205   HIT = 1
      M = MOD(SAVEJ,MODQTY) + 1
      IF(TIME.NE.0) GO TO 210
      LVAL(M) = NLVAL
      GO TO 111
210   CVAL(M) = NLVAL - LVAL(M)
      GO TO 111

C
C REPORT NODE VALUES.
C CHECK TIME DIFFERENTIAL. IF > 10, RESTART TIME.
300   PNUM = (HEAD + LQTY) - 1
      DTIME = TTIME - PTIME
      IF(DTIME.LE.10) GO TO 330
C TERMINATE PREVIOUS TIME & RESTART AT (TIME - 1).
      DO 310 N=1,LQTY
      PVAL(1,N) = ' '
      PVAL(2,N) = ' '
      LVAL1 = LVAL(N) + 1
      GO TO (311,312,313,314),LVAL1
311   PVAL(1,N) = '!'
      GO TO 310
312   PVAL(1,N) = ' ! '
      GO TO 310
313   PVAL(2,N) = ' ! '
      GO TO 310
314   PVAL(2,N) = ' ! '
310   CONTINUE
      PTIME = PTIME + 1
      IF(LINE.GE.0) GO TO 320
      IF(HDONE.EQ.1) GO TO 325
      HDONE = 1
321   IF(FRMAT.LT.3) GO TO 322
      WRITE(3,6)(RPTLST(N),N=HEAD,PNUM)
      WRITE(3,16)(HEAD1(N),HEAD2(N),N=1,LQTY)
      GO TO 325
322   WRITE(3,7)(RPTLST(N),N=HEAD,PNUM)
      WRITE(3,17)(HEAD1(N),HEAD2(N),N=1,LQTY)
      GO TO 325
320   IF(LINE.LE.50)GO TO 325
      LINE = 0
      WRITE(3,5)FILNAM
      GO TO 321
325   WRITE(3,10)PTIME,((PVAL(I,J),I=1,2),J=1,LQTY)
10    FORMAT(1X,I6,1X,10(2A5,2X))
      WRITE(3,11)
11    FORMAT(1H0,7X,'<----- NO CHANGE ----->',/)
      IF(LINE.GE.0) LINE = LINE + 4
      RESTRT = 1

```

```

                GO TO 360
C
C BRING DIAGRAM UP TO CURRENT TIME
330  N = TTIME - PTIME
      IF(N.LE.1) GO TO 360
      DO 340 N = 1,LQTY
        PVAL(1,N) = '      '
        PVAL(2,N) = '      '
        LVAL1 = LVAL(N) + 1
        GO TO (331,332,333,334),LVAL1
331  PVAL(1,N) = '!'
      GO TO 340
332  PVAL(1,N) = ' ! '
      GO TO 340
333  PVAL(2,N) = ' ! '
      GO TO 340
334  PVAL(2,N) = ' !'
340  CONTINUE
      PTIME = PTIME + 1
      MTIME = MOD(PTIME,5)
      IF(LINE.GE.0) GO TO 344
      IF(HDONE.EQ.1) GO TO 345
      HDONE = 1
341  IF(FRMT.LT.3) GO TO 342
      WRITE(3,6)(RPTLST(N),N=HEAD,PNUM)
      WRITE(3,16)(HEAD1(N),HEAD2(N),N=1,LQTY)
      GO TO 345
342  WRITE(3,7)(RPTLST(N),N=HEAD,PNUM)
      WRITE(3,17)(HEAD1(N),HEAD2(N),N=1,LQTY)
      GO TO 345
344  IF(LINE.LE.50) GO TO 345
      LINE = 0
      WRITE(3,5)FILNAM
      GO TO 341
345  IF(MTIME.NE.0) GO TO 350
      WRITE(3,10)PTIME,((PVAL(I,J),I=1,2),J=1,LQTY)
      GO TO 351
350  WRITE(3,12)((PVAL(I,J),I=1,2),J=1,LQTY)
12   FORMAT(8X,10(2A5,2X))
351  IF(LINE.GE.0) LINE = LINE + 1
      GO TO 330
C
C SET-UP LOGIC CHANGE AND PRINT IT
360  DO 381 N=1,LQTY
      PVAL(1,N) = '      '
      PVAL(2,N) = '      '
      LVAL1 = LVAL(N) + 1
      GO TO (361,363,367,371),LVAL1
361  IF(CVAL(N).EQ.0) GO TO 362
      PVAL(1,N) = '!__.'
      GO TO 380

```

```

362     PVAL(1,N) = '!'
        GO TO 380
363     IF(CVAL(N))364,365,366
364     PVAL(1,N) = '._!'
        GO TO 380
365     PVAL(1,N) = '!'
        GO TO 380
366     PVAL(1,N) = '!'
        PVAL(2,N) = '._'
        GO TO 380
367     IF(CVAL(N))368,369,370
368     PVAL(1,N) = '._'
        PVAL(2,N) = '!'
        GO TO 380
369     PVAL(2,N) = '!'
        GO TO 380
370     PVAL(2,N) = '!_'
        GO TO 380
371     IF(CVAL(N).EQ.0) GO TO 372
        PVAL(2,N) = '._!'
        GO TO 380
372     PVAL(2,N) = '!'
380     LVAL(N) = LVAL(N) + CVAL(N)
        CVAL(N) = 0
381     CONTINUE
C
        IF(LINE.GE.0) GO TO 394
        IF(HDONE.EQ.1) GO TO 395
        HDONE = 1
391     IF(FRMAT.LT.3) GO TO 392
        WRITE(3,6)(RPTLST(N),N=HEAD,PNUM)
        WRITE(3,16)(HEAD1(N),HEAD2(N),N=1,LQTY)
        GO TO 395
392     WRITE(3,7)(RPTLST(N),N=HEAD,PNUM)
        WRITE(3,17)(HEAD1(N),HEAD2(N),N=1,LQTY)
        GO TO 395
394     IF(LINE.LE.50) GO TO 395
        LINE = 0
        WRITE(3,5) FILNAM
        RESTRT = 1
        GO TO 391
395     IF(RESTRT.EQ.1) GO TO 395
        MTIME = MOD(TTIME,5)
        IF(MTIME.NE.0) GO TO 390
385     WRITE(3,10)TTIME,((PVAL(I,J),I=1,2),J=1,LQTY)
        GO TO 400
390     WRITE(3,12)((PVAL(I,J),I=1,2),J=1,LQTY)
400     PTIME = TTIME
        TTIME = TIME
        RESTRT = 0
        IF(LINE.GE.0) LINE = LINE + 1

```

```
        IF(EOF.EQ.0) GO TO 205
        GO TO 505
C
C END-OF-FILE. CHECK FOR LAST PRINT-OUT REQUIRED,
C THEN UPDATE GROUP, ETC., & REWIND FILE.
500     EOF = 1
        IF(HIT.NE.0) GO TO 300
505     GROUP = GROUP + 1
        IF(FRMAT.LT.3) GO TO 510
        LQTY = LSTQTY - (GROUP * 10)
        IF(LQTY.GT.10) LQTY = 10
        GO TO 520
510     LQTY = LSTQTY - (GROUP * 6)
        IF(LQTY.GT.6) LQTY = 6
520     IF(LQTY.LE.0) GO TO 900
        HDONE = 0
        PTIME = 0
        RESTRT = 0
        LINE = 99
        REWIND 21
        GO TO 102
C
900     CLOSE(UNIT=21,DISPOSE='DELETE')
        RETURN
        END
```