AN ABSTRACT OF THE THESIS OF

KAMTHORN THONGURAI (Name) for the Master of Science (Degree)

in Electrical Engineering (Major) presented on May 3, 1968 (Date)

Title: A DIGITAL NETWORK SIMULATOR USING FORTRAN

Abstract approved: [redacted]
(Professor Louis N. Stone)

Methods for mathematically modeling digital networks and for constructing a digital network simulator are presented in this paper. Two simulator programs have been written in FORTRAN for the CDC 3300 Computer system: one for analyzing the behavior of a typical combinational network, and one for analyzing the behavior of a typical sequential network.

A Digital Network Simulator Using FORTRAN

by

Kamthorn Thongurai

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1968

APPROVED:

Redacted for Privacy

Professor of Electrical Engineering

In Charge of Major

Redacted for Privacy

Head of Department of Electrical Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented May 3, 1968

Typed by Carol Baker for Kamthorn Thongurai

ACKNOWLEDGMENTS

The author wishes to express his most sincere appreciation to Professor Louis N. Stone for his valuable advice and help in the preparation of this paper.

He would like to thank Professor John F. Engle for his suggestions and help in using the computer.

Thanks are also due to Professor Robert A. Short for critically reading the manuscript.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

A DIGITAL NETWORK SIMULATOR USING FORTRAN

I. INTRODUCTION

The Present State of the Art of Digital Simulation

Computer simulation has come into increasingly widespread use to study the behavior of digital systems. Alternatives to the use of simulation are mathematical analysis, experimentation with either the actual system or a model of the actual system, or reliance upon experience and intuition. All, including simulation, have limitations. The mathematical analysis of complex systems is very often impossible; experimentation is costly and time consuming, and relevant variables are not always subject to control. Intuition and experience are often the only alternatives to computer simulation available, and they can be very inadequate.

Simulation problems usually involve many variables, many parameters, functions which are not well behaved mathematically and random variables. Thus, much effort is now devoted to "computer simulation" because it is a technique that gives answers in spite of its difficulties, costs and time required.

Simulation models are classified into two major types as follows (18):

Continuous-change Models

Continuous-change models are appropriate when the analyst considers the system he is studying as consisting of a continuous flow of information or material rather than as individual items. These models are usually presented mathematically by differential or difference equations that describe the rate of change of the variables over an interval of time. Such models have long been used in the physical sciences and engineering. They have also been applied to economics and social sciences. They have been applied in operations research problems. Continuous-change models can be simulated on digital computers by using finite-difference equations which, in the limit, approach the differential equations of continuous flow.

The method for the solution of "continuous flow" problems is the use of hybrid analog-digital computers (16). There are programming languages developed for this application (8, 18, 19).

Discrete-change Models

In discrete-change models, the changes in the state of the system are conceptualized as discrete rather than continuous. Systems are idealized as network flow systems and are characterized by the following:

1) the system contains "components" or "elements" or "subsystems" each of which perform definite and prescribed functions;

2) items flow through the system, from one component to another, requiring the performance of a function at a component before the item can move on to the next component;

3) components have finite capacity to process the items and, therefore, items may have to wait before reaching a particular component.

Examples of problems which have been formulated and studied as discrete-change models are job shops, communication networks, logistics systems, and traffic systems, etc. Special packages have been prepared for certain specific applications (2, 6, 7, 8, 9, 18, 19).

If special simulators are not available or if the user decides not to use them, he can use general purpose languages such as FORTRAN, ALGOL, and PL/I (18). The application of such general purpose languages constitutes the work reported in this paper.

At the present time as a result of the increasing number of engineers acquiring digital computer programming skills, analytical

experimentation is emerging as a new tool in the support of design and engineering activities. Simulation is an important form of analytical experimentation and digital engineering. One of the important simulation problems is the simulation of digital networks on a digital computer. This kind of simulation problem is most readily treated as a discrete-change model, since the discrete-change model provides the most effective means of analyzing time-quantized logical behavior.

In such an analysis the simulator has to be properly prepared for the simulation problem. In this context, the word "simulator" is defined as "a program which represents a mathematical model or simulation of some physical system (including another computer" (15)). Specifically, the system simulation which is developed in this paper is characterized by the use of a logical and numerical model rather than by a physical model of the real system.

There are many computing systems specifically developed for simulation of digital networks or system (1, 10, 11, 12, 17). In most cases (except (11, 17)), the simulator was constructed for a particular problem at hand. The development of such special systems has been costly for digital simulation demands. The engineer who may wish to construct a simulator for his digital simulation problem, yet is without the benefits of specialized computing systems, may use the method presented in this paper. The simulation technique presented here requires only the additional knowledge of FORTRAN language.

In this paper the construction of a digital network simulator i.e., the task of preparing a computer program which represents the behavior of real networks (5), will utilize FORTRAN (CDC 3300 version (4)) as the source language.

II. BASIC MODEL OF DIGITAL NETWORKS

There are two classes of digital networks, namely, combinational and sequential. The basic models of both classes can be described as follows:

Combinational Networks

Combinational networks (13, 20) involve one internal state. There can be any number of input and output states.

The generalized schematic of combinational networks is shown in Figure 1.

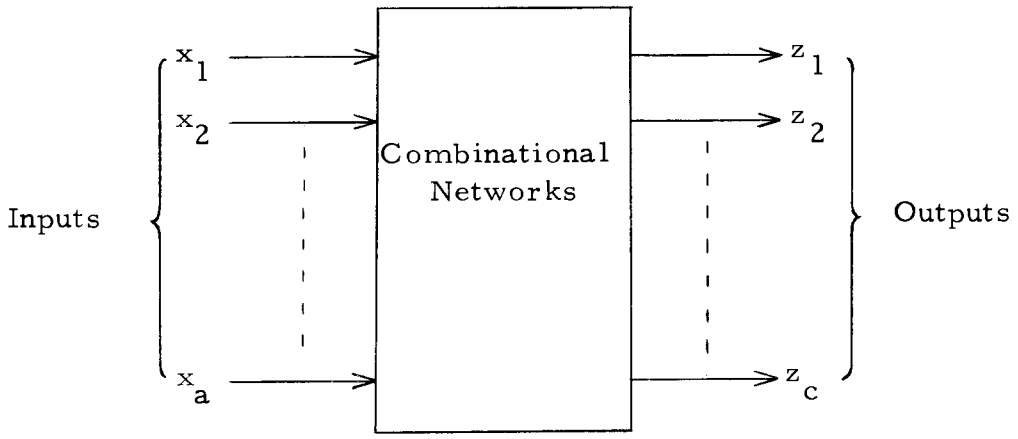


Figure 1. Generalized schematic of a combinational network.

The x variables and z variables are sets of inputs and outputs respectively.

The output can be expressed as c Boolean functions of the input variables $x_1, x_2, \cdots, x_a$. That is, by the set of equations,

$$z_1 = z_1(x_1, x_2, \cdots, x_a)$$

$$z_2 = z_2(x_1, x_2, \cdots, x_a)$$

$$\vdots \qquad \vdots$$

$$z_c = z_c(x_1, x_2, \cdots, x_a)$$

This means that for each input combination, a unique output combination of the z_i's can be obtained. The above set of equations may be expressed in an equivalent truth table form, using 2^a rows for input combinations and $a+c$ columns: one column for each input and one column for each output.

Note that the above description is a behavioral description, that is, it describes only a relation between the inputs and outputs and does not describe any particular internal structure of the combinational networks. The possible internal structures of combinational networks using decision elements can be summarized as combinational multiple output networks "without feedback."

Sequential Networks

Sequential networks (14, 20) can be considered as combinational networks to which b delay circuits of delay Δ have been added. Time delay involves the ability to retain binary information over a certain period of time. A delay element is a memory element, since information can be fed into a delay element and extracted from it at a later time.

The generalized schematic of sequential networks is shown in Figure 2.

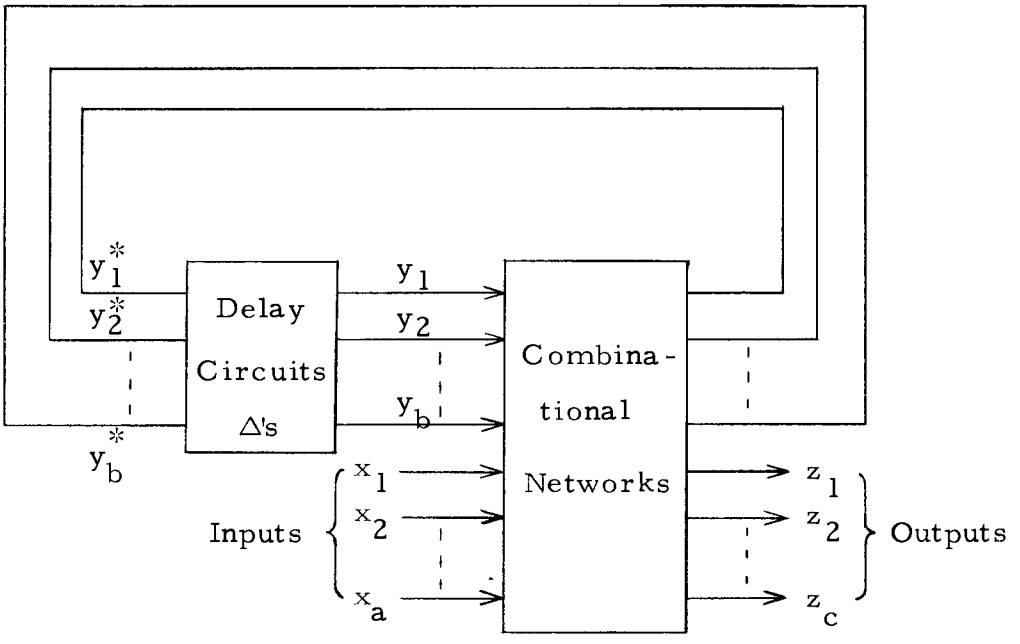


Figure 2. Generalized schematic of a sequential network.

With the input present, the set of next internal states of the sequential network (represented by y^*-notation) is dependent on both the present internal state (represented by y-notation) and the present input state (represented by x-notation).

The inputs are assumed to be independent of the y_i variables. However both the y_i^* variables and z_i variables depend on x_i and y_i variables and consequently may in special cases coincide with one or more of those variables. The output can then be expressed as Boolean functions of the inputs and internal variables as

$$z_1 = z_1(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

$$z_2 = z_2(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

$$\vdots \qquad\qquad \vdots$$

$$z_c = z_c(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

and the set of the next internal state equations are

$$y_1^* = y_1^*(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

$$y_2^* = y_2^*(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

$$\vdots \qquad\qquad \vdots$$

$$y_b^* = y_b^*(x_1, x_2, \cdots, x_a, y_1, y_2, \cdots, y_b)$$

If signals $x_1, \cdots, x_a, y_1, \cdots, y_b$ occur at time t, then $y_1^*, \cdots, y_b^*$ (the next state) will be available as the present state at the next synchronizing interval $t+\Delta$. In other words, after a delay of time Δ the y_i^* values become the y_i values. A network using such a clock for synchronizing is called a "synchronous sequential network." Another type of sequential network is "asynchronous sequential network," wherein the time required to obtain the next state variables $y_1^*, y_2^*, \cdots, y_b^*$ may not be precisely known or controlled by an external synchronizing source. In this instance, certain precautions must be taken in designing the network so that correct operation is obtained with some degree of independence from the actual delay times of the various network elements.

III. MODELING DIGITAL NETWORKS

The fundamental problem in simulating digital networks is that of economically constructing a mathematical model capable of representing the real network behavior with regard to simulation objectives. For this purpose a systematic and easily understood modeling technique is needed in order to enable the enginner to economically employ simulation as an integral part of the design process.

For this purpose let x denote the set of binary input variables;

$$x = \{x_1, x_2, \cdots, x_a\}; \qquad x_i = 0, 1$$

let y denote the set of internal state variables for sequential networks:

$$y = \{y_1, y_2, \cdots, y_b\}; \qquad y_i = 0, 1$$

let z denote the set of binary output variables:

$$z = \{z_1, z_2, \cdots, z_c\}; \qquad z_i = 0, 1$$

and let y^* denote the set of binary next internal input variables:

$$y^* = \{y^*_1, y^*_2, \cdots, y^*_b\} \qquad y^*_i = 0, 1.$$

Then the relationship between input-output of the combinational network can be symbolized as a mapping

$$m_c = < x : z >$$

Similarly, the input-output relationship of a sequential network can be symbolized as

$$m_s = < x, y : z >$$

and the input-next internal state relationship can be symbolized as

$$m_s^* = < x, y : y^* >$$

The above transformations symbolize a many-to-many mapping of Boolean variables which characterize the digital network model underlying the simulation investigation.

Since a model of digital network is composed of digital elements or logical blocks, the model of a digital network can be functionally decomposed for the purpose of obtaining a simpler method of analyzing the behavior of the real digital network. The model elements can represent an AND gate, OR gate, NOT gate, full adder, multiplier, etc.

The following are the methods by which a digital network can be modeled.

Modeling Combinational Networks

The combinational network model in Figure 1 is a collection

of interrelated combinational elements. A combinational element is a functional entity that performs a time-independent many-to-one mapping of Boolean variables. The method of modeling combinational networks will be described by the following examples.

Recall, the model representation of combinational element is

$$m_c = < x : z >$$

As an example, a p-input $(p = 2, 3, \cdots, a)$ OR gate can be modeled as follows:

$$OR = < x_1, x_2, \cdots, x_p : z >$$

or

$$z = OR(x_1, x_2, \cdots, x_p)$$

$$= x_1 + x_2 + \cdots + x_p$$

i.e. for a two-input OR gate $(p = 2)$, the model representing this element is

$$OR = < x_1, x_2 : z >$$

or

$$z = OR(x_1, x_2)$$

$$= x_1 + x_2$$

Another example, a q-input NAND gate $(q = 2, \cdots, a)$ can be modeled as follows:

$$\text{NAND} = < x_1, x_2, \cdots, x_q : z >$$

or

$$z = \text{NAND}(x_1, x_2, \cdots, x_q)$$

$$= (x_1 x_2 \cdots x_q)'$$

$$= x_1' + x_2' + \cdots + x_q'$$

i.e. for a three-input NAND gate $(q = 3)$, the model representing this element is

$$\text{NAND} = < x_1, x_2, x_3 : z >$$

or

$$z = \text{NAND}(x_1, x_2, x_3)$$

$$= (x_1 x_2 x_3)'$$

$$= x_1' + x_2' + x_3'$$

where $x_1', x_2', \cdots, x_a'$ are the complements of the inputs $x_1, x_2, \cdots, x_a$ respectively.

Modeling Sequential Networks

The model of a sequential network assumes that the present

internal state of the sequential network is uniquely determined by the previous internal state and the previous network inputs. The present network outputs are uniquely determined by the present internal state and the present network inputs or by the present internal state only, where we let

t_1 denote the previous time,

t_2 denote the present time $(t_2 = t_1 + \Delta)$.

Recall the model of sequential network

$$m_s = < x, y : z >$$

In this case, both of internal state and output state or present state are dealt with. The next step is to express these states in some forms.

The internal state transition can be expressed as

$$y(t_2) = f(x(t_1), y(t_1))$$

and the output state as

$$z(t_2) = g(x(t_2), y(t_2))$$

Again, the above expression symbolizes a many-to-many mapping of Boolean variables.

Many sequential networks can be decomposed into cascaded stages, each stage possessing more simplified sequential properties, while the overall cascade represents the behavior of the complete network.

For example, a modulo 2^n register can be decomposed into n stages, where each stage becomes a simplified sequential element (flip-flop). In this case, if we assume the T flip-flop (3), then the model takes the general form of

$$m_c = < x, y : z > \quad .$$

The next step is to characterize a behavior of a T flip-flop by means of the Boolean transformation as

$$T = < t(t_1), f(t_1) : f(t_2) >$$

where

$$x = t(t_1)$$

$$y = f(t_1)$$

and $$z = f(t_2)$$

The state transition of T flip-flop is

$$f(t_2) = t'(t_1)f(t_1) + t(t_1)f'(t_1)$$

As another example, the behavior of an RS flip-flop (3) may be

characterized by means of a Boolean transformation as

$$RS = < \{r(t_1), s(t_1)\} , f(t_1) : f(t_2) >$$

where

$$x = \{r(t_1), s(t_1)\}$$

$$y = f(t_1)$$

and $$z = f(t_2)$$

The state transition of RS flip-flop is

$$f(t_2) = r'(t_1)f(t_1) + s(t_1)$$

with the input constraint

$$r(t_1)s(t_1) = 0$$

IV. APPLICATION AREAS FOR THE SIMULATOR

Simulation can play significant roles in two areas of digital engineering: namely, designing and testing (11). These application areas which are amenable for simulation are outlined as follows:

1. Digital network behavior analysis:

   To ascertain time-quantized digital network behavior in relation to given performance specifications of the digital network for the designs under evaluation.

2. Validation of hardware implementation:

   To validate the physical implementation of a set of Boolean transformations in accordance with given performance specifications.

3. Digital network malfunctional studies:

   To ascertain digital network behavior which results from occurrence of certain element malfunctions.

4. Validation of test procedures:

   To validate the malfunction dependency on stimulus-response relationships defining a sequential test procedure.

V. FORTRAN AS A DIGITAL NETWORK SIMULATION LANGUAGE

Most of the special digital simulators (12) are computing systems which are generally constructed with two processing steps as follows:

1. The compilation of an object program representing the model.
2. Operating the model in simulated time.

It is noted that these two sequential processing steps are functionally analogous to those of FORTRAN system. Specifically, the FORTRAN language therefore can be used in preparing a digital network simulator. Another important reason for choosing the FORTRAN language is the richness of the language itself.

The FORTRAN language which is dealt with in this paper is based on the FORTRAN version available for the CDC 3300 Computer system.

The following is an informal description of FORTRAN simulator program (input language).

Description of the Simulator Program

The description of a digital network and instructions for its testing consists of two important parts: the main program and the

subprogram (Figure 3). These are described as follows:

Main Program

A main program can be written with or without reference to subprograms. This part consists of two sections, declarative and executive. The first statement of the main program which is also the first statement of the simulator must be of the following form

PROGRAM name

where "name" is a 1 to 8 character alphameric identifier beginning with a letter.

1) Declarative section: The usage of all variables must be declared. The declarative statements may be placed in any order prior to the first executable statement of a program. Typical declaration statements are

REAL list

INTEGER list

CHARACTER list

where "list" is a string of unsubscripted identifiers separated by commas. For example:

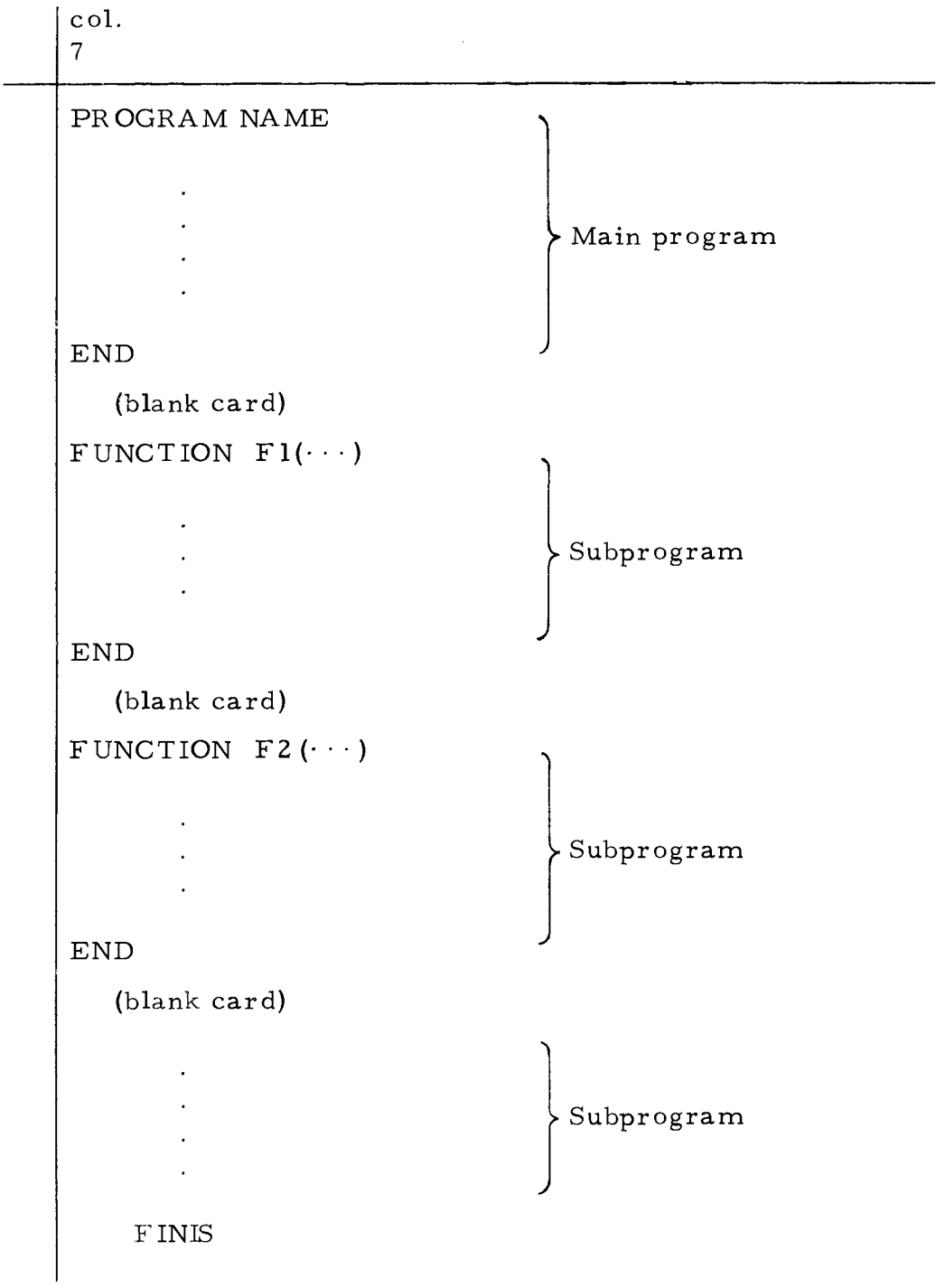


Figure 3. Description of the simulator program.

INTEGER T, A, B

DIMENSION $v_1(s_1, s_2, s_3), v_2(s_4, s_5, s_6), \cdots$

The nonexecutable statement DIMENSION reserves storage for arrays. A subscribed variable is an expression representing an element of an array of variables. An array name, v_i, has up to three unsigned integer subscripts, s_i, separated by commas. DIMENSION statements appear with other declarative statements prior to the first executable statements in the program. For example:

DIMENSION A(10), B(10)

2) Executive section: This section of the system description is to allow setting the system variables, operation of the system, and printing out the results. A variety of commands which are available in FORTRAN are permitted for checking the logic equations.

Some of the statement types are illustrated as follows:

READ n, list	Read card record
READ (i), list	Read binary record
READ (i, n) list	Read BCD record

For example:

READ 10, A, B, C

PRINT n, list

PUNCH n, list

WRITE (i) list

WRITE (i, n) list

For example:

PRINT 100, A, B

Formats are required for input and output.

Each print statement results in a printing of at least one line. The results are printed in the form of assigned formats.

function EOFCKF (i)

This statement (standing for "end-of-file check") is used for checking the end of the data. The symbol "i" is the logical unit.

GO TO n Control is transferred to statement labeled n

GO TO $(n_1, n_2, \cdots, n_m)$, e

where n_i are statement numbers, e is an arithmetic expression.

STOP Program execution ceases

The logical operators are available in FORTRAN and they are very useful in simulation. They are

$$.\text{NOT}.\, r_1$$

$$r_1 .\text{AND}.\, r_2$$

$$r_1 .\text{OR}.\, r_2$$

where r_i are simple variables, arithmetic expressions or relational expressions.

Typical arithmetic forms are available in the executive section: i.e. integer arithmetic, Boolean arithmetic etc. For example:

```
N = I*J/K
X = .NOT. X
X = 1                    X is set true
X = 0                    X is set false
```

The further extension for Boolean expression is available in the executive section, the arithmetic expression which takes the form: i.e. X = (A*B).LT.(C-D). The relation operators are

.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

Similarly, an arithmetic expression alone is a Boolean expression with value "true" if the arithmetic expression is not 0, and "false" otherwise. Furthermore, a Boolean expression of value 1 or 0 as the Boolean expression is true or false, respectively.

One of the most important statements for creating effective simulating programs is the conditional, logical IF, statement which has the form:

$$\text{IF } (\ell)\ n_1, n_2$$

Arithmetic, logical, or relational expression ℓ is true (1) or false (0). If ℓ is true, statement n_1 is executed next; if ℓ is false, statement n_2 is executed next. For example:

```
IF(.NOT.A)5,6
```

A "C" in column 1 identifies the line as comment. Comments are for the convenience of using to describe the program step.

They do not influence the program. A comment may be inserted at any point in the program.

END

The END statement marks the physical end of a program. It must be the last card of the main program and of each subprogram.

Function Subprogram

This part of a simulator contains the functions of the equations for the model elements and the functions of the equations for the intermediate signals (Boolean variables). Function subprograms may be compiled separately from the main program.

The function subprogram is bounded by a FUNCTION statement and an END statement. A function subprogram calculates the single values used in evaluating an expression. A function subprogram begins with the statement

$$\text{FUNCTION name } (p_1, p_2, \cdots, p_n) \quad 1 \leq n \leq 63$$

A function "name" contains up to eight characters, the first of which is alphabetic. Typical equations for computing the model elements are

```
AND(X1, X2)      = X1.AND. X2
MAJ(X1, X2, X3) = X1.AND. X2.OR. X2.AND. X3.OR. X1.AND.X3
EXOR(X1, X2)     = X1.AND..NOT. X2.OR..NOT. X1.AND. X2
```

The first expression represents a two-input AND element, the second one represents a three-input majority element, or it can represent the equation for the carry of the adder. The last expression is the equation of the exclusive-or.

There can be many function subprograms in a simulator. It depends on the types of model elements in the main program.

(blank card)

The blank card must be in between the main program and function subprogram, and also between function subprograms.

FINIS

The last statement of a simulator must be the FINIS statement. The FINIS statement notifies the compiler that there are no more decks to be compiled. The word FINIS must begin in column 10.

The description of a digital network simulator is shown in Figure 3, page 21.

VI. TYPICAL DIGITAL NETWORK SYSTEMS TO BE SIMULATED

Two networks, a 5-bit odd parity checker and an 18-stage binary adder, are selected for illustrating how systems can be simulated by the simulator programs (in FORTRAN). These two networks are typical combinational and sequential networks.

A 5-bit Odd Parity Checker

The Boolean variables used in modeling are defined as follows:

x_1, x_2, x_3, x_4, and x_5 are inputs.

z is an output (parity check bit).

The network which represents a 5-bit odd parity checker is shown in Figure 4. The logical elements which are used to compose this network are three-input majority elements. A three-input majority element realizes the Boolean function:

$$M(a, b, c) = ab + bc + ac$$

where a, b, and c are the inputs.

The equation for the output[1/] is

[1/] The output function "z" has been obtained by the author in his previous unpublished work on the synthesis of three-input majority networks.

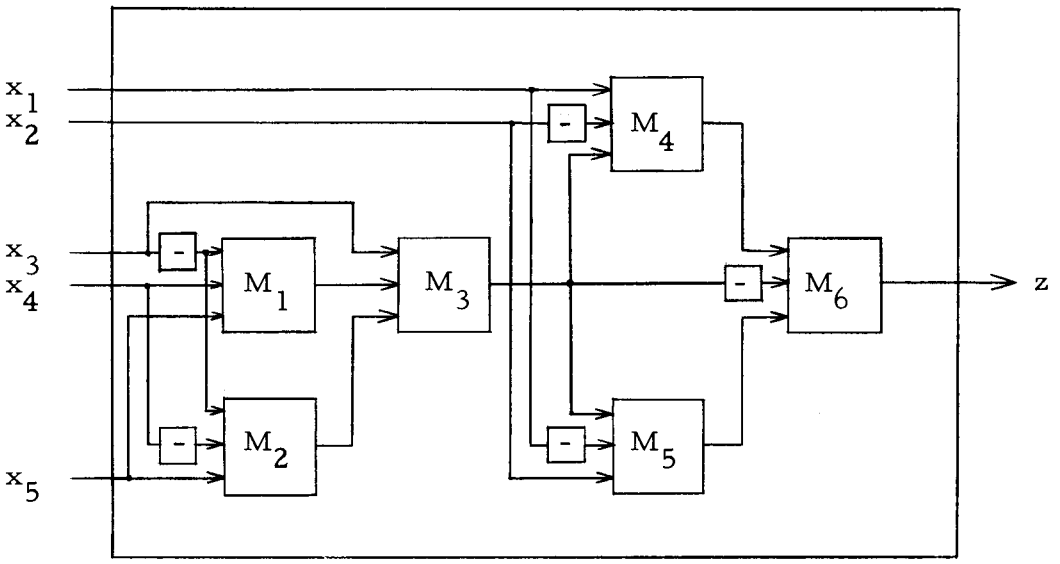


Figure 4. A 5-bit odd parity checker using three-input majority elements.

$$z = M_6(M_3'(x_3, M_1(x_3', x_4, x_5), M_2(x_3', x_4', x_5)),$$

$$M_4(M_3(x_3, M_1(x_3', x_4, x_5), M_2(x_3', x_4', x_5)), x_1, x_2'),$$

$$M_5(M_3(x_3, M_1(x_3', x_4, x_5), M_2(x_3', x_4', x_5)), x_1', x_2))$$

Figure 5 shows the output table which simulates the behavior of the network of Figure 4, the 5-bit odd parity checker, and was obtained by the simulator program. (The program is found in Appendix III.)

Simulation of a 5-bit Odd Parity Checker

	Inputs					Output
T	×5	×4	×3	×2	×1	Z
0	0	0	0	0	0	1
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	0	0	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	1	1
6	0	0	1	1	0	1
7	0	0	1	1	1	0
8	0	1	0	0	0	0
9	0	1	0	0	1	1
10	0	1	0	1	0	1
11	0	1	0	1	1	0
12	0	1	1	0	0	1
13	0	1	1	0	1	0
14	0	1	1	1	0	0
15	0	1	1	1	1	1
16	1	0	0	0	0	0
17	1	0	0	0	1	1
18	1	0	0	1	0	1
19	1	0	0	1	1	0
20	1	0	1	0	0	1
21	1	0	1	0	1	0
22	1	0	1	1	0	0
23	1	0	1	1	1	1
24	1	1	0	0	0	1
25	1	1	0	0	1	0
26	1	1	0	1	0	0
27	1	1	0	1	1	1
28	1	1	1	0	0	0
29	1	1	1	0	1	1
30	1	1	1	1	0	1
31	1	1	1	1	1	0

Figure 5. Table representing the behavior of a 5-bit odd parity checker generated by a simulator program (in FORTRAN).

An 18-stage Binary Adder

An 18-stage binary adder is shown in Figure 6.

The Boolean variables used in modeling are defined as follows:

A_i and B_i are the T flip-flops.

a_i and b_i are the augend and addend bits of stage i as specified by the present state of flip-flops A_i and B_i respectively.

s_i is the sum bit in adder stage i.

c_i is the carry bit from adder stage i-1.

a_i^* is the next stage of flip-flop A_i.

The binary addition is performed in a parallel manner on the magnitude of two 18-bit operands.

The model of the 18-stage binary adder can be described as follows:

1) Half adder stage

$$s_1 = b_1$$

$$c_2 = a_1 b_1$$

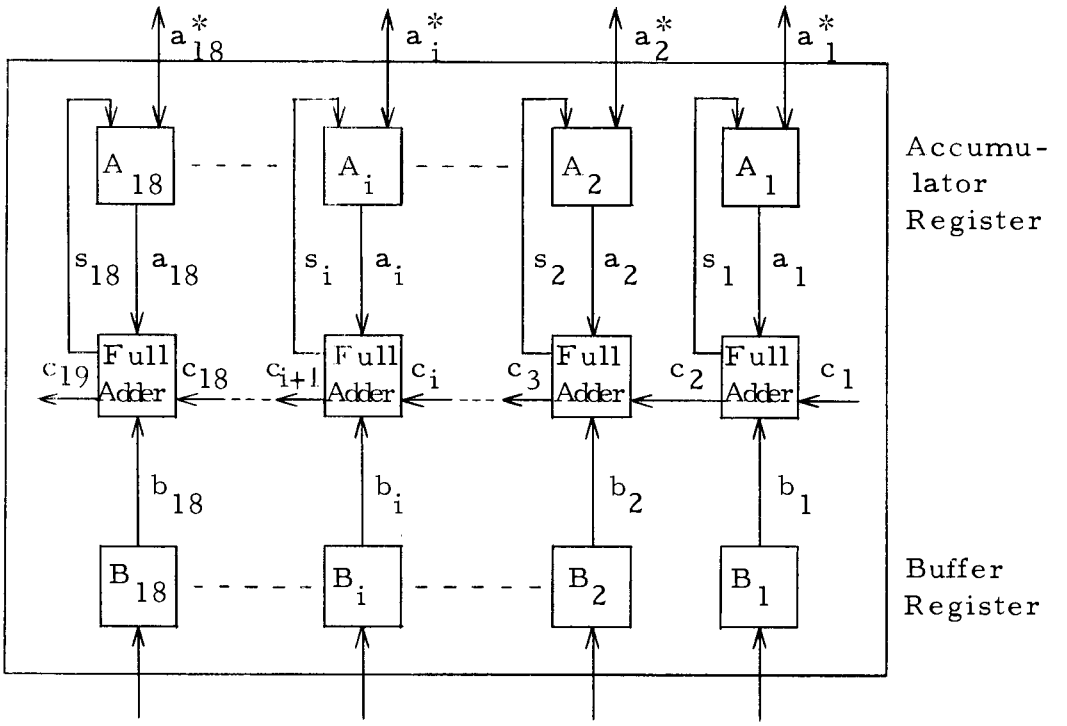


Figure 6. An 18-stage binary adder.

2) Full adder stage

$$s_i = b_i c_i' + b_i' c_i \qquad i = 2, 3, \cdots, 18$$

$$c_{i+1} = a_i b_i + b_i c_i + a_i c_i$$

3) Next state of accumulator stage

$$a_1^* = a_1 s_1' + a_1' s_1$$

since $s_1 = b_1$, then

$$a_1^* = a_1 b_1' + a_1' b_1$$

$$a_i^* = a_i (b_i c_i + b_i' c_i') + a_i' (b_i c_i' + b_i' c_i) \qquad i = 2, 3, \cdots, 18$$

The behavior-states table developed by the simulator program is shown in Figure 7 which represents the contents of the accumulator and buffer registers as a function of time. The program is found in Appendix IV.

Simulation of an 18-stage Binary Adder

	Contents of B-Register																		Contents of A-Register																	
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	O	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
T	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	V	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
M	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	F	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1
0																				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0	1	1	1	0	0	0
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1	1	1
8	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	1
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	0	0
10	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	0
11	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	1
12	1	0	1	0	1	1	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	0	0

Figure 7. Table representing the behavior-states of an 18-stage binary adder generated by a simulator program (in FORTRAN).

VII. ADVANTAGES AND DISADVANTAGES OF USING SIMULATOR PROGRAM

The following are some advantages and disadvantages of using simulator program (in FORTRAN).

Advantages

1. All FORTRAN programming capabilities (i. e. the logical operators, the relational operators, etc.) are available and these provide richer capabilities than some other programming systems.

2. The output (outputs) of the system to be simulated is easily formatted to the specific application because of the variety of output format statements in FORTRAN programming system.

3. The initial cost of the construction of a simulator program (in FORTRAN) is low compared to the initial cost of construction of most special digital network simulators.

4. Program maintenance for a simulator program (in FORTRAN) is not required for its specific application to the system, since standard FORTRAN compilers and running procedures are utilized. This is not the case for

some special digital network simulators.

Disadvantages

1. An input language program is required for each application whereas no programming is required for some special digital network simulators.

2. The size of digital network to be simulated is restricted by the limitations of the FORTRAN programming system.

VIII. COMMENTS ON THE SIMULATORS

Two simulators have been written to simulate combinational and sequential networks. These were run on the CDC 3300 Computer system under MASTER control (see Appendix I) in the Computer Center at Oregon State University. The examples chosen analyze the behavior of a 5-bit odd parity checker and an 18-stage binary adder. The source decks language is FORTRAN.

Program 1

The first system (5-bit odd parity checker) is simulated by Program 1 associated with 32 data cards (see Appendix VI). Program 1 consists of 54 cards. Of this number are 47 FORTRAN instructions, 2 blank cards (see the details in p. 27), 5 control cards (not shown in Program 1 in Appendix III, see Appendix I). It contains 2 function subprograms. They are "three-input majority" function and "not" function.

The FORTRAN program which is capable of simulating a 5-bit odd parity checker is shown in Appendix III. Of particular interest are:

Line 20-26. The output equations of each three-input majority element are set in the form of

three-input majority functions. Line 26 is the output equation of the system.

Line 41. Three-input majority function subprogram is described.

Line 45. "not" function subprogram is described.

This use of logical function subprograms increases the model compilation efficiency since Boolean function of similar form, but different variable lists, are evaluated by the same function subprogram.

The table (shown in Figure 5, p. 30), represents the behavior of a 5-bit odd parity checker. There are 32 possibilities of input sequences (for 5 variables) and 32 corresponding outputs.

The input sequences are formulated prior (shown in Appendix VI) for the simulation experiment. The simulation program reads the input data under a specified format (Line 33) with those variables expressed by 0 and 1.

Program 2

The second system (18-stage binary adder) is simulated by Program 2 associated with 13 data cards (see Appendix VI) which the first card represents the initial contents of the accumulator register and the rest of them represent the contents of the buffer

register. Program 2 consists of 78 cards. This number includes 70 FORTRAN instructions, 3 blank cards, (see p. 27), 5 control cards (not shown in Program 2 in Appendix V, see Appendix I). This simulator contains 3 function subprograms. They are "and" function, "carry" function, and "exclusive-or" function.

The FORTRAN program which is capable of simulating an 18-stage binary adder is shown in Appendix V. Of particular interest are:

Line 35. The equation of "sum" is set in term of "exclusive-or" function.

Line 36. The equation of "carry" is set in term of "carry" function which is equivalent to "three-input majority" function.

Line 60. "and" function subprogram is described.

Line 64. "carry" function subprogram is described.

Line 68. "exclusive-or" function subprogram is described.

This use of logical function subprograms increases the model compilation efficiency since Boolean functions of similar form, but different variable lists, are evaluated by the same function subprogram.

The behavior-states table which is the output result (shown in Figure 7, p. 34) presents the contents of the accumulator and

buffer registers versus quantized time. This simulation output is interpreted as follows:

(1) At bit time 0 it lists the initial contents of the accumulator register.

(2) At bit time t it lists the absolute sum in the accumulator register at t as derived from the contents of the buffer register at t plus the contents of the accumulator register at t-1.

The initial conditions and input state sequences are formulated prior (see Appendix VI) for the simulation experiment. The simulation program reads the input data under a specified format (Line 45) with variable states expressed by 0 and 1.

Running time of the first system was 3.510 seconds and 3.987 seconds for the second system. Running time of both systems can be reduced without using comment cards since comment cards are put in the simulation only for describing program step. They do not influence the program. Running time is also dependent on the size of the data. However, an accurate estimate of the running time can be made (it is automatically printed out by the computer).

BIBLIOGRAPHY

1. Breuer, M.A. Technique for simulation of computer logic. Communications of the ACM 7:443-446. 1964.

2. Buxton, J. N. and J. G. Laski. Control and simulation language. Computer Journal 5:194-199. 1962.

3. Chu, Yaohan. Digital computer design fundamentals. New York, McGraw-Hill, 1962. 481 p.

4. Control Data Corporation. 31/32/33/3500 Computer systems FORTRAN reference manual. Palo Alto, California, 1966. Various paging.

5. Conway, R.W., B.M. Johnson and W.L. Maxwell. Some problems of digital systems simulation. Management Science 6:92-110. 1959.

6. Evans, II, G.W., G. F. Wallace and G.L. Sutherland. Simulation using digital computers. Englewood Cliffs, New Jersey, Prentice-Hall, 1967. 198 p.

7. Gordon, G. A general purpose systems simulator program. In: Proceedings of the Eastern Joint Computer Conference of the American Federation of Information Processing Societies, Washington D.C., 1961. Vol. 20. [New York], 1961. p. 87-104.

8. Krasnow, H.S. and B. Merikallio. The past, present and future of general simulation languages. Management Science 11:236-267. 1963.

9. Lackner, M.R. Toward a general simulation capability. In: Proceedings of the Spring Joint Computer Conference of the American Federation of Information Processing Societies, San Francisco, 1962. Vol. 21. [Santa Monica, California], 1962. p. 1-14.

10. Larsen, R.P. Logic simulator program. Utica, New York, General Electric Corporation, Light Military Electronics Department, 1962. (Publication no. S1-144)

11. Larsen, R. P. and M. M. Mano. Modeling and simulation of digital networks. Communication of the ACM 8: 308-312. 1965.

12. McClure, R. M. A programming language for simulating digital systems. Journal of the Association for Computing Machinery 12: 14-22. 1965.

13. Miller, R. E. Switching theory. Vol. 1. New York, Wiley, 1965. 351 p.

14. Miller, R. E. Switching theory. Vol. 2. New York, Wiley, 1965. 250 p.

15. Parkhill, D. F. The challenge of the computer utility. Reading, Massachusetts, Addison-Wesley, 1966. 207 p.

16. Scramstad, H. K. Combined analog-digital technique in simulation. In: Advanced in computers. Vol 3. New York, Academic, 1962. p. 275-318.

17. Stang, D. R. Simulation of small logic system with a FORTRAN program. Computer Design 7: 56-60. 1968.

18. Teichroew, Daniel and J. F. Lubin. Computer simulation-discussion of the technique and comparison of languages. Communications of the ACM 9: 723-741. 1966.

19. Tocher, K. D. Review of simulation languages. Operations Research Quarterly 16: 189-217. 1965.

20. Warfield, J. N. Principle of logic design. Boston, Ginn, 1963. 291 p.

APPENDICES

APPENDIX I

Definition of MASTER Control Card Fields

$JOB, (1)(2), (3)/(4), (5), (6), (7), (8)

(1) Validity Code (4 characters)

(2) Programmer name or Identifier (1-12 characters)

(3) Job number (5 digits)

(4) Job identifier or Student code (2 characters)

(5) Department number (3 digits)

(6) Time limit in minutes

(7) Lines of output expected

(8) Number of cards to be punched.

$SCHED, (1), (2)

(1) CORE = M M is quarter pages (512 words) of memory needed for compilation and execution
FORTRAN needs 22

(2) SCR = N N is segments of search area of disk
FORTRAN needs 3

$FTN(L, X)

$SIM, LGO

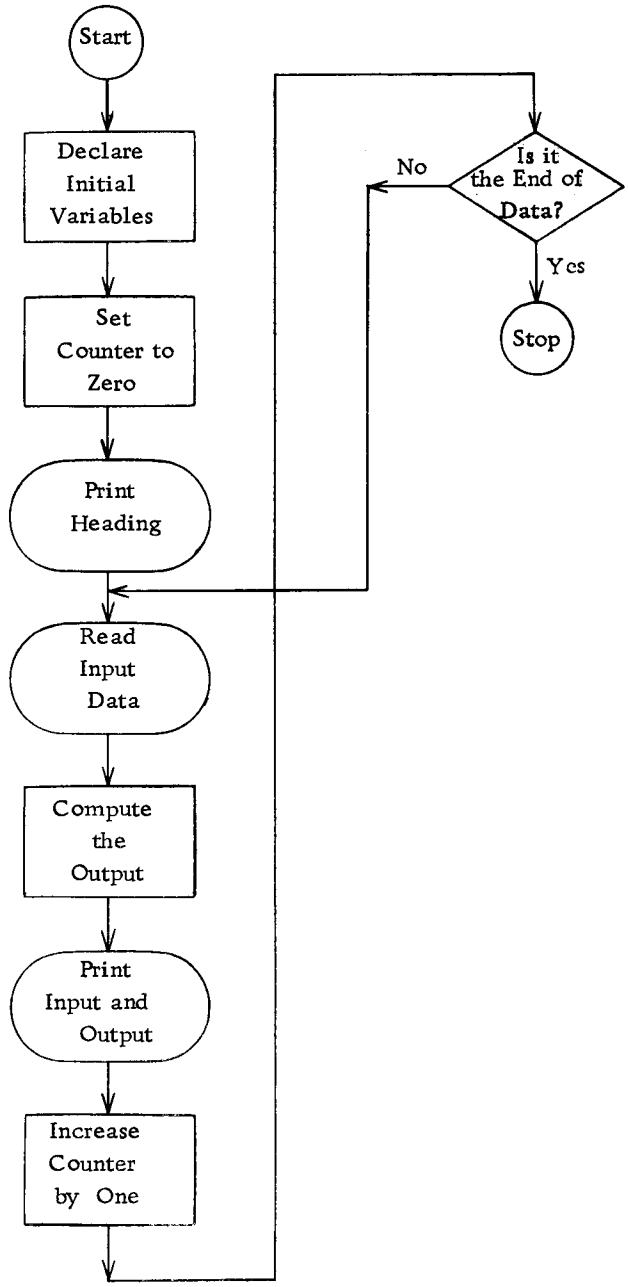
77 (End of file)
88

Typical FORTRAN Job Deck

```
$JOB,CODEUSERNAME,70000/FN,101,1,200
$SCHED,CORE=22,SCR=3
$FTN(L,X)
      PROGRAM NAME
            .
            .
            .
      END
            (blank card)
         FINIS
$SIM,LGO
            .
            .
      DATA
            .
            .
77 (End of file)
88
```

APPENDIX II

Flow chart of the program for simulating a 5-bit odd parity ckecker.



APPENDIX III

FORTRAN program for simulating a 5-bit odd parity checker.

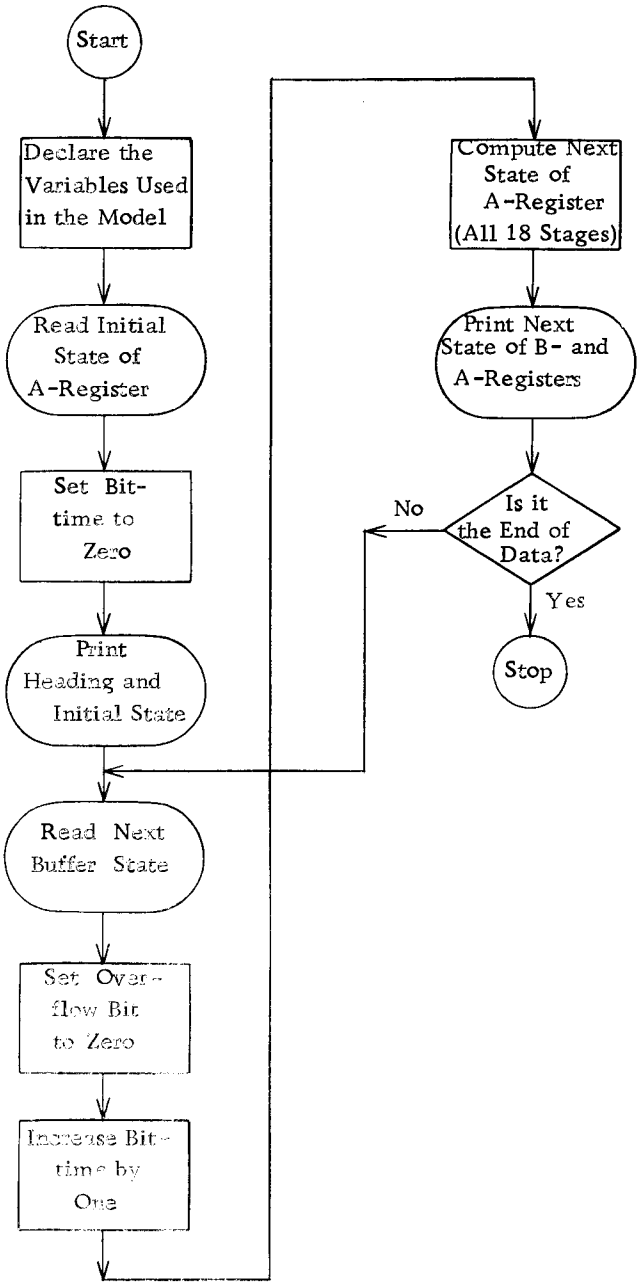
```
01              PROGRAM SIM1
02    C         SIMULATION OF A 5-BIT ODD PARITY CHECKER
03    C
04    C         DECLARATION SECTION
05    C
06              INTEGER T,X1,X2,X3,X4,X5,Z
07    C
08    C         EXECUTIVE SECTION
09    C
10    C         SET COUNTER TO ZERO
11              T = 0
12    C         PRINT HEADING
13              PRINT 110
14              PRINT 120
15              PRINT 130
16    C         READ INITIAL INPUTS
17          10  READ 100,X5,X4,X3,X2,X1
18              GO TO (1,2)EOFCKF (60)
19    C         COMPUTE OUTPUT
20           2  M1 = MAJ(N(X3),X4,N(X5))
21              M2 = MAJ(N(X3),N(X4),X5)
22              M3 = MAJ(X3,M1,M2)
23              M4 = MAJ(M3,X1,N(X2))
24              M5 = MAJ(M3,N(X1),X2)
25              M6 = MAJ(N(M3),M4,M5)
26              Z = M6
27    C         PRINT INPUTS AND OUTPUT
28              PRINT 140,T,X5,X4,X3,X2,X1,Z
29    C         INCREASE COUNTER BY ONE
30              T = T + 1
31              GO TO 10
32           1  STOP
33         100  FORMAT (5I1)
34         110  FORMAT(24X,41H SIMULATION OF A 5-BIT ODD PARITY CHECKER)
35         120  FORMAT(///,34X,7H INPUTS,19X,7H OUTPUT)
36         130  FORMAT(/,22X,42H T      X5      X4      X3      X2      X1            Z,//)
37         140  FORMAT(21X,I3,5X,I2,3X,I2,3X,I2,3X,I2,3X,I2,11X,I2)
38              END

39              FUNCTION MAJ(A,B,C)
40    C         DEFINE MAJORITY FUNCTION
41              MAJ = A.AND.B.OR.B.AND.C.OR.A.AND.C
42              END
```

```
43              FUNCTION N(D)
44    C         DEFINE NOT FUNCTION
45              N = .NOT.D
46              END
47                  FINIS
```

APPENDIX IV

Flow chart of the program for simulating an 18-stage binary adder.



APPENDIX V

FORTRAN program for simulating an 18-stage binary adder.

```
01          PROGRAM SIM2
02    C     SIMULATION OF AN 18-STAGE BINARY ADDER
03    C
04    C     DECLARATION SECTION
05    C
06          INTEGER T, A, B, C, S
07          DIMENSION A(18), B(18)
08    C
09    C     EXECUTIVE SECTION
10    C
11    C     READ INITIAL STAGE OF A-REGISTER
12          READ 100, A
13    C     SET BIT-TIME TO ZERO
14          T = 0
15    C     PRINT HEADING
16          PRINT 110
17          PRINT 120
18          PRINT 130
19          PRINT 140
20          PRINT 150
21    C     PRINT INITIAL STATE
22          PRINT 160, T, A
23    C     READ NEXT BUFFER STATE
24       10 READ 100, B
25          GO TO (1, 2)EOFCKF(60)
26    C     SET OVERFLOW BIT TO ZERO
27        2 C = 0
28    C     INCREASE BIT-TIME BY ONE
29          T = T + 1
30    C     COMPUTE NEXT STAGE OF A-REGISTER
31          C = AND(A(18), B(18))
32          A(18) = EXOR(A(18), B(18))
33          I = 2
34       20 J = 19-I
35          S = EXOR(B(J), C)
36          C = CARY(A(J), B(J), C)
37          A(J) = EXOR(A(J), S)
38          IF(J.EQ.1)30, 31
39       31 I = I + 1
40          GO TO 20
41    C     PRINT NEXT STATE
42       30 PRINT 170, T, B, C, A
43          GO TO 10
44        1 STOP
```

```
45          100  FORMAT(18I1)
46          110  FORMAT(46X, 39H SIMULATION OF AN 18-STAGE BINARY ADDER)
47          120  FORMAT(///, 30X, 23H CONTENTS OF B-REGISTER, 25X, 23H CONTENTS OF A-RE
48              1GISTER)
49          130  FORMAT(//, 21X, 88H   B   B B B B B B B B B B B B B B B B B B
50              10  A A A A A A A A A A A A A A A A A A )
51          140  FORMAT(22X, 87H  T  1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0                    V
52              11 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0)
53          150  FORMAT(22X, 87H M  8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1                   F
54              18 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1)
55          160  FORMAT(//, 22X, I2, 49X, 18I2)
56          170  FORMAT(21X, I3, 1X, 18I2, 9X, I2, 1X, 18I2)
57               END

58               FUNCTION AND (X1, X2)
59    C          DEFINE AND FUNCTION
60               AND = X1. AND. X2
61               END

62               FUNCTION CARY(X1, X2, X3)
63    C          DEFINE CARRY FUNCTION
64               CARY = X1. AND. X2. OR. X1. AND. X3. OR. X2. AND. X3
65               END

66               FUNCTION EXOR (X1, X2)
67    C          DEFINE EXCLUSIVE OR FUNCTION
68               EXOR = X1. AND.. NOT. X2. OR.. NOT. X1. AND. X2
69               END
70                   FINIS
```

APPENDIX VI

Data inputs of Program 1 and Program 2 are punched into cards from column 1-5 and column 1-18 respectively. They are as follows.

Input data of Program 1

```
0 0 0 0 0
0 0 0 0 1
0 0 0 1 0
0 0 0 1 1
0 0 1 0 0
0 0 1 0 1
0 0 1 1 0
0 0 1 1 1
0 1 0 0 0
0 1 0 0 1
0 1 0 1 0
0 1 0 1 1
0 1 1 0 0
0 1 1 0 1
0 1 1 1 0
0 1 1 1 1
1 0 0 0 0
1 0 0 0 1
1 0 0 1 0
1 0 0 1 1
1 0 1 0 0
1 0 1 0 1
1 0 1 1 0
1 0 1 1 1
1 1 0 0 0
1 1 0 0 1
1 1 0 1 0
1 1 0 1 1
1 1 1 0 0
1 1 1 0 1
1 1 1 1 0
1 1 1 1 1
```

Input data of Program 2

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1
```