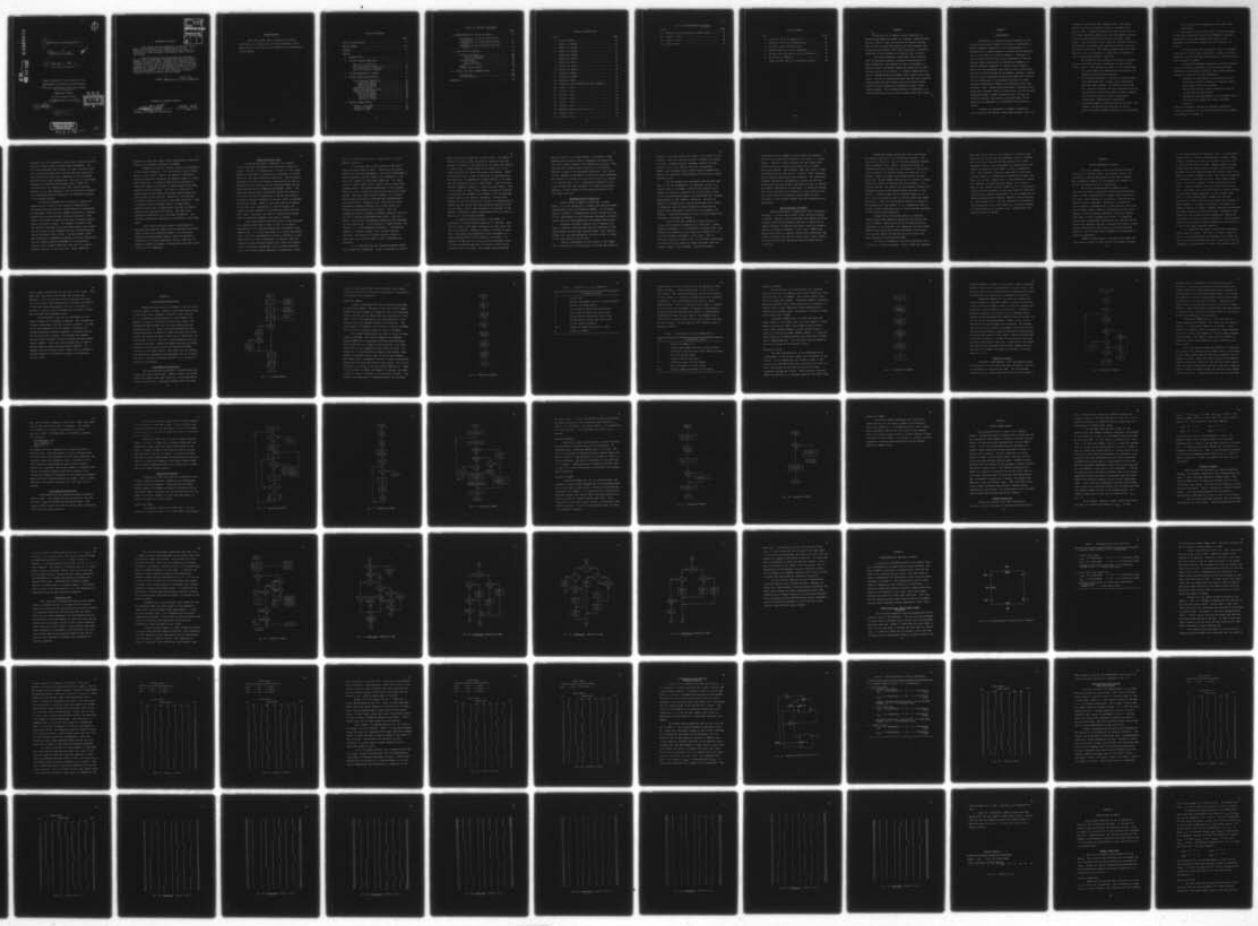
AD-A052 613 ARIZONA UNIV TUCSON DEPT OF ELECTRICAL ENGINEERING F/G 9/2
SIMULATION OF DIGITAL CIRCUITS.(U)
1976 D M MOEN

UNCLASSIFIED NL

1 OF 2

AD
A052613



(1)

(6) SIMULATION OF DIGITAL CIRCUITS.

by

(10) Dennis Milton/Moen

(9) Master's thesis,

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

(12) 115p.

(11) 1976

403696

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Dennis M. Myers

APPROVAL BY THESIS DIRECTOR

Fredrick J. Hill — 29 Dec. 1975

FREDRICK J. HILL — Date

Professor of Electrical Engineering

ACKNOWLEDGMENT

The author would like to express his sincere gratitude to Dr. Fred J. Hill for his suggestion of the topic, for his aid, and for his guidance in the preparation of this thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS--Continued

LIST OF TABLES

ABSTRACT

Simulation is a problem solving procedure for defining and analyzing a model of a system. Computer-aided design of digital logic has provided the design engineer with an aid to reduce the tedious and time consuming task of design verification. This paper describes a simulation technique for the simulation of digital logic circuits.

This paper presents a level mode logic simulator that has improved economy in execution time and ease of model generation. The passage of time is simulated in a precise fashion and element models are executed only when activity occurs in the circuit. A behavior model description is accomplished on an element level rather than a gate level. The use of three-valued logic and the use of precise timing delays for both rising and falling signal levels present a very accurate and informative circuit output timing diagram. This is demonstrated by simulation of an even-odd detection circuit and an up-down gray code counter.

CHAPTER 1

INTRODUCTION

Simulation is a problem solving procedure for defining and analyzing a model of a system; in particular, a digital computer simulation is the establishment of a mathematical or logical model of a system and the manipulation of that model on a digital computer. Designers of digital logic circuitry have long recognized the need for computer-aided design and have placed much emphasis on digital computer simulation of digital logic circuits. Menial and time consuming tasks such as logic verification are now routinely carried out by computer programs. Several types of logic simulators ranging from simple one-of-a-kind circuit simulations to very complex programs are found throughout government, industry, and education (Hill and Peterson, 1974). These simulations permit inferences to be drawn about systems; without building them, if they are only proposed systems; without disturbing them, if they are operating systems; and without destroying them, if the object of an experiment is to determine their limits of stress.

Without the assistance of computer simulation, verification of the design of any large digital circuit is

a manually tedious and time consuming task. The design engineer requires no creative skills to accomplish this routine task. Computer simulation relieves the designer of this task, therefore reducing the time and expense involved in testing designs before fabrication (Williams, 1975a, 1975b). Once the basic circuit operation has been verified, the designer can optimize his design and quickly re-verify the circuit operation. The design engineer thus has a very powerful tool in a simulator to assist him in obtaining a more nearly optimal design in less time.

The purpose of this study was to develop a computer program that would perform a simulation of large digital logic circuits. The constraints placed on the program were:

1. The input format for element descriptions was to be compatible with existing programs.
2. The simulator would be a level mode simulator where level mode means that a delay time is associated with each logic element. It should be noted that this constraint assures that the program will not be restricted to a particular family of logic elements nor a particular type of logic (synchronous, asynchronous, combinational, sequential).
3. Equivalent models for devices will not be used. For example, AND and OR logic will not be used to describe systems of NAND or NOR logic, or flip-flops

will not have to be expanded to their gate level equivalence.

4. The output will be of the form of a timing diagram.

The program, called Event Simulator (EVESIM), is primarily intended for use by the students and faculty of the Department of Electrical Engineering, The University of Arizona.

The study began with a thorough review of available literature on digital logic simulation. Chapter 2 briefly summarizes previous work and techniques while presenting the basic concepts of EVESIM.

Structured programming techniques have been applied and basic functions performed by the program have been implemented with separate routines. Using these separate routines is supported in three advantages:

1. Debugging of the program is simplified since all subroutines could be checked out separately before being added to the main program.
2. With minimal effort, someone unfamiliar with the program can gain familiarity.
3. Maintenance and improvements to the programs can be done easily by changing or adding individual routines.

The use of separate routines for separate functions is readily recognizable in the detailed description of EVESIM as presented in Chapter 4.

Since it is beyond the scope of this study to develop element models for all the available logic components, Chapters 5, 6, and 7 are dedicated to providing sufficient information to enable a user to write and implement additional element models. EVESIM, as it exists, forms a basic system that is designed to allow easy expansion. Future work to enhance the capabilities of EVESIM might include: optional data inputs/outputs and a random delay time generator. Further recommendations are included in Chapter 8.

CHAPTER 2

DIGITAL SYSTEMS SIMULATION

The usefulness and purpose of digital logic simulators has been well established in Chapter 1. It is now appropriate to review the many approaches to logic simulation. Hill and Peterson (1974) discuss the relation of cost to useful information output for several approaches to logic simulation. The cost of operation of simulators for large circuits is great, thus providing the incentive to develop efficiency in simulation techniques.

History of Simulation

Logic simulators continue, as in the past, to be developed based on how much information is required about a logic circuit. They are certainly influenced by the size of the circuit and type of circuit. Some simulators are part of sophisticated computer-aided logic design packages as developed by Hayes (1969), and others are logic verifiers as written by Williams (1975a, 1975b). Simulators, however, normally appear at least once in the automated design process prior to the design being committed to hardware. In the case of Chappell, Elmendorf, and Schmidt (1974), several different simulators are used for different purposes in the design process. For example, using a level mode

simulator will not necessarily insure that a design is valid over the allowed range of element delay parameters. The level mode simulator can, however, be used to predict worst case timing conditions and to provide some verification to clock mode design that strict adherence to the clock mode design rules has been accomplished. The use of a clock mode simulator as presented by Williams (1975a, 1975b) for verification of the combinational logic would certainly enhance a level mode simulator, thus providing the incentive for developing different simulators for different purposes in the design process.

Simulation programs differ greatly in nature and complexity as a result of the variety of applications. Simulators such as electronic network analysis as presented by Anwaruddin (1969) are very complex and costly to operate, however they provide the most accurate simulation. Simulators also differ in regard to the type of network specifications accepted as input. For example, Stockwell (1962) uses Boolean equations for circuit descriptions while others use the actual circuit elements and their interconnections, as does Williams (1975a, 1975b). Larson and Mano (1965) have presented a simulation technique which uses appropriate digital network models and FORTRAN IV to economically simulate logic networks where the high cost of a general-purpose simulator is not justified. Stang (1968) and

Weingarten (1965) have taken similar approaches by using the logical and arithmetic features of FORTRAN.

Probably the area of greatest interest in simulators is level mode simulators. Hill and Peterson (1974) discuss these in great detail. Again, the amount of information required versus cost determine the approach taken in a level mode simulation. In order to make detailed analysis of races and hazards in circuits, an analog output level mode simulator may be required. This type of simulation is accomplished by usng quantized voltage levels between the logic values of zero and one. Neglecting rise time and only delay results in a 2-level level mode simulation. Such a simulator using an up-down counter to model delay presently exists and is accessible on the DEC 10 computer system at The University of Arizona. The most basic approach to simulation is a clock mode simulation. One iteration through the simulation is equivalent to one clock pulse.

Great effort has been placed in development of advanced techniques for improving the calculation and efficiency of the basic simulation models to be discussed. Event-directed techniques (Breuer, 1970) and exclusive simulation of activity techniques (Ulrich, 1969) are intended to reduce the amount of calculations required in one iteration of a simulation.

Modeling Digital Logic

As stated previously, simulation is a problem solving procedure for defining and analyzing a model of a system. The value of simulations, of course, is that large amounts of data about a system can be collected in a short period of time under controlled conditions. How accurately the behavior of the real system is depicted by the data is reflective of the accuracy of the simulation model and the model characteristics (Szygenda and Thompson, 1975). The simulator designer must first decide whether the system is to be used for logic verification or design verification. He needs to ascertain whether to use two, three, or multiple gate output values. There is also the question of whether to use zero, unit, assignable, or precise delays; and closely related, whether to use differing signal rise and fall times. Certainly, the type of internal device model must be considered, whether it be gate or element.

Logic verification would be similar to the operation of a combinational logic circuit in that it is purely a computation of all possible outputs for all possible input combinations. It is also equivalent to a clock mode simulation. Since the implementation of such a simulator is very simple and if no other information is required, then the simulator designer should consider only logic verification. If more information is desired, such as timing considerations, then a more complex approach is appropriate, noting

that logic verification can be a special case of a more complex simulator.

It is obvious that a logic simulator requires at least two logic levels, 0 and 1. Using only two values, however, has two major drawbacks. The first disadvantage is that establishing initial values on all elements is very difficult, except for clock mode simulation. For large circuits, establishing initial element output values then becomes costly. The second disadvantage is that unknown conditions due to spikes or other sources can not be represented. The third value normally added is X or unknown which allows simulators to overcome the disadvantages of two-valued logic. Chappell et al. (1974) define a non-propagating unknown condition that is used during the process of establishing initial values. This non-propagating feature prevents an unknown condition from over writing an already established known value. They use a fourth value called a propagating unknown for the remainder of their simulation. It is feasible to expand further; for example 0, 1, 2 (non-propagating unknown), 3 (propagating unknown), 4 (signal rising), and 5 (signal falling). Certainly, the addition of 4 and 5 would be useful in simulating rise and fall times if that information was required.

It is important that the difference between gates and elements be established. A gate is defined as a device

that has multiple inputs and a single output. An element is defined as a device that has multiple inputs and multiple outputs; in addition, the ordering of the inputs and outputs is significant. A good example of an element is a flip-flop; certainly the inputs and outputs are ordered. These differences result in major differences of implementation. The complexity of implementing element models is the reason that most simulators use only gate models. There are some serious drawbacks, however, in using only gate models. The first of these is that a large number of gates are required to implement some elements, for example, a JK flip-flop may require eight to ten gates to implement. Another major drawback is that the interaction of gates in implementing an element may not present the same operating characteristics as the element itself, thus presenting a poor model of the element. Therefore, careful consideration must be given to the type of internal device models.

To achieve greater accuracy in the model, an assignable or precise timing capability is required. Zero and unit delay (clock mode) simulators are only useful for logic verification. Timing is inherent in design verification. Assignable delay simulators allow assignment of a single average delay to each element or gate type. For example, one AND gate in a circuit may be assigned 5 units of delay and another AND gate in the same circuit may be assigned 4 units of delay. The up-down timing technique

mentioned earlier is a good example. This model is made somewhat more precise if delay assignments are made on an individual element based on such parameters as fanout. This technique also allows circuits consisting of different families of logic to be accurately simulated. This can be further expanded to assignment of different delays dependent upon whether the signal is rising from logical 0 to logical 1, or falling from 1 to 0. Though this technique is not a true rise and fall time simulator, as needed in electronic analysis, it does present a more accurate picture of the real world situation.

Implementation of Simulators

Having looked at modeling techniques in great detail, it is now appropriate to answer some questions about implementation prior to discussing EVESIM. At a minimum, areas of interest in implementation should include the ease of use, independence of data processing equipment, selective tracing and time flow mechanisms, and element descriptions.

Engineers who make extensive use of computer programs, invariably avoid programs that are difficult to use. Good programs minimize user required tasks. This requires complex pre- and post-processing, but must be done in order to gain wide usage of the program.

Machine independence of the program is very important, particularly if the program is large and was costly to

develop. A program is not very useful if the author of the program is the only one that can use it because his machine is unique. Though low-level languages normally provide a savings in run time, they are not very transportable. Higher level languages such as ANSI standard FORTRAN have proven to be much more transportable and therefore, a better language to use.

Gate and element have previously been defined; however, as far as implementation is concerned, a gate is a special case of an element. Defining the operation of an element in terms of a subroutine or in-line code is relatively straightforward assuming the programmer has a through understanding of the element's operation. Defining an element in a subroutine has obvious advantages such as ease of debugging, ease of being replaced or overwritten by future users, and ease of calling for use throughout the program. A detailed discussion of how to write element subroutines is contained in Chapter 5.

Breuer (1970) has suggested a partitioning of a circuit into subsets of elements $U_j(i)$ such that if after simulating element i, it is found that i changes state, then simulate the elements of $U_j(i)$ where the elements of $U_j(i)$ are those elements that have i as an input. This is selective tracing; in other words, if an element output does not change when new inputs have been evaluated, then the element output is not followed. Since partitioning

algorithms are very complex, Ulrich (1969) has suggested that this partitioning be limited to the fanout of a single element. Breuer's (1970) technique has been named event-directed simulation, where event means that an element output has changed. The simulator jumps from event to event as opposed to being time driven. Ulrich's (1969) technique can be labeled time-driven event simulator. The time-driven simulator operates by incrementing a fixed time unit and checking for activity at that time where activity is defined as elements scheduled to be evaluated. It should be obvious that these techniques are required for large circuits since any system relying on evaluating each element on each increment of time quickly becomes too time consuming and in turn costly.

Design Concepts of EVESIM

Having established a background of basic simulation concepts, it is now time to begin a more detailed analysis of EVESIM. Chapter 1 well established that EVESIM should be a design verification simulator and that it should use an element approach to modeling; therefore, these two subjects warrant no further discussion at this time. EVESIM uses three-valued logic, it applies precise delay techniques that reflect different rise and fall times, and it uses a hybrid of event-directed and time-driven event-directed control techniques.

EVESIM uses three-valued logic and in particular, the unknown condition X, for two primary purposes. The first use is similar to that of the non-propagating unknown used by Chappell et al. (1974). During the process of establishing initial values for the elements of the circuit to be simulated, all element outputs are first set to X, unknown, except for memory type elements which are user specified. Memory element outputs are not allowed to change during the process of updating the other elements. In a circular process, new outputs for the remaining elements are calculated until no more unknown conditions exist in the circuit. A default mode and an override capability are built into the system in case all unknown conditions cannot be cleared. (This will be discussed further in Chapter 4.) The second use of the unknown is in a propagating mode to reflect illegal conditions such as timing problems generated in the circuit being simulated.

Though the use of precise delays for modeling normally means more complex programs, precise delays were used in EVESIM. Based on the use of the hybrid driving technique, to be discussed, this complexity has been offset. EVESIM allows individual element assignment of differing rising signal delay time and falling signal delay time, thus presenting a more accurate model.

The heart of EVESIM is the driving mechanism, as it is with any simulation program. Ulrich (1969) has reported

that only one per cent of all elements in a typical simulated digital network are simultaneously active. Assuming this to be true, then any driving technique that takes advantage of this is an improvement over conventional techniques of simulating each element in every increment of time. This suggests a discrete event type of simulation. In other words, it is desirable to simulate an element only when it is active. Define a set C, the elements of C being the elements, including external inputs, of a circuit C. Then, at any instant of time a subset A_1 of C, consisting of currently active elements, is the only source of future network activity, signals originating in A_1 are transmitted to associated destination elements, forming a destination set D_1, also a subset of C. Not all elements of D_1 respond to the stimulation, thus a new subset, A_2, of active elements is formed where A_2 is a subset of D_1. EVESIM takes full advantage of this concept.

CHAPTER 3

DRIVING MECHANISM OF EVESIM

Until now, simulator driving mechanisms have been discussed in a general sense. This chapter carries the discussion of driving mechanisms to that used in EVESIM. The information required for the modeling of elements in the driving mechanism is also identified.

The driving mechanism used by EVESIM is best identified as being a time-driven event-directed mechanism where an event is defined as an element output taking on a new value. The technique is a hybrid of a time-driven and and event-driven mechanism. Basically, a time-driven circuit model is incremented by a fixed time unit and a new set of element output values is calculated per time incremented. An event-directed approach is driven by jumping from event to event irrespective of time. It should be obvious that considerable calculations are required for the time driven approach, and that a considerable amount of data must be maintained for each event in the event-directed approach. The technique used by EVESIM is a combination of these two approaches.

EVESIM is time driven in the sense that events can occur only at points in time that are an integral multiple

of the fixed increment of simulation time. In other words, simulation time is merely represented by a counter, $TIME_{n+1} = TIME_n + 1$, beginning at time 0, and events can occur only at these integer values of TIME. Element models are used to determine new output values as element inputs change. The model must also determine when the element output is to change in terms of simulation time. This timing information is calculated by the model from user input element delay times. This time of change or time of occurrence for an event is used to properly insert the event into an event list. This event list is the only source of future activity in the circuit being simulated. In other terms, these events or known element output changes are the only occurrences in the circuit that can cause future changes of other elements. The event is executed by following the element output to all fanout elements connected and checking the fanout elements for future activity. Therefore, simulating only the effects of these changes has reduced the simulation to an event directed simulation. Allowing these events to occur only at fixed increments of time results in a time driven event directed simulator.

To accomplish the tasks of driving the simulation, EVESIM uses two arrays. The first is ICL(N,2) where N is equal to the maximum delay time assigned to an element of the circuit being simulated plus one. Each row of ICL(I,J) represents a discrete point in time. The second column of

ICL(I,J) contains a pointer to the second array, the event list array LIST(K). The first column of ICL(I,J) gives the total number of events in the event list for time I. The remaining time I events follow in the next ICL(I,1)-1 elements in LIST(K). Time is incremented as discussed above and a modulo function is used to determine the value for I from TIME. Thus, the simulation time at which an event is to occur is described implicitly by the location of the pointer in the ICL(I,J) array. The use of these two arrays in this fashion allows a very long simulation over many increments of simulation time without requiring a large storage array to maintain timing information.

With this background, a general flow through the program can be outlined. The first step is to increment TIME. The clocking array, ICL(I,J), is then accessed. If there are events scheduled for this time, they are executed as listed in the events list array. Each fanout element of each element in the event list is in turn simulated with the appropriate element model. An element status and description array, IDESOU(I,J), is used to maintain a cell of information about each element in the circuit. The information included in this array is a list of fanout elements for each element of the circuit, the simulation time when the last output change occurred, the old value of the element output, the present value of the element output, the simulation scheduled, and the future value of the element output.

With this information the element model determines a new element output and calculates the simulation time of a future element output change. If the event will cause a future state change to occur in the fanout element, the future event is properly scheduled into the event list. When all events have been processed, time is again incremented and the process continues.

It is interesting to note that if the cyclic clock discussed above is based on maximum element delay time, old information in the event list will automatically be replaced with new information. This reduces the required length of the event list. For example, assume that the maximum delay of any element in a circuit is 10 units of time. Define LIST(I) to have I = 1000. With a clock length of 10 units, then each clock unit of time could point to, on an average of 100 locations in LIST(1000) or 100 events could be scheduled per increment of time. Based on the work by Ulrich (1969), one per cent of the circuit elements is equal to the 100 events or a circuit comprising of 10,000 elements could be simulated by this example without increasing the length of LIST(I).

It has already been pointed out that an element model must be writtin in order to not only determine the new value of an element, but also to determine when the change is to occur. The information required to make these calculations must include the simulation time when the last

output change occurred and the old value of the output. The model must also know if the element has already been scheduled. This can be done either by checking the event list or by adding this information to the above and storing it with the element description cell as is the other information. Using a cell of information, of course, is much more desirable than searching lists.

Also included in the cell of information is the fanout elements of the element concerned. This information is used to determine a new set of active elements. In order to get all parameters into the scheme, internally the program generates fictitious elements to represent each of the external circuit inputs. The program also expands multiple output elements to look as though they were single output elements. Thus, the timing scheme does not need a separate mode to handle multiple output elements. All elements and the external circuit inputs are all handled in the same fashion, and therefore minimize the simulation control effort.

CHAPTER 4

EVESIM PROGRAM DESCRIPTION

EVESIM has been written in FORTRAN IV for use on the CDC 6400 computer system. EVESIM is best described as being a time-driven event simulator of digital logic circuits. It is a fundamental mode simulator in that elements are modeled in terms of rising and falling delay times. The program has been divided into five major subroutines as indicated in Fig. 1. There are several subroutines that perform functions in support of these five and will be discussed individually as their time of use is identified. As can be seen in Fig. 1, the first three subroutines are preprocessing routines. Subroutine FUNSIM controls the actual simulation of the logic circuit and subroutine OUTPUT controls, as the name implies, data output for the program. The program has debugging aids programmed into it to assist the user in running the program and for use in future updating of the program.

Preprocessing Subroutines

The preprocessing requirements of EVESIM have been broken into three subroutines: READIN, INCOND, and INTIAL. Subroutine READIN reads from cards the circuit description array of the circuit. Subroutine INCOND reads from cards

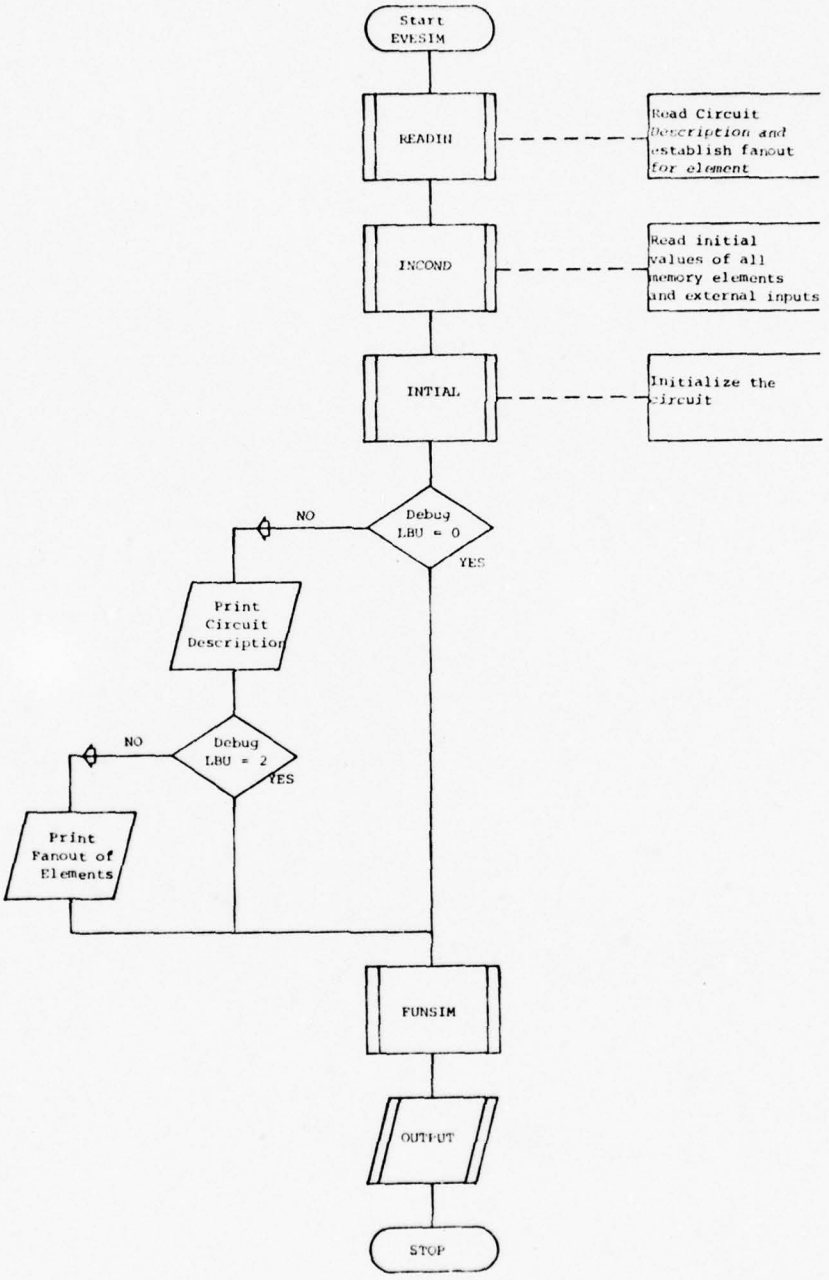


Fig. 1. Program EVESIM.

initial circuit values and initial external input values, and subroutine INTIAL initializes the circuit in preparation for beginning the simulation.

Subroutine READIN

Figure 2 describes the flow of functions performed by subroutine READIN. The date of run, the circuit number (LCKT), the total number of simulation times to be executed (LAST), and the mode of debug (LBU) are read from the first card. Since error messages can be generated early in the program, the error message heading is printed next. The total number of elements in the circuit is read in as NGATE, and the total of external inputs is read in as NEXIN.

The element descriptions are read in using the same format as SCIRTSS. The circuit elements are numbered 1 through NGATE and are assigned a 4-letter type designator such as JKM1 for a J-K master slave flip-flop. The input connections to the element are then listed in order. The connection is indicated by the number of the element whose output is connected to the input of the present element. The element delay information follows in terms of simulation times both for rising and falling logic values. All of this information is stored in an array called IDESCR(I,J). Index I will be equal to NGATE, the number of elements, and Index J can have values 1 through 19 as indicated in Table 1. The element type designator is compared against the available

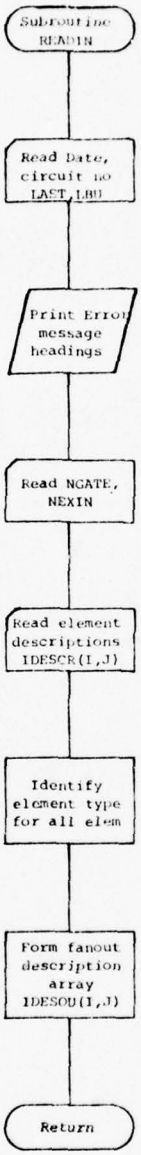


Fig. 2. Subroutine READIN.

Table 1. Definition of J for IDESCR(I,J).

J	Information stored
1	Element type
2	Element type number used for internal control
3	Pointer to IDESOU(IOD,JO)
4	First value of delay time low to high
5	First value of delay time high to low
6	Second value of delay time low to high
7	Second value of delay time high to low
8	Circuit output indicator
9-18	Number of element connected to input
19	Status of element

element models listed in LTYAL(J) and the appropriate type number assigned. A fanout description of the circuit is then formed. This fanout description, based on the fanout of each element, is in effect the destination set discussed in Chapter 2. The fanout description array is called IDESOU(IOD,JO). IOD is a function of the number of elements in the circuit and the number of outputs of each element. An element that has more than one output is represented in IDESOU(IOD,JO) once for each output. For example, each flip-flop will appear twice, once for the Q output and once for the $\overline{Q}$ output. The JO index for each element output is defined in Table 2.

Table 2. Definition of JO for IDESOU(IOD,JO).

JO	Information stored
1	Pointer to IDESCR(I,J)
2	Element type number used for internal control
3	Simulation time when last output change occurred
4	Old value of the output
5	Present value of the output
6	Simulation time of future change
7	Value of output at future change
8-17	Element numbers connected to the output

Subroutine INCOND

Though normally all required data for a program would be read into the computer by a single subroutine, this task has been split in EVESIM. This allows greater flexibility in input data formats. Subroutine READIN is oriented around the SCRITSS input format and may not be suitable to another user's needs; therefore, doing preprocessing in smaller pieces allows easier replacement of sections to meet a particular user's needs.

Subroutine INCOND (Fig. 3) reads from cards the total number of flip-flops (LFF), the total number of other memory type devices (LOM), two minimum values of clock pulse widths (NCLWID, NCLW2), and two minimum values of preset or clear pulse widths (NPCWID, NPCW2). It reads the output value of all memory devices and stores that information in IDESOU(IOD,JO5). The final data read by INCOND is the initial values of all external inputs.

Subroutine INTIAL

The most important part of the preprocessing is establishment of the initial values for all elements of the circuit. It is assumed that the external inputs to the circuit have remained stable for a long period of time, in fact, long enough for any previous activity to have propagated through the circuit. This situation certainly cannot be achieved if an unstable condition can result from

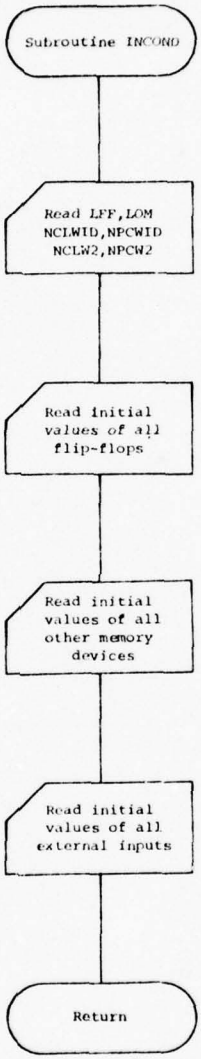


Fig. 3. Subroutine INCOND.

network feedback internal to the circuit. Such an unstable condition may or may not be designed into the circuit. Any unstable condition is recognized by the program and the user provided with sufficient data to diagnose the situation.

Subroutine INTIAL (Fig. 4) begins by reading the maximum number of iterations (LIMIT) or passes through the circuit that can be made in order to clear all unknowns and establish a stable condition in the circuit. Failure to establish stability will result in error messages being printed and program execution stopped, unless of course, the initialization override parameter, LOVER, is set to one. New outputs for each element are calculated. The unknown condition X is not allowed to propagate through the circuit, and therefore provides a means of identifying indeterminate situations that exist in the circuit. These situations may or may not be desired. If they are not desired, then they have been identified to the user. Iterations of new output calculations are continued until no more changes occur or the limit of attempts is reached as outlined in the flow chart in Fig. 4.

Simulation Control

Efficiency requirements in the preprocessing phase of simulation is easily put aside when considering that the preprocessing is done only one time. Even the program efficiency of an element model does not have to be great if

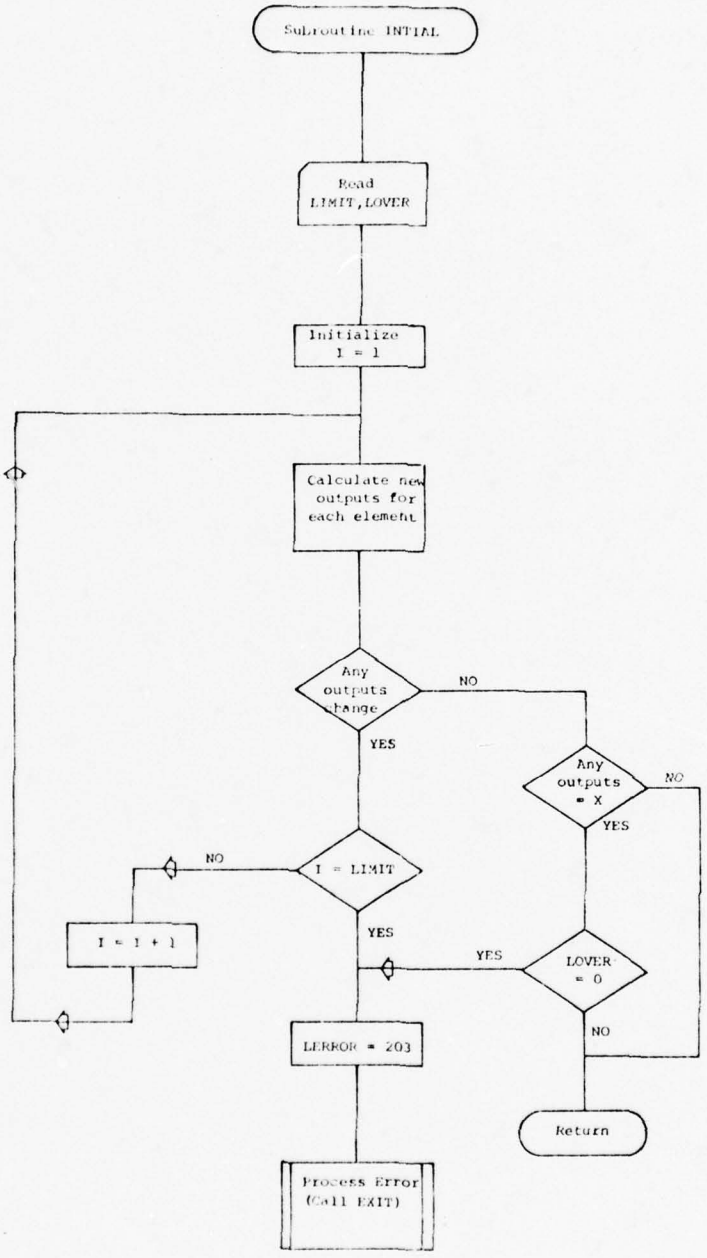


Fig. 4. Subroutine INTIAL.

maximum effort is exerted to call the model only when necessary. Certainly any simulator that calculates new element outputs for each element during each increment of time requires a very efficient element model. The process of searching long lists as proposed by pure event directed simulations is also not very efficient. EVESIM does not require major list searching routines or efficient element modeling. Great flexibility and efficiency have been gained in EVESIM by not involving the element model in the control and driving mechanism of the simulator.

Subroutine FUNSIM (Fig. 5) is the control and driving mechanism for EVESIM. The event list, discussed in Chapter 3, is defined for EVESIM to be LIST(K). K was chosen to be equal to 1024 for purposes of easier manipulation of the events list; however, K is by no means limited to this number. The internal clock array for EVESIM is defined as ICL(I,J) and simulation time is defined to be equal to NOW.

The first step in subroutine FUNSIM is to determine the length of the internal clock, LENCL, which is equivalent to the longest delay time that has been assigned to any one element plus one. Then, the initial value of the elements' outputs is written to file IOU, and the first event card is read. An event card contains the event time for a change in value of external circuit inputs, the external input number, and the new value. A card can have only one time value but

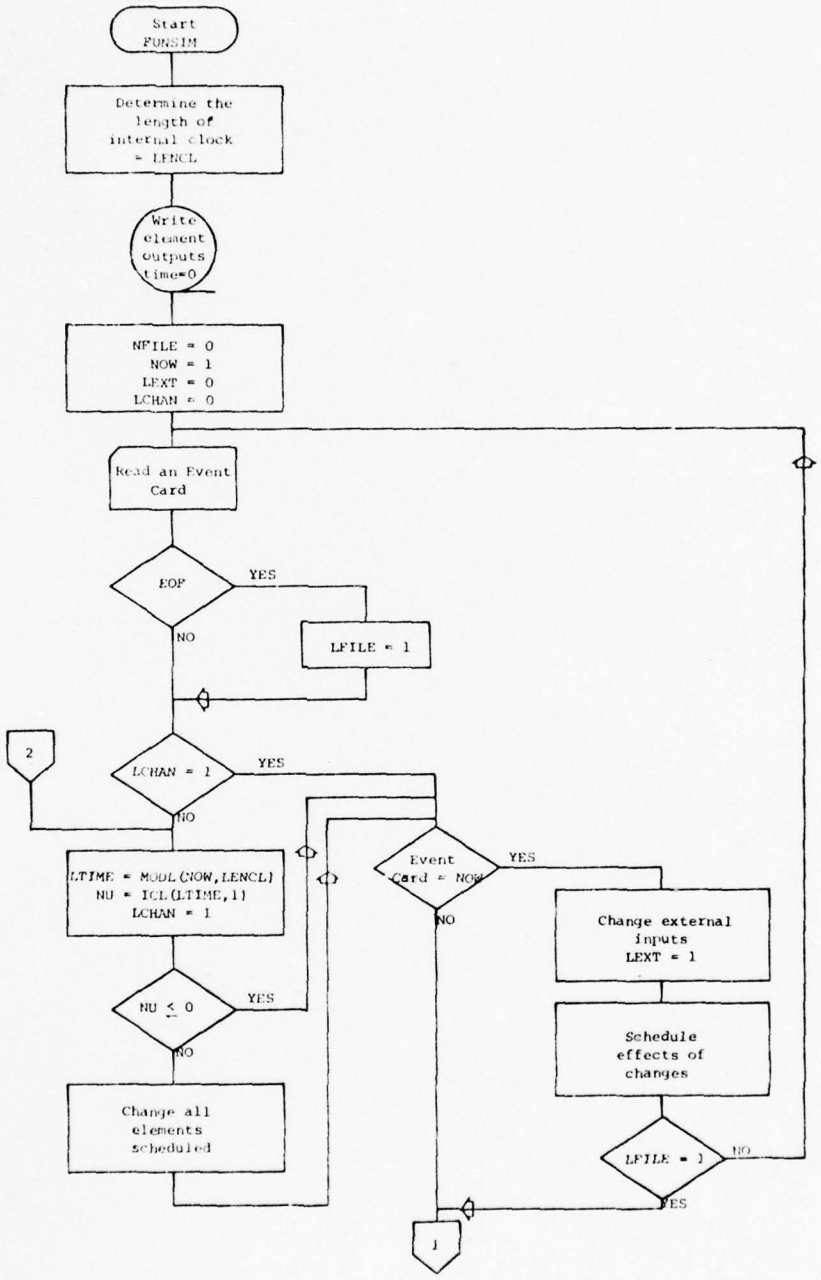


Fig. 5. Subroutine FUNSIM.

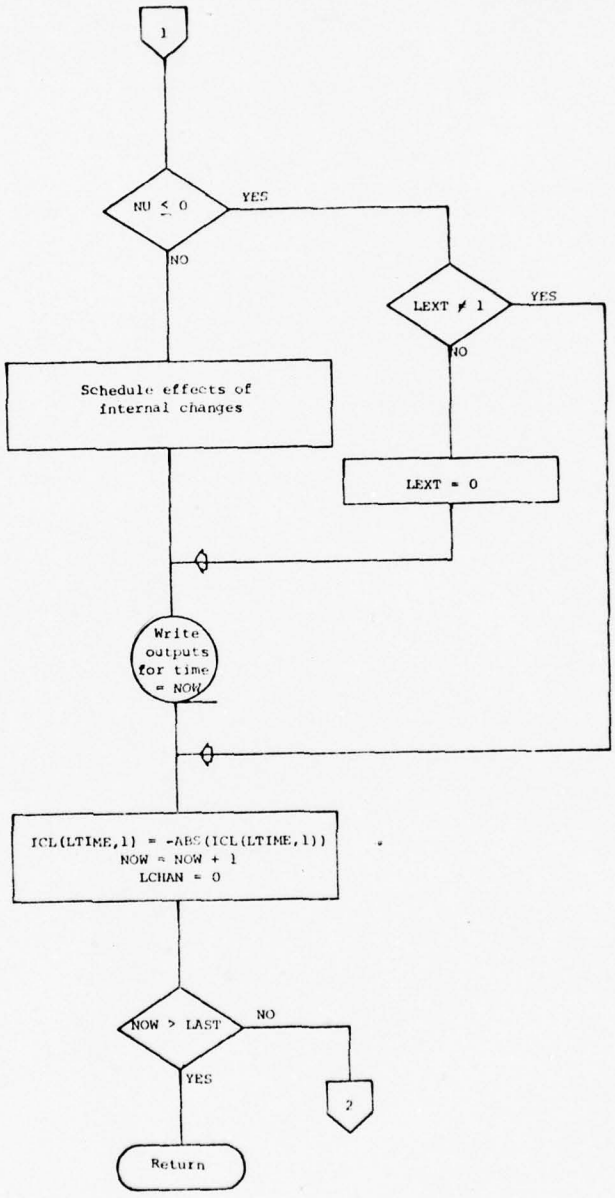


Fig. 5--Continued

may contain several changes for that time. Additional cards for the same time may be used if necessary. The present time (LTIME) of the clock is determined by a modulo function, LTIME = MOD1(NOW,LENCL), where MOD1 is defined as a function:

```
FUNCTION MOD1 (M,N)
LT = M-(M÷N)xN
IF(LT.EQ.O)LT = N
MOD1 = LT
```

The number of events scheduled for a particular time is defined as NU. NU being negative in value indicates that the events have been processed for that time; otherwise, the scheduled events are allowed to occur. External circuit input changes are processed and then internal element changes are processed. During the process of scheduling the effects of element changes, each fanout element of the element that changed is processed by the appropriate element model and future events scheduled as needed. Time is incremented and the process continues until the time limit is reached.

Postprocessing Subroutines

Postprocessing of the simulation merely provides a usable output. The process has been divided into two subroutines, subroutine OUTPUT and subroutine PLOT1. Subroutine OUTPUT is used for control purposes while subroutine PLOT1 does the actual printing.

The external circuit inputs and the element outputs to be plotted are read from a card. Up to 12 outputs can be requested per card. As indicated in Fig. 6, the output requests are first sorted and subroutine PLOT1 is called to do the printing.

Subroutine PLOT1 (Fig. 7) reads a record from the output file IOU. It then fills an output buffer with the appropriate output value and prints the contents of the buffer. Another record is read from IOU and the process continues until the file is empty. It should be obvious that the printing routine PLOT1 has been separated from the control in order that changes to the actual output format can be easily accomplished with minor change only to PLOT1 and not the whole output process.

Supporting Functions

Discussion to this point has only referred to such things as modeling routines, scheduling and unscheduling events, and error messages. Though subroutines SCHED, UNSCHE, ERROR, and IODEST are normally referenced only by the element model routines, they are discussed prior to the modeling routines (Chapter 4) since they are general in nature and apply to all the models.

Subroutine SCHED

The purpose of subroutine SCHED (Fig. 8) is to schedule an event to occur at the time (NSCTI) specified by

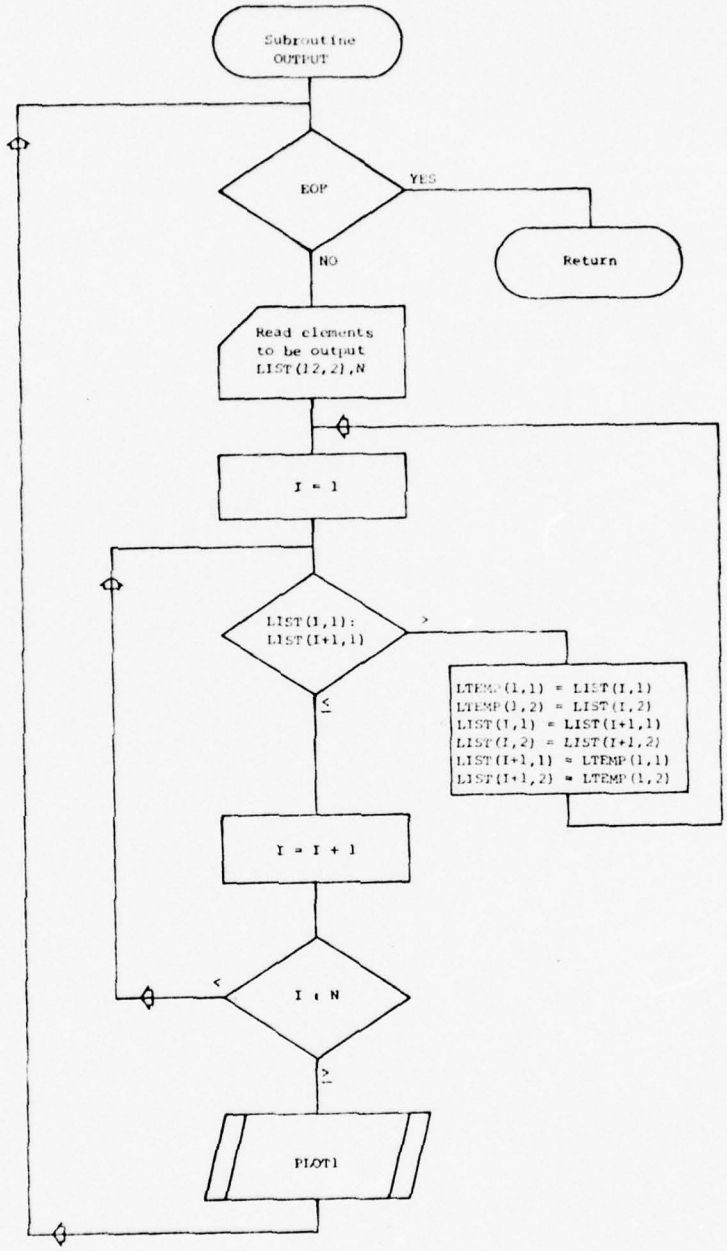


Fig. 6. Subroutine OUTPUT.

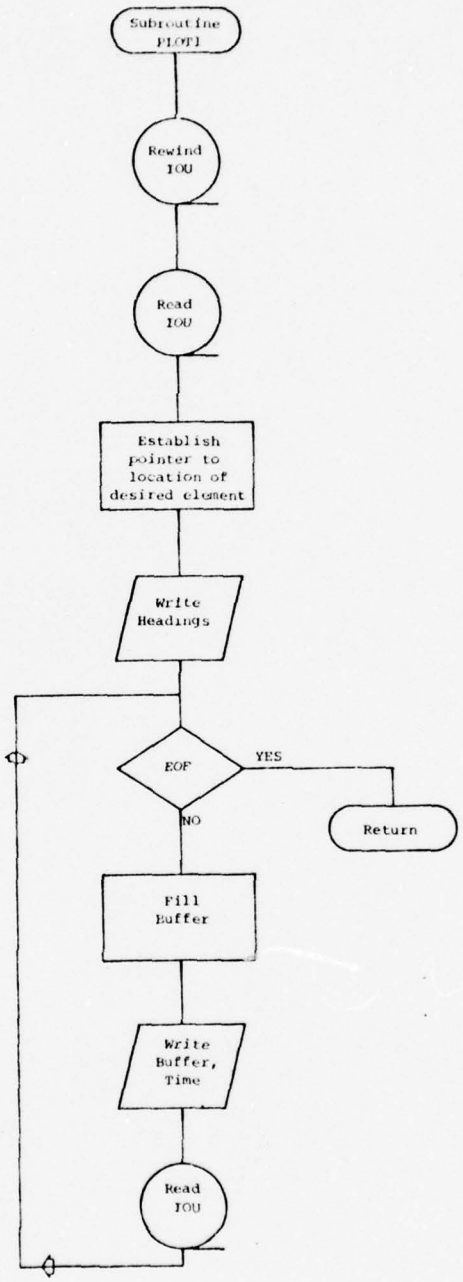


Fig. 7. Subroutine PLOT1.

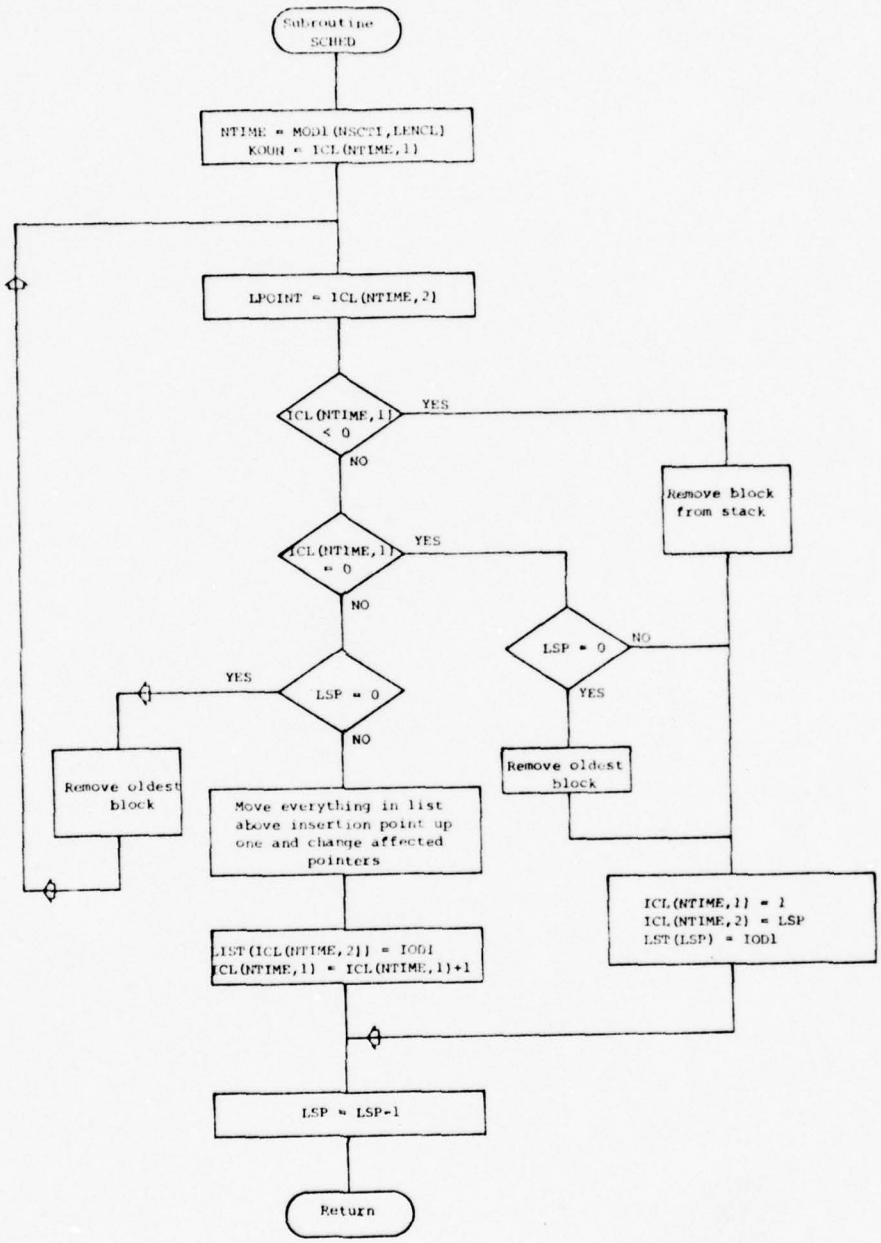


Fig. 8. Subroutine SCHED.

an element model. It is a very simple process and involves determining whether or not a block has been reserved in LIST(I) for time NSCTI. If a block exists, it is opened and the event inserted. If a block does not exist, one is created.

Subroutine UNSCHE

There are certain instances where an event that had been previously scheduled needs to be unscheduled. To accomplish this, subroutine UNSCHE (Fig. 9) is used. The event to be removed is identified by the element model, the pointer (LPOIN) to the beginning of the block in LIST(I) is then established. The block is searched for the event and it is removed. IDESOU(IOD,JO6) is updated to reflect that no event is to occur and control is returned to the element model subroutine.

Subroutine ERROR

Subroutine ERROR (Fig. 10) is used throughout the program to process error messages and to stop program execution in the case of fatal errors. Errors are processed by assignment of a number. It is noted by Fig. 10 that error messages greater than 199 are fatal and cause execution to cease by using the library function CALL EXIT. After processing of non-fatal errors, control is returned to the calling location. A list of error messages that can occur is contained in Chapter 7.

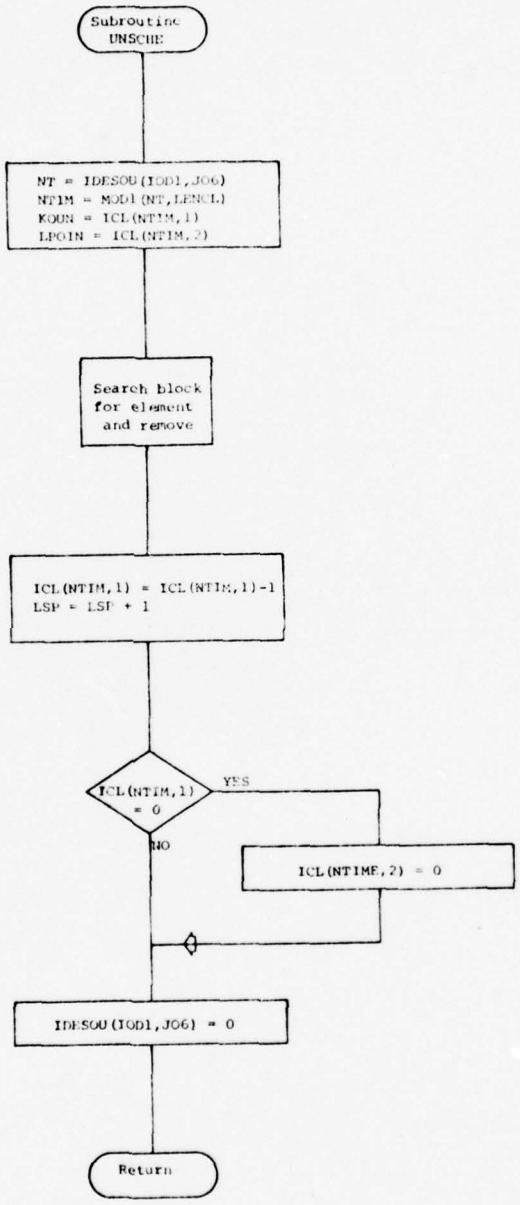


Fig. 9. Subroutine UNSCHE.

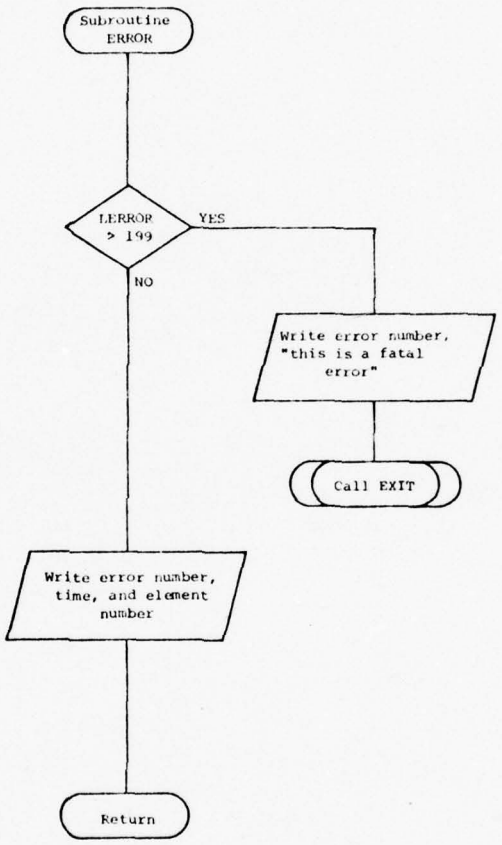


Fig. 10. Subroutine ERROR.

Subroutine IODEST

Subroutine IODEST establishes the index IOD of IDESOU(IOD,JO) for the current element being processed during the simulation. It establishes IOD1 as the element being proeessed, IOD2 as the first input to the element, IOD3 as the second input, and on to as many indices as there are inputs to the element. It is merely a "look-up" routine intended to minimize cross-referencing efforts between IDESOU and IDESCR arrays.

CHAPTER 5

EVESIM ELEMENT MODELS

Many approaches exist to model logic elements. Chapter 2 established that an element level approach was the most appropriate since modeling elements with gates may not present the same operating characteristic as the element itself. It was also established that an assignable or precise timing capability was most desirable in order to achieve great accuracy. In fact, being able to assign delays in terms of both rising signal levels and falling levels provides the most accurate model. *EVESIM has been* designed such that delays can be assigned in terms of rising and falling signal levels. The problem of modeling for EVESIM is a matter of assigning the correct delay times to the element and being able to calculate from these delays when an element's outputs are to change. The models have been split into two distinct types. The first type is for single output non-memory type devices normally called combinational logic gates. The second type is for all other devices which were defined earlier as elements.

Element Delay Time

EVESIM uses precise delay time calculations. Obviously, either efficiency or accuracy would quickly be

lost if element delay times were randomly picked or an attempt was made to run the simulation in real time. It is therefore necessary to establish certain ground rules for determining the element delay times.

It is important that the delay times for the elements have some common parameter; otherwise, the accuracy of the simulation is lost. If all element delays were made to be of the form: $D = n\Delta t$, where D is delay time in real time, n is a positive integer, and Δt is a positive constant, then all delays can be represented by a positive integer n. This has forced all element delays in a given circuit to be relative to each other; and therefore, the accuracy of the simulation has been preserved while not losing efficiency. An EVESIM user would refer to a manufacturer's logic data book and determine approximate values for t_{PLH} and t_{PHL} for all the elements in the circuit to be simulated. It should be noted here that the choice of delay times is not totally random and requires the user to have some intuitive knowledge of the expected operation of the circuit being simulated. Then, from this information, the user should determine a value for Δt which by definition is nothing more than the greatest common divisor of the element delays. All element delay times (n) can then be determined by: $n_e = D_e \div \Delta t$.

As an example, consider a small circuit containing two types of elements with delays of t_{PLH1} = 10 nsec,

t_{PHL1} = 6 nsec, t_{PLH2} = 12 nsec, and t_{PHL2} = 8 nsec. The greatest common divisor of these times is 2 nsecs which is equal to Δt. The values for n are then computed:

$$n_{1LH} = 10 \div 2 = 5 \qquad n_{1HL} = 6 \div 2 = 3$$

$$n_{2HL} = 12 \div 2 = 6 \qquad n_{2HL} = 8 \div 2 = 4.$$

One simulation time is now equivalent to 2 nsec. The maximum length of the internal control clock has been determined and is equal to the maximum n_e which in this case is 6. It should be noted that another set of delays will determine a different Δt. Thus, it is desirable to maximize Δt such that the maximum element delay time n_e is kept as small as possible in order to economize the execution of a simulation.

Subroutine COMLOG

When an event has occurred, an element output has changed. The elements connected to that changed output are each, in turn, processed through an element model. Subroutine COMLOG (Fig. 11) serves as the model for all combintational logic gates. The first action taken is a calculation of the output. Then, a comparison is made to the old output. If a change has occurred, the new output is appropriately scheduled as an event to occur in the future.

There are many techniques available that can be used to determine the logic values. Boolean algebra can be used.

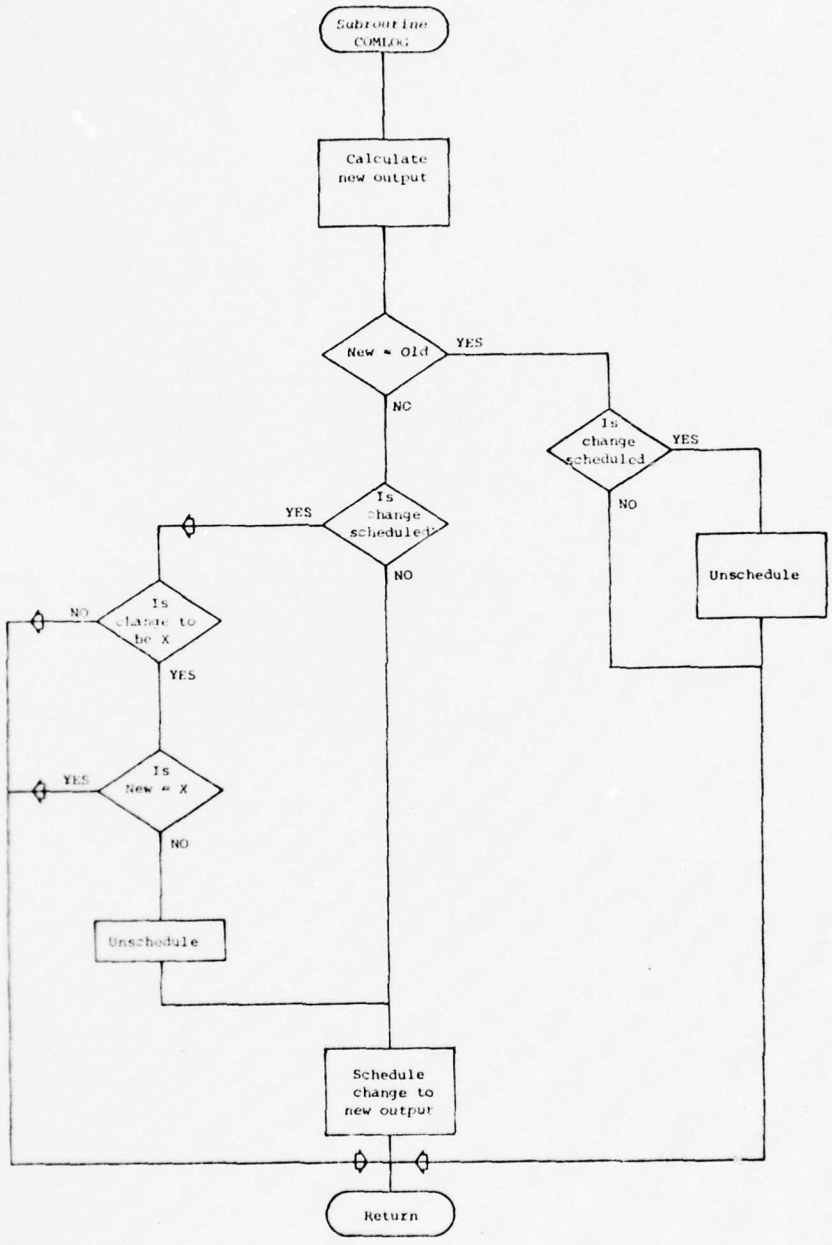


Fig. 11. Subroutine COMLOG.

The available logic functions of an operating system provide a means of calculation. Table look-up techniques can be used or arithmetic expressions can be used for the calculations. The technique used for the calculation is not really relative to the modeling of the element as far as modeling the delay is concerned. Therefore, whatever is easiest for the programmer's system is appropriate. EVESIM uses both table look-up techniques and the available logic functions.

The important part of the modeling scheme is that an event that is to occur is properly scheduled in the future. This scheduling is based on the delay times as calculated above. It is important that noise spikes are passed or rejected as required. Whether a gate will pass or reject a spike is a function of the gate propagation delay times, t_{PLH} and t_{PHL}. The modeling of this is easily accomplished with an event type of simulation program. For example, let a NAND gate have values $t_{PLH} = 11$ and $t_{PHL} = 7$. Obviously, immunity to 1 - to - 0 transitions is greater than 0 - to - 1 transitions since t_{PLH} is greater than t_{PHL}. Let the output of the NAND gate at time 0 be equal to 0. Assume that at time 0, an input to the gate has changed such that the output should change to 1. An event should be scheduled for time $0 + t_{PLH}$ which is equal to time 11. Following the scheme of the flow chart in Fig. 11, any attempt to schedule a t_{PHL} transition prior to time 11 would negate the event scheduled at time 11. The earliest that a valid t_{PHL}

transition would register would be at time = 11 + t_{PHL} = 18. Therefore, the narrowest pulse that could be passed through the NAND gate would be 18 - 11 = 7, which is what is expected. A similar analysis can be done for the 1 - to - 0 noise immunity. The result is a minimum pulse of 11, again which is what is expected. Making use of the rising and falling delay times in conjunction with proper scheduling and unscheduling of events has effectively modeled the combinational logic. The modeling of multiple output devices is accomplished in a similar fashion. The difference, as seen in the next section, is in the complexity of determining when an event should be scheduled.

Subroutine JKM1

The complexity of determining when an event should occur for elements such as a JK master-slave flip-flop results from the fact that the flip-flop has several inputs, it has memory capability, and has more than one output. There are also other parameters for consideration such as minimum pulse width requirements to the clock input and the preset and clear inputs, and the J and K inputs remaining stable while the clock is high. Since the data depicting these parameters is available in the data book, they are also applied against the greatest common divisor and an integer value assigned to represent the parameters for modeling purposes.

The JK flip-flop model, subroutine KJM1 (Fig. 12), is broken into four paths dependent on the three inputs that can possibly change the outputs. Since preset and clear override the clock, they first are tested for an unknown condition. An unknown on preset or clear will automatically force an unknown condition on the output. If neither are unknown, further checks are made and the appropriate branching takes place as indicated in Fig. 12. If the J or K input to the flip-flop was the input change that caused the model to be called, control is returned to the calling subroutine without any further action. It should also be noted that a leading edge clock change requires no action for a master-slave flip-flop and control is returned to the calling routine.

Assuming that a trailing edge clock caused the call of subroutine JKM1, the clock pulse is first checked to insure that it meets minimum width requirements. If not, an error message (error 1) is processed and the outputs scheduled as unknowns. If the clock is valid and preset and clear are both high, new outputs are calculated and scheduled to occur as events in the future.

A single preset change or a clear change are similar in nature and relatively simple to program. The processing of these changes are self explanatory and are indicated on the flow chart on pages 52 and 54. The complexity is greatly increased if both preset and clear change at the

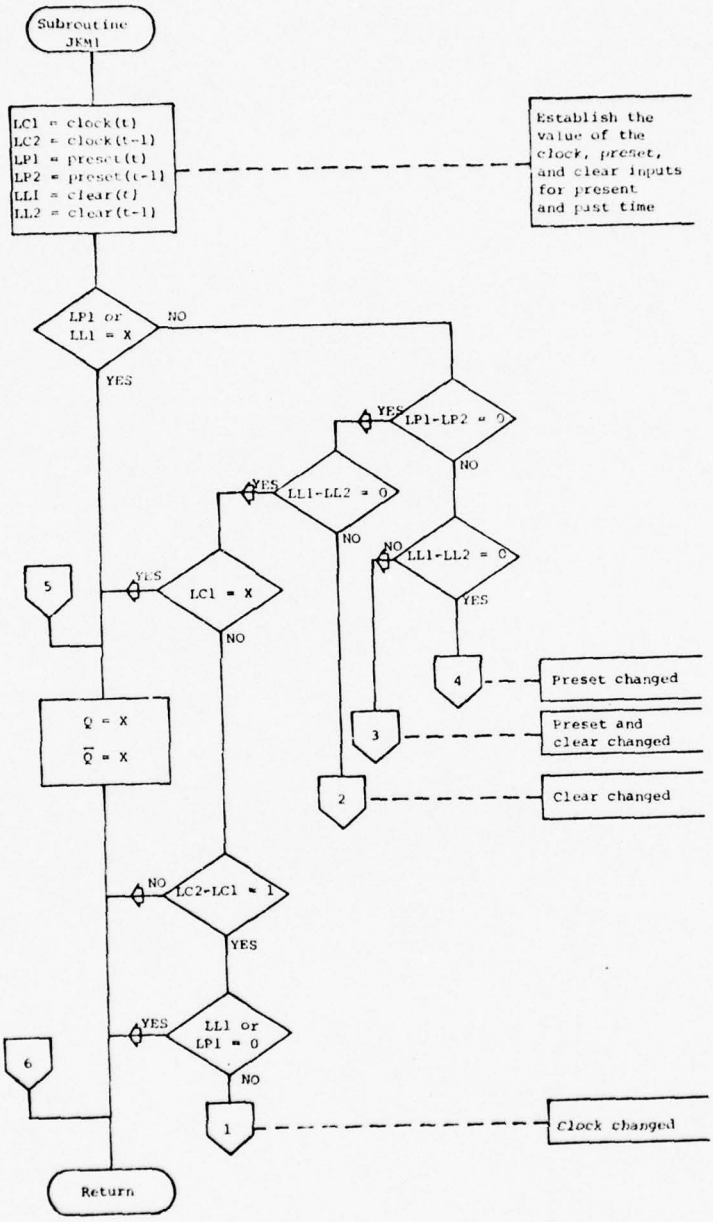


Fig. 12. Subroutine JKM1.

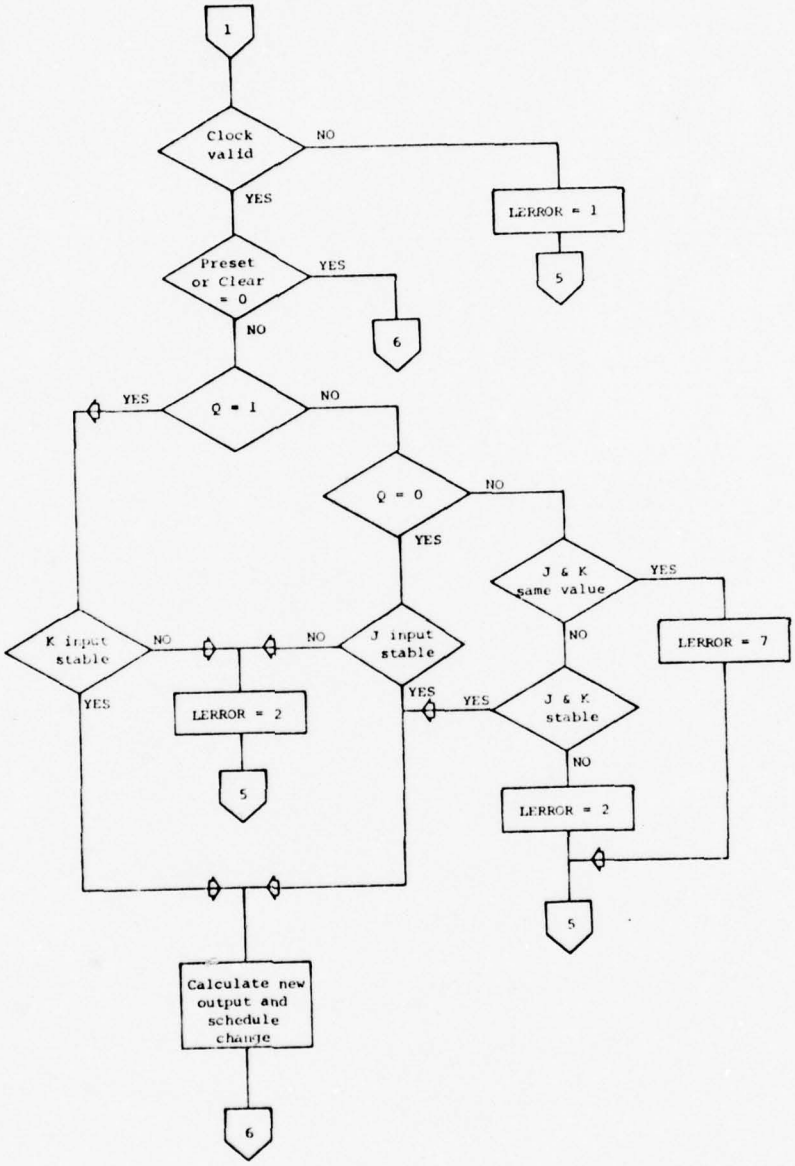


Fig. 12.--Continued Subroutine JKM1.

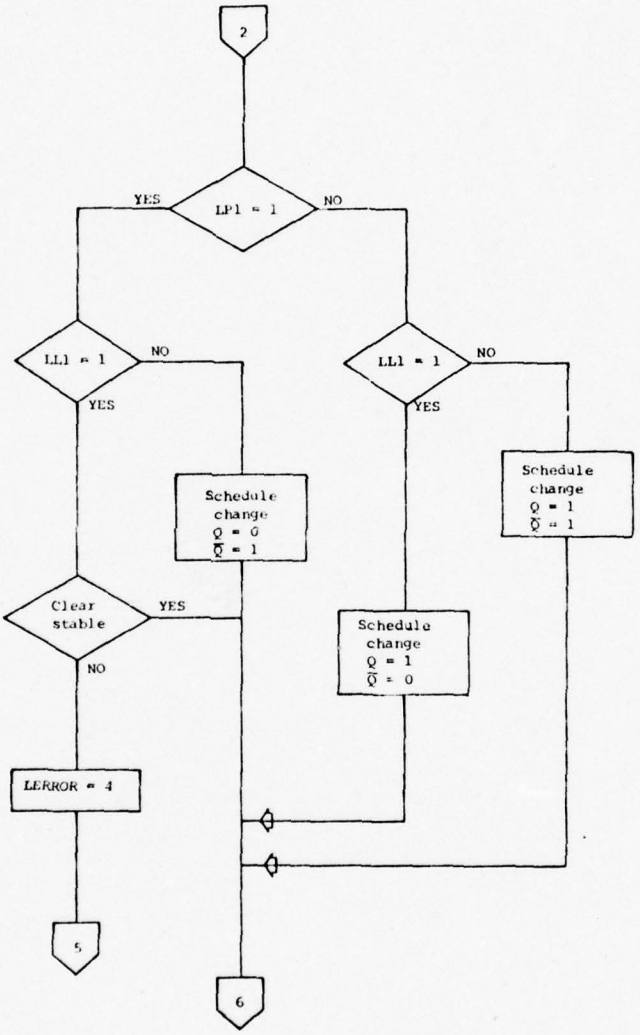


Fig. 12.--Continued Subroutine JKM1.

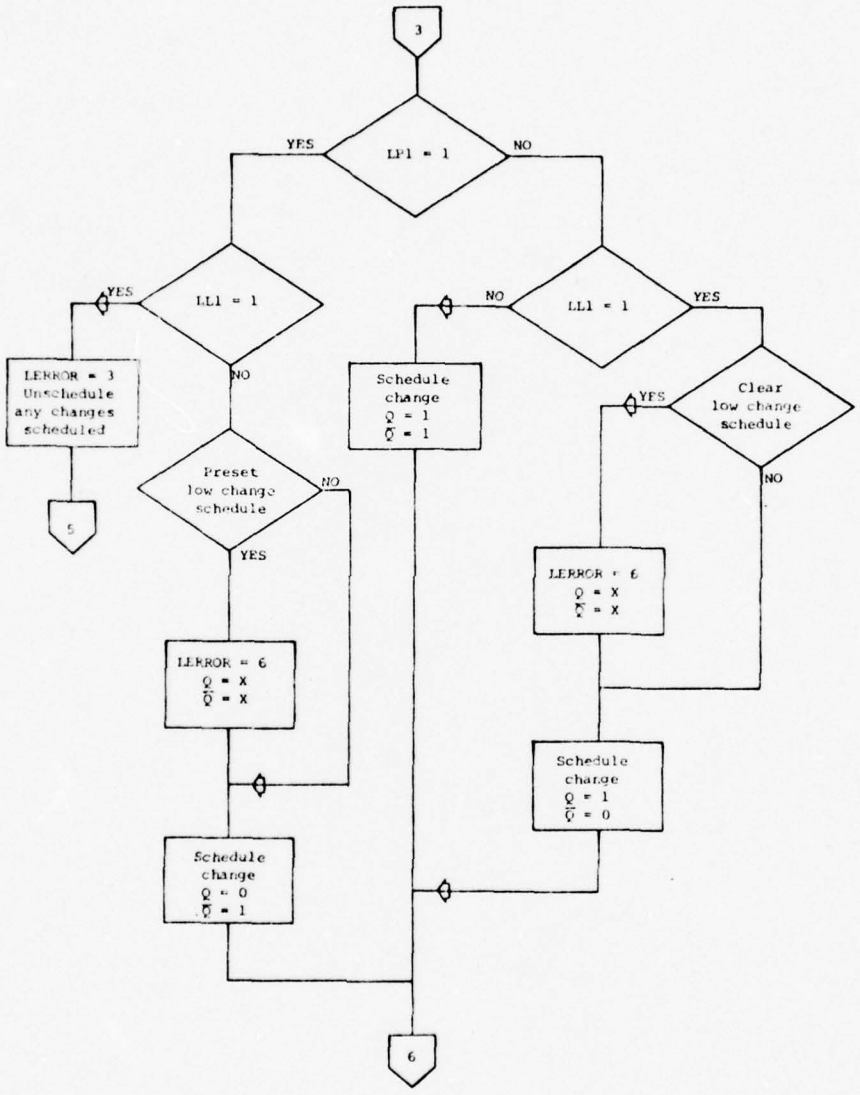


Fig. 12.--Continued Subroutine JKM1.

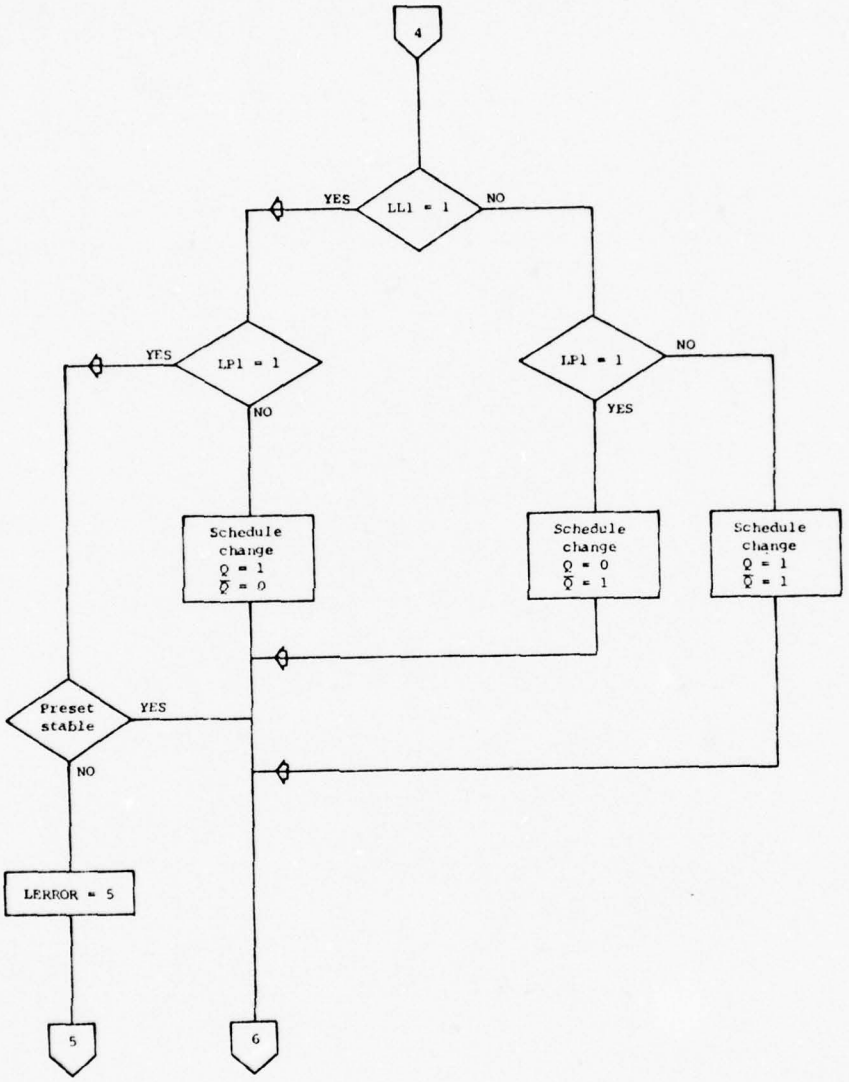


Fig. 12.--Continued Subroutine JKM1.

same time. If they were both low, the outputs are both high. If both preset and clear go high at the same time, it can not be determined what the outputs will be; thus, the outputs are changed to the unknown condition. If preset and clear are opposite in value and they both change, a spike can be generated on the output of the flip-flop. This is designated by error message 6 and the outputs are temporarily changed to an unknown condition until the new values of preset and clear cause the outputs to stabilize.

On the surface, modeling of the JK master-slave flip-flop appears to be a lengthy process. However, by having full understanding of how the element functions, the model can be broken into small processing sections each representing an element reaction to a particular stimulus, thus providing an easily understood and logical approach to solving the modeling problem for a JK flip-flop or any multiple input-multiple output element.

CHAPTER 6

DEMONSTRATION OF THE USE OF EVESIM

A description of EVESIM would not be complete without a demonstration of some circuits being simulated. The first example is strictly a JK master-slave flip-flop run against the simulator all by itself. This was chosen in order to demonstrate the functioning of the flip-flop model discussed in Chapter 4. The remaining two circuits were chosen to demonstrate the interaction of the models. It should be noted that both have critical feedback paths that restrict the frequency of the clock, thus surely demonstrating the need for a level mode simulation. Data used for the simulations were taken from the Texas Instruments TTL Data Book (Components Group, Engineering Staff, 1973).

Demonstration of the JK Master-Slave Flip-Flop

The flip-flop operation has been demonstrated by six different runs of the simulator. The six runs were designed in order that all possible paths through the flip-flop model would be exercised. Figure 13 identifies the input names to the flip-flop and Table 3 contains the data used for the six runs. It should be noted that the output of the flip-flop is given by the Q output being equal to element number 1 and

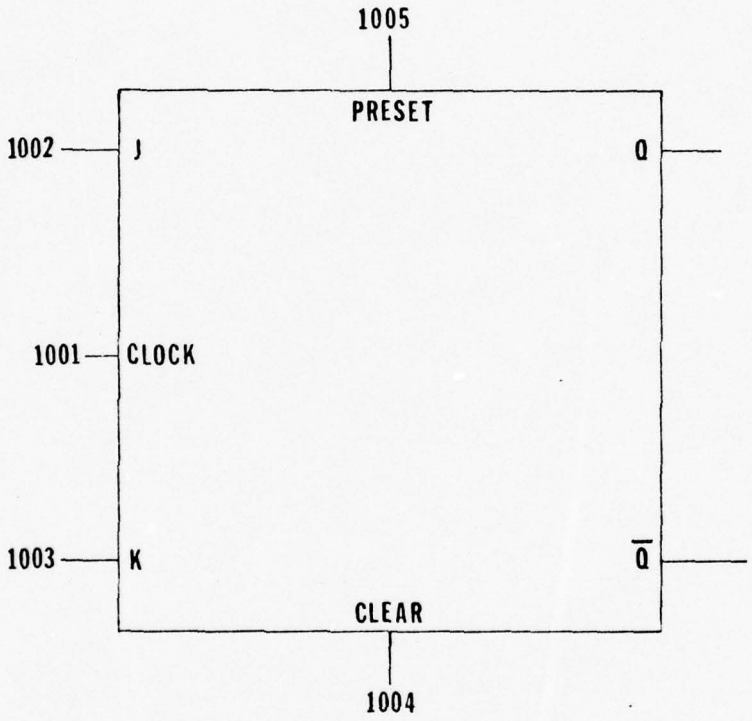


Fig. 13. JK master-slave flip-flop circuit number 3.

Table 3. JK master-slave flip-flop data.

1. Greatest common divisor = Δt = 5 nanoseconds

2. Clocked Input Data

   t_{PLH} = 15 nanoseconds n = 15 ÷ 5 = 3 Simulation times

   t_{PHL} = 25 nanoseconds n = 25 ÷ 5 = 5 Simulation times

   Minimum allowed clock pulse width = 20 nanoseconds
   NPCWID = 20 ÷ 5 = 4 Simulation Times

3. Preset and Clear Input Data

   t_{PLH} = 20 nanoseconds n = 20 ÷ 5 = 4 Simulation times

   t_{PHL} = 30 nanoseconds n = 30 ÷ 5 = 6 Simulation times

   Minimum allowed preset or clear input pulse width = 25 nanoseconds
   NPPWID = 25 ÷ 5 = 5 Simulation Times

the $\overline{Q}$ output by element number 2001. The 2000 indicates the second output of element number 1 or 2001.

Figure 14 contains the first run. The J input high and the K input low at time 2 causes no change at the outputs as is expected. Similarly, with both J and K low, the clock pulse at time 14 causes no change at the output. The clock pulse at time 26 causes the output to change since the K input is high. It should also be pointed out that the model at this time is demonstrating the fact that a JK master-slave flip-flop requires a set up time equal to zero and a hold time equal to zero. In other words, the J and K inputs must be stable while the clock is high in order to guarantee proper operation of the master-slave flip-flop. With both J and K inputs high, the clock pulse at time 38 causes the output to toggle.

Figure 15 is the output of the flip-flop for run number 2. Run 2 is intended to demonstrate the operation of the clear and preset inputs. Notice that at time 1 the clear input goes low, overriding the clock pulse and causing an output change. The preset is similarly demonstrated at time 11. Notice that at time 18 both preset and clear are low causing both outputs to be high. At time 23 the clear input returns to the one level and the outputs align themselves according to clear remaining low.

The remaining four runs made of the JK flip-flop demonstrate the possible error conditions that can cause an

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	0	0	1	1	0
1	0	1	0	1	0	1	1	1
2	0	1	1	1	0	1	1	2
3	0	1	1	1	0	1	1	3
4	0	1	1	1	0	1	1	4
5	0	1	1	1	0	1	1	5
6	0	1	0	0	0	1	1	6
7	0	1	0	0	0	1	1	7
8	0	1	0	0	0	1	1	8
9	0	1	0	0	0	1	1	9
10	0	1	0	0	0	1	1	10
11	0	1	0	0	0	1	1	11
12	0	1	0	0	0	1	1	12
13	0	1	0	0	0	1	1	13
14	0	1	1	0	0	1	1	14
15	0	1	1	0	0	1	1	15
16	0	1	1	0	0	1	1	16
17	0	1	1	0	0	1	1	17
18	0	1	0	0	0	1	1	18
19	0	1	0	0	0	1	1	19
20	0	1	0	0	0	1	1	20
21	0	1	0	0	0	1	1	21
22	0	1	0	0	0	1	1	22
23	0	1	0	0	0	1	1	23
24	0	1	0	0	0	1	1	24
25	0	1	0	0	0	1	1	25
26	0	1	1	0	1	1	1	26
27	0	1	1	0	1	1	1	27
28	0	1	1	0	1	1	1	28
29	0	1	1	0	1	1	1	29
30	0	1	0	0	0	1	1	30
31	0	1	0	0	0	1	1	31
32	0	1	0	0	0	1	1	32
33	1	1	0	0	0	1	1	33
34	1	1	0	1	1	1	1	34
35	1	0	0	1	1	1	1	35
36	1	0	0	1	1	1	1	36
37	1	0	0	1	1	1	1	37
38	1	0	1	1	1	1	1	38
39	1	0	1	1	1	1	1	39
40	1	0	1	1	1	1	1	40
41	1	0	1	1	1	1	1	41
42	1	0	0	1	1	1	1	42
43	1	0	0	1	1	1	1	43
44	1	0	0	1	1	1	1	44
45	1	1	0	1	1	1	1	45
46	1	1	0	1	1	1	1	46
47	0	1	0	1	1	1	1	47
48	0	1	0	1	1	1	1	48
49	0	1	0	1	1	1	1	49
50	0	1	0	1	1	1	1	50

Fig. 14. Circuit 3 run 1.

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	1	1	1	1	0
1	0	1	0	1	1	0	1	1
2	0	1	0	1	1	0	1	2
3	0	1	1	1	1	0	1	3
4	0	1	1	1	1	0	1	4
5	1	1	1	1	1	0	1	5
6	1	1	1	1	1	1	1	6
7	1	0	0	1	1	1	1	7
8	1	0	0	1	1	1	1	8
9	1	0	0	1	1	1	1	9
10	1	0	0	1	1	1	1	10
11	1	0	0	1	1	1	0	11
12	1	0	0	1	1	1	0	12
13	1	0	0	1	1	1	0	13
14	1	0	0	1	1	1	0	14
15	1	1	1	1	1	1	0	15
16	1	1	1	1	1	1	1	16
17	0	1	1	1	1	1	1	17
18	0	1	1	1	1	0	0	18
19	0	1	0	1	1	0	0	19
20	0	1	0	1	1	0	0	20
21	0	1	0	1	1	0	0	21
22	1	1	0	1	1	0	0	22
23	1	1	0	1	1	0	1	23
24	1	1	0	1	1	0	1	24
25	1	1	0	1	1	0	1	25
26	1	1	0	1	1	0	1	26
27	1	1	1	1	1	0	1	27
28	1	1	1	1	1	0	1	28
29	1	0	1	1	1	1	1	29
30	1	0	1	1	1	1	1	30
31	1	0	0	1	1	1	1	31
32	1	0	0	1	1	1	1	32
33	1	0	0	1	1	1	1	33
34	1	0	0	1	1	1	1	34
35	1	0	0	1	1	1	1	35
36	1	0	0	1	1	1	1	36
37	1	0	0	1	1	1	1	37
38	1	0	0	1	1	1	1	38
39	1	0	1	1	1	1	1	39
40	1	0	1	1	1	1	1	40
41	1	0	1	1	1	1	1	41
42	1	0	1	1	1	1	1	42
43	1	0	0	1	1	1	1	43
44	1	0	0	1	1	1	1	44
45	1	0	0	1	1	1	1	45
46	1	1	0	1	1	1	1	46
47	1	1	0	1	1	1	1	47
48	0	1	0	1	1	1	1	48
49	0	1	0	1	1	1	1	49
50	0	1	0	1	1	1	1	50

Fig. 15. Circuit 3 run 2.

unknown condition to appear on the output. The error messages are clearly printed with the error number, time of occurrence, and the element involved. Figure 16 demonstrates two detected errors. The first is error 1, which is a clock pulse being too narrow. This error occurred at time 2. Error 7, though not necessarily an error in the true sense of the word error, points out that the outputs are unknown, and since the J and K inputs are of the same value, a new set of outputs cannot be determined. Notice that at time 23, the J and K inputs have different values, and therefore a new output set can be determined. The worst case is assumed; that is, that for the Q output the unknown was equal to 1 and had to change to 0, and that the opposite case exists for $\overline{Q}$. The remaining point of interest on Fig. 16 is that at time 35, a clock pulse occurs with the Q output low and the K input low, and because of this time situation, the K input changing midway through the clock pulse has no effect on the output as should be expected.

At time 5 on Figure 17, error message is generated. This error results from the fact that the K input to the flip-flop does not meet the hold time required, or the K input is unstable while the clock is high. This results in the unknown condition being generated on the outputs. The next interesting point is the spike generated on the Q output at time 20. This is a result of the clear input having a low value and the preset input going low temporarily and

CIRCUIT NUMBER = 3

**********ERROR MESSAGES**********

ERROR = 1 TIME = 3 ELEMENT = 1

ERROR = 7 TIME = 15 ELEMENT = 1

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	2
3	0	1	0	1	1	1	1	3
4	X	X	0	1	1	1	1	4
5	X	X	0	1	1	1	1	5
6	X	X	0	1	1	1	1	6
7	X	X	0	1	1	1	1	7
8	X	X	0	1	1	1	1	8
9	X	X	0	1	1	1	1	9
10	X	X	0	1	1	1	1	10
11	X	X	1	1	1	1	1	11
12	X	X	1	1	1	1	1	12
13	X	X	1	1	1	1	1	13
14	X	X	1	1	1	1	1	14
15	X	X	0	1	1	1	1	15
16	X	X	0	0	1	1	1	16
17	X	X	0	0	1	1	1	17
18	X	X	0	0	1	1	1	18
19	X	X	0	0	1	1	1	19
20	X	X	0	0	1	1	1	20
21	X	X	0	0	1	1	1	21
22	X	X	0	0	1	1	1	22
23	X	X	1	0	1	1	1	23
24	X	X	1	0	1	1	1	24
25	X	X	1	0	1	1	1	25
26	X	X	1	0	1	1	1	26
27	X	X	0	0	1	1	1	27
28	X	X	0	0	1	1	1	28
29	X	X	0	0	1	1	1	29
30	1	X	0	0	1	1	1	30
31	1	X	0	0	1	1	1	31
32	1	0	0	0	1	1	1	32
33	1	0	0	0	1	1	1	33
34	1	0	0	0	1	1	1	34
35	1	0	1	0	1	1	1	35
36	1	0	1	0	1	1	1	36
37	1	0	1	0	0	1	1	37
38	1	0	1	0	0	1	1	38
39	1	0	0	0	0	1	1	39
40	1	0	0	0	0	1	1	40
41	1	0	0	0	0	1	1	41
42	1	0	0	0	0	1	1	42
43	1	0	0	0	0	1	1	43
44	1	0	0	0	0	1	1	44
45	1	0	0	0	0	1	1	45

Fig. 16. Circuit 3 run 3.

CIRCUIT NUMBER = 3

**********ERROR MESSAGES**********

ERROR = 2 TIME = 5 ELEMENT = 1

ERROR = 4 TIME = 32 ELEMENT = 1

ERROR = 4 TIME = 40 ELEMENT = 1

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	2
3	0	1	1	1	0	1	1	3
4	0	1	1	1	0	1	1	4
5	0	1	0	1	0	1	1	5
6	X	X	0	1	0	1	1	6
7	X	X	0	1	0	1	1	7
8	X	X	0	1	0	1	1	8
9	X	X	0	1	0	0	1	9
10	X	X	0	1	0	0	1	10
11	X	X	0	1	0	0	1	11
12	X	X	0	1	0	0	1	12
13	1	X	1	1	0	0	1	13
14	1	X	1	1	0	0	1	14
15	1	0	1	1	0	0	1	15
16	1	0	1	1	0	0	1	16
17	1	0	0	1	0	0	0	17
18	1	0	0	1	0	0	0	18
19	1	0	0	1	0	0	0	19
20	1	1	0	1	0	0	1	20
21	1	0	0	1	0	0	1	21
22	1	0	0	1	0	0	0	22
23	1	0	0	1	0	0	0	23
24	1	0	0	1	0	0	0	24
25	1	0	1	1	0	0	0	25
26	1	1	1	1	0	0	0	26
27	1	1	1	1	0	0	1	27
28	1	1	1	1	0	0	1	28
29	1	1	0	1	0	0	1	29
30	1	1	0	1	0	0	1	30
31	1	1	0	1	0	0	1	31
32	1	1	0	1	0	1	1	32
33	X	X	0	1	0	1	1	33
34	X	X	0	1	0	1	1	34
35	X	X	0	1	0	1	1	35
36	X	X	0	1	0	0	1	36
37	X	X	1	1	0	0	1	37
38	X	X	1	1	0	0	1	38
39	X	X	1	1	0	0	1	39
40	1	X	1	1	0	1	1	40
41	X	X	0	1	0	1	1	41
42	X	X	0	1	0	1	1	42
43	X	X	0	1	0	1	1	43
44	X	X	0	1	0	1	1	44
45	X	X	0	1	0	1	1	45

Fig. 17. Circuit 3 run 4.

then returning to the high value. Certainly, this operation can be expected. Error message 4, clear input pulse is too narrow, occurs at time 32 because the clear input was not low a sufficient amount of time after the preset input returned to one. Error 4 occurs again at time 40.

Figure 18 is the output for run 5. There are 3 errors demonstrated by this run. Error 3 occurs when both preset and clear inputs change to a high value at the same time. Error 5 appears when a spike occurs on the preset input, in other words, preset was low an insufficient amount of time to cause a predicted change on the output. Error 7 occurs again for the same reason as stated earlier.

Error number 6 is demonstrated in Fig. 19, which is run number 6 of the flip-flop. Error 6 is generated because preset and clear are changing at the same time before either preset or clear was able to cause a change to the output of the flip-flop. Notice that this situation is self-correcting if either clear or preset remains low for a sufficient amount of time.

In the following sections, the JK master-slave flip-flop model is used in conjunction with the combinational logic model to simulate realizable circuits. Having fully established the operation of a single element, it is now time to demonstrate the simulation of a complete circuit.

CIRCUIT NUMBER = 3

**********ERROR MESSAGES**********

ERROR = 3 TIME = 9 ELEMENT = 1

ERROR = 5 TIME = 24 ELEMENT = 1

ERROR = 7 TIME = 29 ELEMENT = 1

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	1	1	1	1	0
1	0	1	1	1	1	0	1	1
2	0	1	1	1	1	0	1	2
3	0	1	1	1	1	0	0	3
4	0	1	1	1	1	0	0	4
5	0	1	0	1	1	0	0	5
6	0	1	0	1	1	0	0	6
7	1	1	0	1	1	0	0	7
8	1	1	0	1	1	0	0	8
9	1	1	0	1	1	1	1	9
10	X	X	0	1	1	1	1	10
11	X	X	0	1	1	1	1	11
12	X	X	0	1	1	0	1	12
13	X	X	1	1	1	0	1	13
14	X	X	1	1	1	0	1	14
15	X	X	1	1	1	0	1	15
16	1	X	1	1	1	0	1	16
17	1	X	0	1	1	1	1	17
18	1	0	0	1	1	1	1	18
19	1	0	0	1	1	1	1	19
20	1	0	0	1	1	1	1	20
21	1	0	0	1	1	0	1	21
22	1	0	0	1	1	0	1	22
23	1	0	0	1	1	1	0	23
24	1	0	0	1	1	1	1	24
25	X	X	1	1	1	1	1	25
26	X	X	1	1	1	1	1	26
27	X	X	1	1	1	1	1	27
28	X	X	1	1	1	1	1	28
29	X	X	0	1	1	1	1	29
30	X	X	0	1	1	0	1	30
31	X	X	0	1	1	0	1	31
32	X	X	0	1	1	0	1	32
33	X	X	0	1	1	0	1	33
34	1	X	0	1	1	0	1	34
35	1	X	0	1	1	0	1	35
36	1	0	0	1	1	0	1	36
37	1	0	1	1	1	1	1	37
38	1	0	1	1	1	1	1	38
39	1	0	1	1	1	1	1	39
40	1	0	1	1	1	1	1	40
41	1	0	0	1	1	1	1	41
42	1	0	0	1	1	1	1	42
43	1	0	0	1	1	1	1	43
44	1	1	0	1	1	1	1	44
45	1	1	0	1	1	1	1	45

Fig. 18. Circuit 3 run 5.

CIRCUIT NUMBER = 3

**********ERROR MESSAGES**********

ERROR = 6 TIME = 13 ELEMENT = 1

CIRCUIT NUMBER = 3

TIME	ELEMENT NUMBER							TIME
	2001	1	1001	1002	1003	1004	1005	
0	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	2
3	0	1	1	1	1	1	1	3
4	0	1	1	1	1	1	1	4
5	0	1	0	1	1	1	1	5
6	0	1	0	1	1	1	1	6
7	0	1	0	1	1	1	1	7
8	1	1	0	1	1	1	1	8
9	1	1	0	1	1	1	1	9
10	1	0	0	1	1	1	1	10
11	1	0	0	1	1	1	0	11
12	1	0	0	1	1	1	0	12
13	X	X	1	1	1	0	1	13
14	X	X	1	1	1	0	1	14
15	X	X	1	1	1	0	1	15
16	X	X	1	1	1	0	1	16
17	1	X	0	1	1	0	1	17
18	1	X	0	1	1	0	1	18
19	1	0	0	1	1	0	1	19
20	1	0	0	1	1	1	1	20
21	1	0	0	1	1	1	1	21
22	1	0	0	1	1	1	1	22
23	1	0	0	1	1	1	1	23
24	1	0	0	1	1	1	1	24
25	1	0	1	1	1	1	1	25
26	1	0	1	1	1	1	1	26
27	1	0	1	1	1	1	1	27
28	1	0	1	1	1	1	1	28
29	1	0	0	1	1	1	1	29
30	1	0	0	1	1	1	1	30
31	1	0	0	1	1	1	1	31
32	1	1	0	1	1	1	1	32
33	1	1	0	1	1	1	1	33
34	0	1	0	1	1	1	1	34
35	0	1	0	1	1	1	1	35
36	0	1	0	1	1	1	1	36
37	0	1	1	1	1	1	1	37
38	0	1	1	1	1	1	1	38
39	0	1	1	1	1	1	1	39
40	0	1	1	1	1	1	1	40
41	0	1	0	1	1	1	1	41
42	0	1	0	1	1	1	1	42
43	0	1	0	1	1	1	1	43
44	1	1	0	1	1	1	1	44
45	1	1	0	1	1	1	1	45

Fig. 19. Circuit 3 run 6.

Demonstration of an Odd-Even Detector Circuit

Figure 20 is a diagram of an odd-even detector. It is a sequential circuit with two pulse inputs, A and B, and a single level output z. Following a pulse on line B, the output z is to be 1 provided there has been an even number of pulses on line A since the previous pulse on line B. Otherwise, a B pulse will reset the output to 0. The output will not change except on the arrival of a B pulse. The intent of the example is to determine whether the circuit presented is or is not the best design, and in turn to demonstrate the usefulness of a level mode simulator like EVESIM.

The element delay parameters used for the first run are in Table 4. The circuit diagram is contained in Fig. 20. Under the conditions of Table 4, the circuit functions as it should and the output of the simulator, Fig. 21, verifies that it does function correctly. However, the circuit does not function correctly if the pulse on line B is made wider and gate number 4 is made faster. This situation causes flip-flop 2 to change value very fast; thus, the inputs to flip-flop 1 will be unstable while the clock pulse to flip-flop 1 is still high. The parameters for gate 4 are changed to t_{PLH} = 10 nanoseconds and t_{PHL} = 5 nanoseconds and pulse B is widened to 45 nanoseconds. With

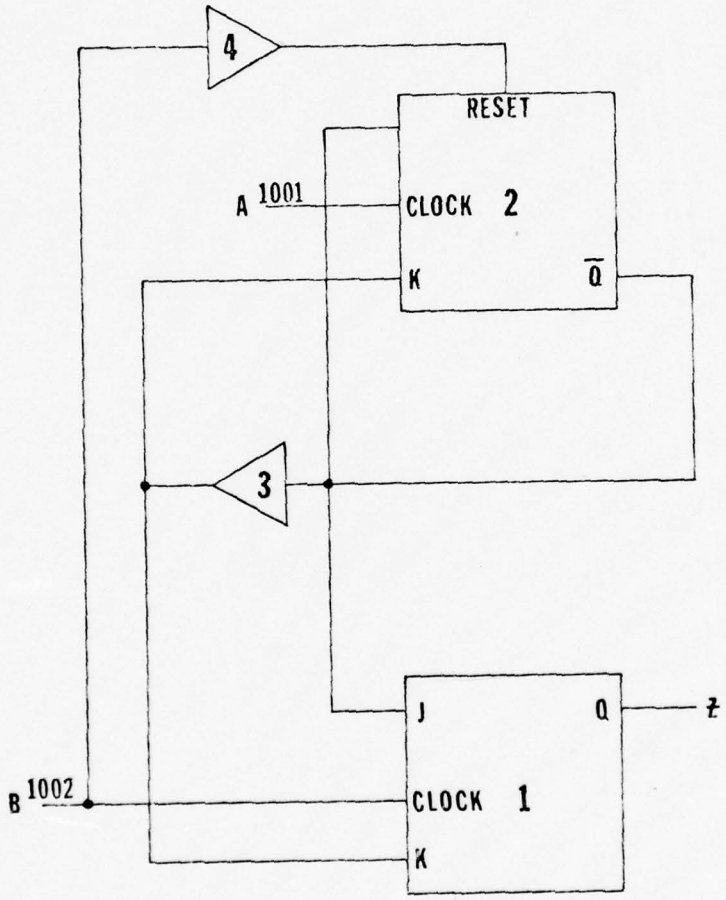


Fig. 20. Odd-even detector circuit no. 5.

Table 4. Odd-even detector circuit parameters.

1. Greatest Common Divisor = Δt = 5 nanoseconds

2. Flip-Flop Data
   a. Clocked input data
      t_{PLH} = 25 nanoseconds n = 25 ÷ 5 = 5 Simulation Times

      t_{PHL} = 40 nanoseconds n = 40 ÷ 5 = 8 Simulation Times

      Minimum allowed clock pulse width = 20 nanoseconds
      NPCWID = 20 ÷ 5 = 4 Simulation Times

   b. Reset Input Data
      t_{PLH} = 15 nanoseconds n = 15 ÷ 5 = 3 Simulation Times

      t_{PHL} = 25 nanoseconds n = 25 ÷ 5 = 5 Simulation Times

      Minimum allowed reset pulse width = 25 nanoseconds
      NPPWID = 25 ÷ 5 = 5 Simulation Times

3. Inverter Data
      t_{PLH} = 15 nanoseconds n = 15 ÷ 5 = 3 Simulation Times

      t_{PHL} = 10 nanoseconds n = 10 ÷ 5 = 2 Simulation Times

CIRCUIT NUMBER = 5

TIME	ELEMENT NUMBER						TIME
	1	2002	3	4	1001	1002	
0	0	1	0	1	0	0	0
1	0	1	0	1	1	0	1
2	0	1	0	1	1	0	2
3	0	1	0	1	1	0	3
4	0	1	0	1	1	0	4
5	0	1	0	1	0	0	5
6	0	1	0	1	0	0	6
7	0	1	0	1	0	0	7
8	0	1	0	1	0	0	8
9	0	1	0	1	0	0	9
10	0	1	0	1	0	0	10
11	0	1	0	1	0	0	11
12	0	1	0	1	0	0	12
13	0	0	0	1	0	0	13
14	0	0	0	1	0	0	14
15	0	0	0	1	0	0	15
16	0	0	1	1	0	0	16
17	0	0	1	1	0	0	17
18	0	0	1	1	1	0	18
19	0	0	1	1	1	0	19
20	0	0	1	1	1	0	20
21	0	0	1	1	1	0	21
22	0	0	1	1	0	0	22
23	0	0	1	1	0	0	23
24	0	0	1	1	0	0	24
25	0	0	1	1	0	0	25
26	0	0	1	1	0	0	26
27	0	1	1	1	0	0	27
28	0	1	1	1	0	0	28
29	0	1	0	1	0	0	29
30	0	1	0	1	0	0	30
31	0	1	0	1	0	1	31
32	0	1	0	1	0	1	32
33	0	1	0	0	0	1	33
34	0	1	0	0	0	1	34
35	0	1	0	0	0	0	35
36	0	1	0	0	0	0	36
37	0	1	0	0	0	0	37
38	0	0	0	1	0	0	38
39	0	0	0	1	0	0	39
40	1	0	0	1	0	0	40
41	1	0	1	1	0	0	41
42	1	0	1	1	0	0	42
43	1	0	1	1	0	0	43
44	1	0	1	1	0	0	44
45	1	0	1	1	0	0	45

Fig. 21. Circuit 5 run 1.

these changes, which are not unreasonable, the circuit no longer functions correctly as indicated in Fig. 22.

Demonstration of an Up-Down Gray Code Counter

The final circuit to be demonstrated is an up-down self-correcting gray code counter. The circuit has three input signals, a control signal X, a control signal K, and a clock input. When X = 1, the counter counts up, and when X = 0, the counter counts down. The K input controls when the counter shall be allowed to count. K = 0 does not allow counting, and K = 1 allows counting to occur. A circuit diagram of the counter is presented in Fig. 23. Typical values for delays were assigned as indicated in Table 5. Two runs were made. The first indicates proper functioning and the second demonstrates a fatal user error.

Figure 24 contains the results of the simulation of the counter as it performs all its required functions. The circuit was driven with a clock pulse equal to approximately 8 mHz with a pulse duration of 36 nsec. Attempts to drive the circuit with a faster clock failed, thus indicating that this circuit's maximum counting frequency would be about 8.0 mHz. Assuming worst case delays and adding along the longest propagation paths, Gate 10 (22 nsecs) + Gate 11 (40 nsecs) + Gate 7 (22 nsecs) + Gate 8 (22 nsecs) + Gate 12 (25 nsecs)= 131 nsecs. This would result in a predicted

CIRCUIT NUMBER = 5

**********ERROR MESSAGES**********

ERROR = 2 TIME = 40 ELEMENT = 1

CIRCUIT NUMBER = 5

TIME	ELEMENT NUMBER						TIME
	1	2002	3	4	1001	1002	
0	0	1	0	1	0	0	0
1	0	1	0	1	1	0	1
2	0	1	0	1	1	0	2
3	0	1	0	1	1	0	3
4	0	1	0	1	1	0	4
5	0	1	0	1	0	0	5
6	0	1	0	1	0	0	6
7	0	1	0	1	0	0	7
8	0	1	0	1	0	0	8
9	0	1	0	1	0	0	9
10	0	1	0	1	0	0	10
11	0	1	0	1	0	0	11
12	0	1	0	1	0	0	12
13	0	0	0	1	0	0	13
14	0	0	0	1	0	0	14
15	0	0	0	1	0	0	15
16	0	0	1	1	0	0	16
17	0	0	1	1	0	0	17
18	0	0	1	1	1	0	18
19	0	0	1	1	1	0	19
20	0	0	1	1	1	0	20
21	0	0	1	1	1	0	21
22	0	0	1	1	0	0	22
23	0	0	1	1	0	0	23
24	0	0	1	1	0	0	24
25	0	0	1	1	0	0	25
26	0	0	1	1	0	0	26
27	0	1	1	1	0	0	27
28	0	1	1	1	0	0	28
29	0	1	0	1	0	0	29
30	0	1	0	1	0	0	30
31	0	1	0	1	0	1	31
32	0	1	0	0	0	1	32
33	0	1	0	0	0	1	33
34	0	1	0	0	0	1	34
35	0	1	0	0	0	1	35
36	0	1	0	0	0	1	36
37	0	0	0	0	0	1	37
38	0	0	0	0	0	1	38
39	0	0	0	0	0	1	39
40	0	0	1	0	0	0	40
41	X	0	1	0	0	0	41
42	X	0	1	1	0	0	42
43	X	0	1	1	0	0	43
44	X	0	1	1	0	0	44
45	X	0	1	1	0	0	45

Fig. 22. Circuit 5 run 2.

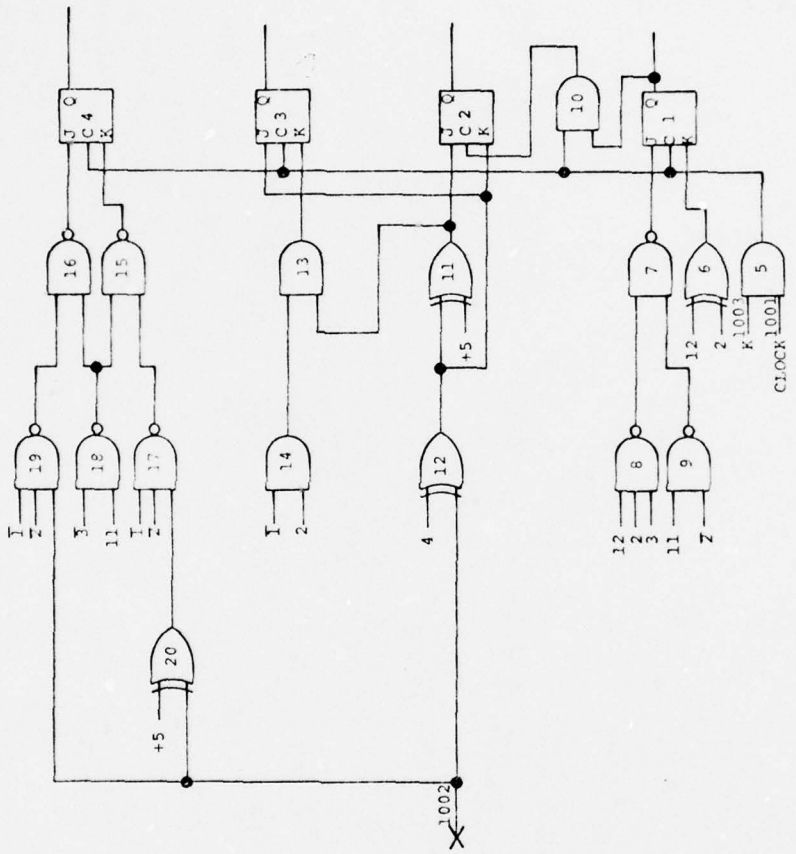


Fig. 23. Up-down self-correcting gray code counter.

Table 5. Up-down gray code counter circuit parameters.

Gate	$n(t_{PLH})$	$n(t_{PHL})$
JK Flip-flop	3 (18 NSEC)	5 (36 NSEC)
NAND	2 (12 NSEC)	1 (6 NSEC)
XOR	3 (18 NSEC)	2 (12 NSEC)

Greatest Common Divisor = 6.

CIRCUIT NUMBER = 4

TIME	ELEMENT NUMBER							TIME
	1	2	3	4	5	10	1001	
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	2
3	0	0	0	0	1	0	1	3
4	0	0	0	0	1	0	1	4
5	0	0	0	0	1	0	1	5
6	0	0	0	0	1	0	1	6
7	0	0	0	0	1	0	0	7
8	0	0	0	0	0	0	0	8
9	0	0	0	0	0	0	0	9
10	0	0	0	0	0	0	0	10
11	0	0	1	1	0	0	0	11
12	0	0	1	1	0	0	0	12
13	0	0	1	1	0	0	0	13
14	0	0	1	1	0	0	0	14
15	0	0	1	1	0	0	0	15
16	0	0	1	1	0	0	0	16
17	0	0	1	1	0	0	0	17
18	0	0	1	1	0	0	0	18
19	0	0	1	1	0	0	0	19
20	0	0	1	1	0	0	0	20
21	0	0	1	1	0	0	0	21
22	0	0	1	1	0	0	1	22
23	0	0	1	1	0	0	1	23
24	0	0	1	1	1	0	1	24
25	0	0	1	1	1	0	1	25
26	0	0	1	1	1	0	1	26
27	0	0	1	1	1	0	1	27
28	0	0	1	1	1	0	0	28
29	0	0	1	1	0	0	0	29
30	0	0	1	1	0	0	0	30
31	0	0	1	1	0	0	0	31
32	1	0	1	1	0	0	0	32
33	1	0	1	1	0	0	0	33
34	1	0	1	1	0	0	0	34
35	1	0	1	1	0	0	0	35
36	1	0	1	1	0	0	0	36
37	1	0	1	1	0	0	0	37
38	1	0	1	1	0	0	0	38
39	1	0	1	1	0	0	0	39
40	1	0	1	1	0	0	0	40
41	1	0	1	1	0	0	0	41
42	1	0	1	1	0	0	0	42
43	1	0	1	1	0	0	1	43
44	1	0	1	1	0	0	1	44
45	1	0	1	1	1	0	1	45
46	1	0	1	1	1	0	1	46
47	1	0	1	1	1	1	1	47
48	1	0	1	1	1	1	1	48
49	1	0	1	1	1	1	0	49

Fig. 24. Circuit 4 run 1.

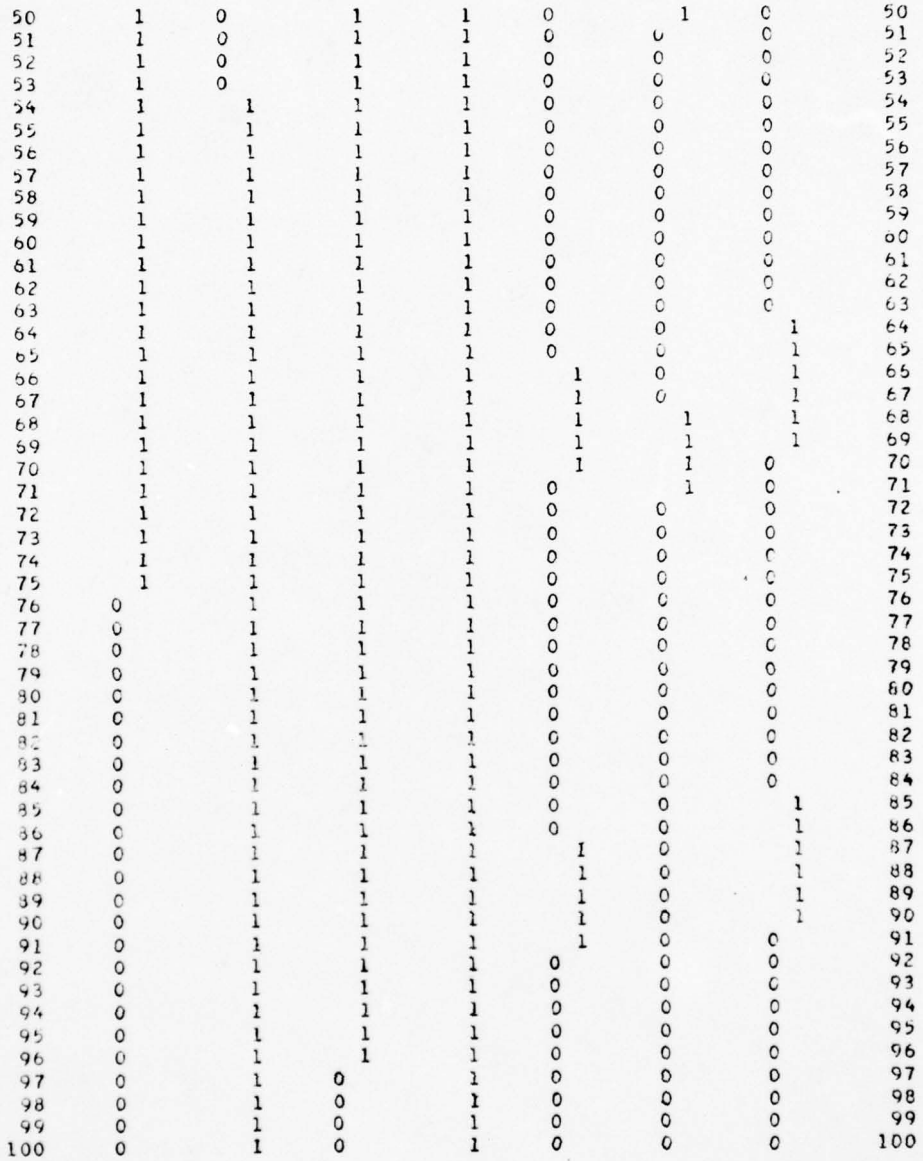


Fig. 24.--Continued Circuit 4 run 1.

101	0	1	0	1	0	0	0		101
102	0	1	0	1	0	0	0		102
103	0	1	0	1	0	0	0		103
104	0	1	0	1	0	0	0		104
105	0	1	0	1	0	0	0		105
106	0	1	0	1	0	0		1	106
107	0	1	0	1	0	0		1	107
108	0	1	0	1	1	0		1	108
109	0	1	0	1	1	0		1	109
110	0	1	0	1	1	0		1	110
111	0	1	0	1	1	0		1	111
112	0	1	0	1	1	0	0		112
113	0	1	0	1	0	0	0		113
114	0	1	0	1	0	0	0		114
115	0	1	0	1	0	0	0		115
116	0	1	0	1	0	0	0		116
117	0	1	0	1	0	0	0		117
118	0	1	0	0	0	0	0		118
119	0	1	0	0	0	0	0		119
120	0	1	0	0	0	0	0		120
121	0	1	0	0	0	0	0		121
122	0	1	0	0	0	0	0		122
123	0	1	0	0	0	0	0		123
124	0	1	0	0	0	0	0		124
125	0	1	0	0	0	0	0		125
126	0	1	0	0	0	0	0		126
127	0	1	0	0	0	0		1	127
128	0	1	0	0	0	0		1	128
129	0	1	0	0	1	0		1	129
130	0	1	0	0	1	0		1	130
131	0	1	0	0	1	0		1	131
132	0	1	0	0	1	0		1	132
133	0	1	0	0	1	0	0		133
134	0	1	0	0	0	0	0		134
135	0	1	0	0	0	0	0		135
136	0	1	0	0	0	0	0		136
137	0	1	1	0	0	0	0		137
138	0	1	1	0	0	0	0		138
139	0	1	1	0	0	0	0		139
140	0	1	1	0	0	0	0		140
141	0	1	1	0	0	0	0		141
142	0	1	1	0	0	0	0		142
143	0	1	1	0	0	0	0		143
144	0	1	1	0	0	0	0		144
145	0	1	1	0	0	0	0		145
146	0	1	1	0	0	0	0		146
147	0	1	1	0	0	0	0		147
148	0	1	1	0	0	0		1	148
149	0	1	1	0	0	0		1	149
150	0	1	1	0	1	0		1	150
151	0	1	1	0	1	0		1	151

Fig. 24.--Continued Circuit 4 run 1.

152	0	1	1	0	1	0	1	152
153	0	1	1	0	1	0	1	153
154	0	1	1	0	1	0	0	154
155	0	1	1	0	0	0	0	155
156	0	1	1	0	0	0	0	156
157	0	1	1	0	0	0	0	157
158	1	1	1	0	0	0	0	158
159	1	1	1	0	0	0	0	159
160	1	1	1	0	0	0	0	160
161	1	1	1	0	0	0	0	161
162	1	1	1	0	0	0	0	162
163	1	1	1	0	0	0	0	163
164	1	1	1	0	0	0	0	164
165	1	1	1	0	0	0	0	165
166	1	1	1	0	0	0	0	166
167	1	1	1	0	0	0	0	167
168	1	1	1	0	0	0	0	168
169	1	1	1	0	0	0	1	169
170	1	1	1	0	0	0	1	170
171	1	1	1	0	1	0	1	171
172	1	1	1	0	1	0	1	172
173	1	1	1	0	1	1	1	173
174	1	1	1	0	1	1	1	174
175	1	1	1	0	1	1	0	175
176	1	1	1	0	0	1	0	176
177	1	1	1	0	0	0	0	177
178	1	1	1	0	0	0	0	178
179	1	1	1	0	0	0	0	179
180	1	1	1	0	0	0	0	180
181	1	1	1	0	0	0	0	181
182	1	0	1	0	0	0	0	182
183	1	0	1	0	0	0	0	183
184	1	0	1	0	0	0	0	184
185	1	0	1	0	0	0	0	185
186	1	0	1	0	0	0	0	186
187	1	0	1	0	0	0	0	187
188	1	0	1	0	0	0	0	188
189	1	0	1	0	0	0	0	189
190	1	0	1	0	0	0	1	190
191	1	0	1	0	0	0	1	191
192	1	0	1	0	1	0	1	192
193	1	0	1	0	1	0	1	193
194	1	0	1	0	1	1	1	194
195	1	0	1	0	1	1	1	195
196	1	0	1	0	1	1	0	196
197	1	0	1	0	0	1	0	197
198	1	0	1	0	0	0	0	198
199	1	0	1	0	0	0	0	199
200	1	0	1	0	0	0	0	200
201	1	0	1	0	0	0	0	201
202	0	0	1	0	0	0	0	202

Fig. 24.--Continued Circuit 4 run 1.

203	0	0	1	0	0	0	0	203
204	0	0	1	0	0	0	0	204
205	0	0	1	0	0	0	0	205
206	0	0	1	0	0	0	0	206
207	0	0	1	0	0	0	0	207
208	0	0	1	0	0	0	0	208
209	0	0	1	0	0	0	0	209
210	0	0	1	0	0	0	1	210
211	0	0	1	0	0	0	1	211
212	0	0	1	0	1	0	1	212
213	0	0	1	0	1	0	1	213
214	0	0	1	0	1	0	1	214
215	0	0	1	0	1	0	1	215
216	0	0	1	0	1	0	0	216
217	0	0	1	0	0	0	0	217
218	0	0	1	0	0	0	0	218
219	0	0	1	0	0	0	0	219
220	1	0	1	0	0	0	0	220
221	1	0	1	0	0	0	0	221
222	1	0	1	0	0	0	0	222
223	1	0	1	0	0	0	0	223
224	1	0	1	0	0	0	0	224
225	1	0	1	0	0	0	0	225
226	1	0	1	0	0	0	0	226
227	1	0	1	0	0	0	0	227
228	1	0	1	0	0	0	0	228
229	1	0	1	0	0	0	0	229
230	1	0	1	0	0	0	0	230
231	1	0	1	0	0	0	1	231
232	1	0	1	0	0	0	1	232
233	1	0	1	0	1	0	1	233
234	1	0	1	0	1	0	1	234
235	1	0	1	0	1	1	1	235
236	1	0	1	0	1	1	1	236
237	1	0	1	0	1	1	0	237
238	1	0	1	0	0	1	0	238
239	1	0	1	0	0	0	0	239
240	1	0	1	0	0	0	0	240
241	1	0	1	0	0	0	0	241
242	1	1	1	0	0	0	0	242
243	1	1	1	0	0	0	0	243
244	1	1	1	0	0	0	0	244
245	1	1	1	0	0	0	0	245
246	1	1	1	0	0	0	0	246
247	1	1	1	0	0	0	0	247
248	1	1	1	0	0	0	0	248
249	1	1	1	0	0	0	0	249
250	1	1	1	0	0	0	0	250
251	1	1	1	0	0	0	0	251
252	1	1	1	0	0	0	1	252
253	1	1	1	0	0	0	1	253

Fig. 24.--Continued Circuit 4 run 1.

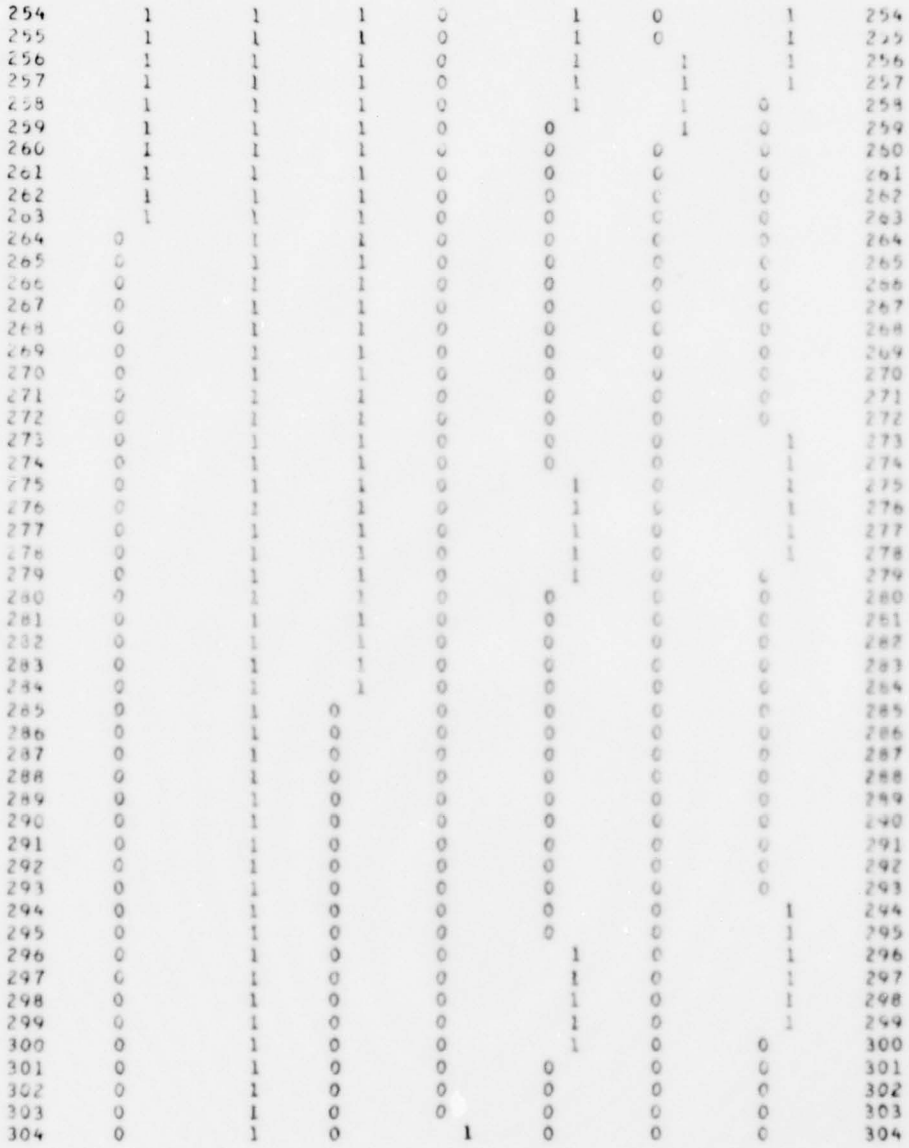


Fig. 24.--Continued Circuit 4 run 1.

305	0	1	0	1	0	0	0		305
306	0	1	0	1	0	0	0		306
307	0	1	0	1	0	0	0		307
308	0	1	0	1	0	0	0		308
309	0	1	0	1	0	0	0		309
310	0	1	0	1	0	0	0		310
311	0	1	0	1	0	0	0		311
312	0	1	0	1	0	0	0		312
313	0	1	0	1	0	0	0		313
314	0	1	0	1	0	0	0		314
315	0	1	0	1	0	0		1	315
316	0	1	0	1	0	0		1	316
317	0	1	0	1	1	0		1	317
318	0	1	0	1	1	0		1	318
319	0	1	0	1	1	0		1	319
320	0	1	0	1	1	0		1	320
321	0	1	0	1	1	0	0		321
322	0	1	0	1	0	0	0		322
323	0	1	0	1	0	0	0		323
324	0	1	0	1	0	0	0		324
325	0	1	1	1	0	0	0		325
326	0	1	1	1	0	0	0		326
327	0	1	1	1	0	0	0		327
328	0	1	1	1	0	0	0		328
329	0	1	1	1	0	0	0		329
330	0	1	1	1	0	0	0		330
331	0	1	1	1	0	0	0		331
332	0	1	1	1	0	0	0		332
333	0	1	1	1	0	0	0		333
334	0	1	1	1	0	0	0		334
335	0	1	1	1	0	0	0		335
336	0	1	1	1	0	0		1	336
337	0	1	1	1	0	0		1	337
338	0	1	1	1	1	0		1	338
339	0	1	1	1	1	0		1	339
340	0	1	1	1	1	0		1	340
341	0	1	1	1	1	0		1	341
342	0	1	1	1	1	0	0		342
343	0	1	1	1	0	0	0		343
344	0	1	1	1	0	0	0		344
345	0	1	1	1	0	0	0		345
346	1	1	1	1	0	0	0		346
347	1	1	1	1	0	0	0		347
348	1	1	1	1	0	0	0		348
349	1	1	1	1	0	0	0		349
350	1	1	1	1	0	0	0		350
351	1	1	1	1	0	0	0		351
352	1	1	1	1	0	0	0		352
353	1	1	1	1	0	0	0		353
354	1	1	1	1	0	0	0		354
355	1	1	1	1	0	0	0		355

Fig. 24.--Continued Circuit 4 run 1.

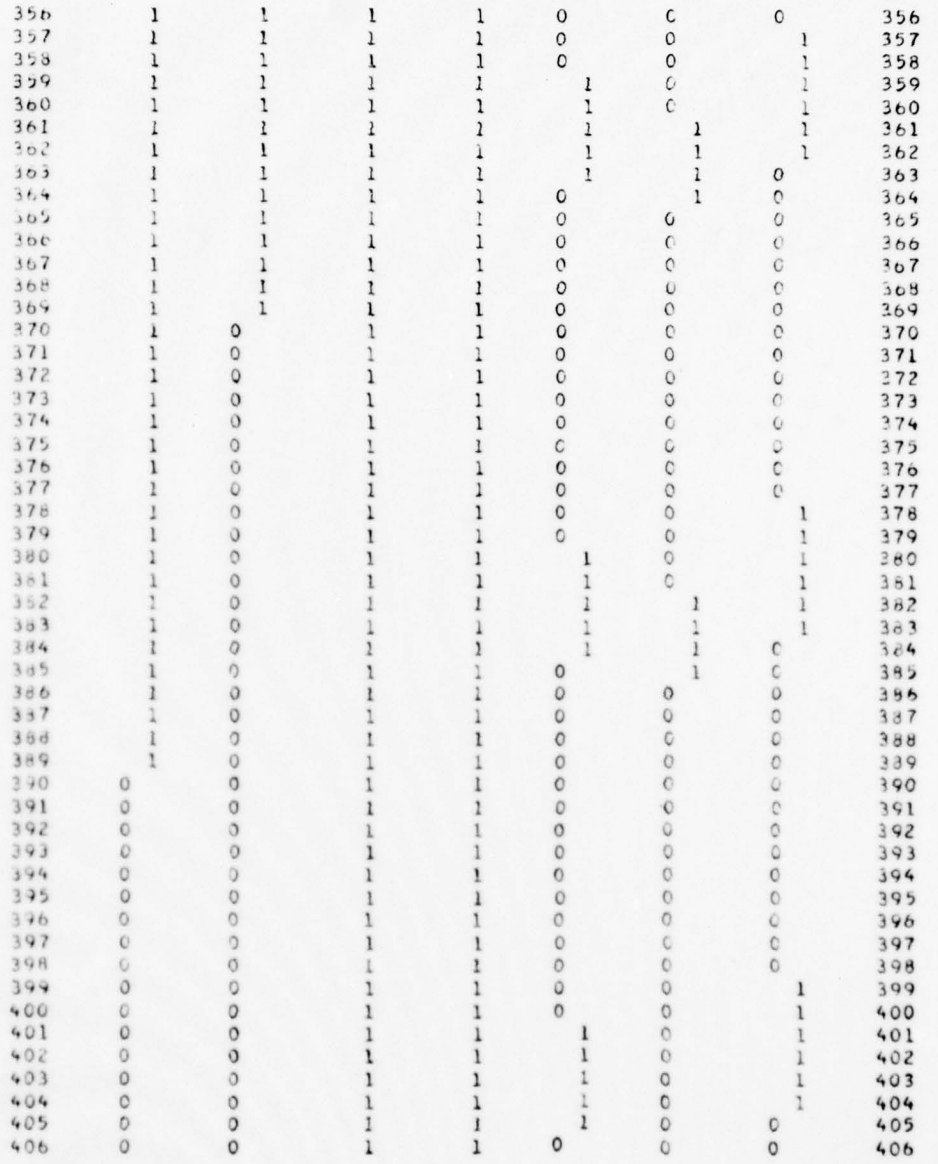


Fig. 24.--Continued Circuit 4 run 1.

407	0	0	1	1	0	0	0	407
408	0	0	1	1	0	0	0	408
409	0	0	1	1	0	0	0	409
410	0	0	1	1	0	0	0	410
411	0	0	1	0	0	0	0	411
412	0	0	1	0	0	0	0	412
413	0	0	1	0	0	0	0	413
414	0	0	1	0	0	0	0	414
415	0	0	1	0	0	0	0	415
416	0	0	1	0	0	0	0	416
417	0	0	1	0	0	0	0	417
418	0	0	1	0	0	0	0	418
419	0	0	1	0	0	0	0	419
420	0	0	1	0	0	0	1	420
421	0	0	1	0	0	0	1	421
422	0	0	1	0	1	0	1	422
423	0	0	1	0	1	0	1	423
424	0	0	1	0	1	0	1	424
425	0	0	1	0	1	0	1	425
426	0	0	1	0	1	0	0	426
427	0	0	1	0	0	0	0	427
428	0	0	1	0	0	0	0	428
429	0	0	1	0	0	0	0	429
430	1	0	1	0	0	0	0	430
431	1	0	1	0	0	0	0	431
432	1	0	1	0	0	0	0	432
433	1	0	1	0	0	0	0	433
434	1	0	1	0	0	0	0	434
435	1	0	1	0	0	0	0	435
436	1	0	1	0	0	0	0	436
437	1	0	1	0	0	0	0	437
438	1	0	1	0	0	0	0	438
439	1	0	1	0	0	0	0	439
440	1	0	1	0	0	0	0	440
441	1	0	1	0	0	0	0	441
442	1	0	1	0	0	0	0	442
443	1	0	1	0	0	0	0	443
444	1	0	1	0	0	0	0	444
445	1	0	1	0	0	0	0	445
446	1	0	1	0	0	0	0	446
447	1	0	1	0	0	0	0	447
448	1	0	1	0	0	0	0	448
449	1	0	1	0	0	0	0	449
450	1	0	1	0	0	0	0	450

Fig. 24.--Continued Circuit 4 run 1.

clock frequency of 7.6 mHz. Certainly, the simulation is valid.

Figure 25 is provided in order to point out what happens when the user makes an input control error. Notice that the user has requested output for element number 35. Referring back to Fig. 23, it can be seen that no such element existed.

```
                CIRCUIT NUMBER =     4
**************ERROR MESSAGES**************
ERROR =  205      THIS IS A FATAL ERROR
LIST OF REQUESTED ELEMENT OUTPUTS
   1    2    3    4    5   35 1001   -0   -0   -0   -0   -0
```

Fig. 25. Circuit 4 run 2.

CHAPTER 7

USER'S GUIDE TO EVESIM

This chapter describes the use of EVESIM as it exists at The University of Arizona. It outlines the control cards required and the input data format. Also included is a detailed list of the existing error messages and their interpretation. The last two sections of the chapter are dedicated to explaining the debug capabilities of the program and explaining how a user would write his two element models.

General Input Data

There are two phases to the process of using EVESIM. The circuit to be simulated must be prepared for the simulation and the required data must be punched on cards. Though the circuit preparation requirements have been discussed throughout the previous chapters, it is repeated for user clarity.

Circuit Preparation

EVESIM must be provided with a description of the actual circuit to be simulated. The information is taken from the circuit diagram. This requires that each element

have a unique number for identification. The numbering is begun at 1 and continues sequentially through n elements.

An EVESIM user must then refer to a manufacturer's logic data book and determine the values for t_{PLH} and t_{PHL} for all the elements in the circuit. From this information the user should determine a value for Δt which by definition is the greatest common divisor of the element delays. All element delay times are then determined by $n_e = D_e \div \Delta t$ where D_e is the element delays t_{PLH} or t_{PHL} as appropriate. As an example, consider two types of elements with delays $t_{PLH1} = 10$ nsec, $t_{PHL1} = 6$ nsec, $t_{PLH2} = 12$ nsec, and $t_{PHL2} = 8$ nsec. The greatest common divisor (Δt) is 2 nsec. The values for n are then computed:

$$n_{1LH} = 10 \div 2 = 5 \qquad n_{1HL} = 6 \div 2 = 3$$

$$n_{2LH} = 12 \div 2 = 6 \qquad n_{2HL} = 8 \div 2 = 4$$

One simulation time is now equivalent to 2 nsec and all delay parameters have been defined in terms of simulation times. It should be noted that all timing information now required as input data shall be in this same format, adjusted by Δt.

Card Input Format

The data cards are presented and discussed in the sequence that they should appear for a proper program execution. The normal system control cards required are:

AD-A052 613 ARIZONA UNIV TUCSON DEPT OF ELECTRICAL ENGINEERING F/G 9/2
SIMULATION OF DIGITAL CIRCUITS.(U)
1976 D M MOEN

UNCLASSIFIED NL

2 OF 2
AD
A052 613

END
DATE
FILMED
5-78
DDC

```
JOB CARD, CM70000
ATTACH (SIM, ID = MOEN)
SIM
7/8/9
```

The circuit data cards follow the system control cards. Unless otherwise noted, all numbers are right justified integers in 5 column fields.

Card 1: Title Card. The first 10 columns are used for the month, day, and year; for example, Mar10,1975. The circuit number is right justified to column 20. The total number of simulation times to be executed is right justified to column 25 and the debug mode indicator is right justified to column 30. The use of debug is discussed later.

Card 2: Circuit Parameters. NGATE is the total number of elements in the circuit (col. 1-5). NEXIN is the total number of external inputs to the circuit (col. 6-10).

Card 3: Element Description. Each circuit element description is contained on a single card. That is, each element in the circuit will have a card with the following format:

Element circuit number (col. 1-5).

Element type (col. 7-10). The element type must be one of the following: JKM1, NAN2, NAN3, NAN4, NAN8, NOR2, INVE, XOR2, OR2, or AND. All

flip-flops are to be listed first. Then all other memory devices followed by the combinational logic elements.

Element input connections to this element. They are listed in 5-column fields beginning with columns 16-20 up to column 45 and then beginning with columns 56-60 up to column 75. The maximum number of inputs is 10. External input X is designated by 1000 + X. A constant logical 1 is indicated by 1000. A constant logical 0 is indicated by -1000. The second output of an element Y, for example $\overline{Q}$ of a flip-flop, is indicated by 2000 + Y. The third output would be indicated by 3000 + Y, etc.

Element delay information appears as indicated below (right justified):

clocked or normal t_{PLH} (col. 47-48)

clocked or normal t_{PHL} (col. 49-50)

preset, clear or second t_{PLH} (col. 51-52)

preset, clear or second t_{PHL} (col. 53-54)

Card 4: Memory Element Information.

LFF is the total number of flip-flops (col. 1-5)

LOM is the total number of other memory type elements (col. 6-10)

NCLWID is the first minimum clock pulse width (col. 11-15)

NPCWID is the first minimum preset or clear pulse width (col. 16-20)

NCLW2 is the second clock pulse width (col. 21-25)

NPCW2 is the second minimum preset or clear pulse width (col. 25-30)

Card 5: Initial value of the Q output of all flip-flops, listed in order as they appear above, beginning with column one. Every column is used. A 0 represents logical 0. A 1 represents logical 1. A 2 represents an unknown condition.

Card 6: Same as card 5 except for all other memory devices, if any, otherwise skip this card.

Card 7: Initial value of external inputs listed in order, beginning with the clock (1001). Again, each column should have a 0, 1, or 2.

Card 8: Initialization parameters LIMIT is the limit of the number of attempts that will be made to initialize. LOVER is set to 1 if the user desires to override LIMIT, otherwise LOVER is left blank.

Card 9: External Input Changes.

IETIME is the simulation time that the indicated external inputs are to change value (col. 1-5)

INUV(15,2) is the number (1000 + K) of the external input that is to change and the value (0, 1, or 2) to which it is to change. The number appears in 5-column fields beginning in columns 7-11,

the first four (7-10) columns contain the external input number and the fifth column (11) contains the new value of the output. This can be repeated fifteen times per card; however, only events for the time indicated in the first five columns can be on a card.

Following the circuit data cards, a 7/8/9 card is used to separate the output requirements card, card number 10.

Card 10: Element Outputs to be Plotted. The numbers of the elements to be plotted appear in 5-column fields beginning in columns 6-10. Twelve numbers are allowed per card.

The final card is a system control card. It is a 6/7/8/9 card.

Error Messages

The error messages can occur as either fatal or non-fatal. Fatal errors are indicated by numbers greater than 199. Non-fatal errors are less than 199.

Fatal Errors

Fatal errors cause execution to stop in all cases except error 203 which has an override condition. Normally, the fatal errors are programming errors. The present fatal error messages are as follows:

Error 200: The user has asked for an element model that does not exist.

Error 201: The user has specified an element that has a fanout greater than 10.

Error 202: A memory type element is out of sequence as required by data card 3.

Error 203: EVESIM was unable to initialize the circuit. Increase LIMIT and/or set LOVER equal to 1 on data card 8.

Error 204: Array LIST(I) is full and an attempt was made to schedule another event. The user has reached the limit of EVESIM. The array LIST(I) must be increased in size.

Error 205: The user requested the output of an element that could not be found in the output file.

Non-Fatal Errors

Non-fatal errors are errors that occur during the course of a simulation of a circuit caused by generation of illegal conditions within the circuit. These error messages are intended to aid the user in identifying flaws in the circuit design or implementation of a design. The existing error messages are as follows:

Error 1: The input pulse to the clock input of a flip-flop is too narrow.

Error 2: The J and/or K inputs to the JK master-slave flip-flop are not stable while the clock is high.

Error 3: The preset and clear inputs to a JK master-slave flip-flop are changing to a high value at the same time.

Error 4: The input pulse to the clear input of a flip-flop is too narrow.

Error 5: The input pulse to the preset input of a flip-flop is too narrow.

Error 6: The preset and clear inputs to a flip-flop are changing at the same time before either a preset or clear input was able to cause a change on the output of the flip-flop.

Error 7: The J and K inputs to a JK flip-flop have the same value. The present state of the flip-flop is unknown; therefore, a new output cannot be determined.

Debug

During the course of the development of EVESIM, a debug ability was integrated into the program. Originally, the intent for its use was only for development purposes and was later to be removed from the program. However, it has been left in the program as an aid to continued maintenance of the program.

Debug basically consists of three options. The first is to leave column 30 blank on data card 1. This effectively provides no debug information. A 1 in column 30 will execute all portions of debug. If DEBUG is set equal to 2, array IDESOU(I,J) is not printed. All other debug information is printed. If DEBUG equals 3, array LCOUNT(I) is printed.

Figure 26 is an example output for DEBUG equal to 1. The first four lines is the array IDESCR(I,J) as defined in Chapter 3. The next 10 lines is the array IDESOU(I,J) also defined in Chapter 3. The clock fanout is listed next. The remaining information is self-explanatory.

Array LCOUNT(I) is used as a bookkeeper. It is used to count the number of times a particular subroutine is called during a simulation execution. Table 6 defines LCOUNT(I).

Adding Element Models

Table 6 indicates that there are more element models available for use than were previously mentioned. These include a D flip-flop (DET1), a 4-bit counter (COUN), a 4-bit parallel-in-parallel-out shift register (SHRE), a 4-bit adder (ADD4), and a BCD-to-decimal decoder (DECO). In addition, an unused name for a combinational logic gate has been provided called UNS. At this time, these models have not been included in EVESIM. However, the overhead required

```
                CIRCUIT NUMBER =    5

**************ERROR MESSAGES***************

 1  JKM1      11   1   5   8  -0  -0  -0 2002    3 1002 1000 1000   -0  -0  -0  -0  -0  -0
 2  JKM1      11   3   5   8   3   5  -0 2002    3 1001 1000    4   -0  -0  -0  -0  -0  -0
 3  INVE       4   5   3   2  -0  -0  -0 2002   -0  -0  -0  -0  -0  -0  -0  -0  -0  -0
 4  INVE       4   6   3   2  -0  -0  -0 1002   -0  -0  -0  -0  -0  -0  -0  -0  -0  -0
 1       1  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 2       1  11   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0
 3       2  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 4       2  11   0   1   1   0   0   1   2   3   0   0   0   0   0   0   0
 5       3   4   0   0   0   0   0   1   2   0   0   0   0   0   0   0   0
 6       4   4   0   1   1   0   0   2   0   0   0   0   0   0   0   0   0
 7   -1000   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 8    1000   0   0   1   1   0   0   0   0   0   0   0   0   0   0   0   0
 9    1001   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
10    1002   0   0   0   0   0   0   1   4   0   0   0   0   0   0   0   0

CLOCK FAN OUT    2   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   0   0
    0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0
EVENT CARD TIME =     1     INPUT NUMBER AND VALUE     AS ON CARD =
                          10011  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*
NUMBER OF ELEMENTS SCHEDULED FOR THIS TIME =     0
JKM1 WAS CALLED AT     1     LC1 =     1     LC2 =     0
    LP1 =     1     LP2 =     1     LL1 =     1     LL2 =     1
EVENT CARD TIME =     5     INPUT NUMBER AND VALUE     AS ON CARD =
                          10010  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*  -0*

TIME NOW =     1
NUMBER OF ELEMENTS SCHEDULED FOR THIS TIME =     0
TIME NOW =     2
NUMBER OF ELEMENTS SCHEDULED FOR THIS TIME =     0
TIME NOW =     3
NUMBER OF ELEMENTS SCHEDULED FOR THIS TIME =     0
TIME NOW =     4
NUMBER OF ELEMENTS SCHEDULED FOR THIS TIME =     0
JKM1 WAS CALLED AT     5     LC1 =     0     LC2 =     1
    LP1 =     1     LP2 =     1     LL1 =     1     LL2 =     1
TIME =     5     LAST TIME CLOCK CHANGED =     1     THE CLOCK HAS CHANGED FROM HIGH TO LOW AND PRESET AND CLEAR ARE   BOTH HIGH
INTERNAL CLOCK TIME OF INSERTION =     1     REAL     TIME OF INSERTION =    10LENGTH OF INTERNAL CLOCK =     9
ELEMENT BEING WORKED ON = IOD1 =     3
TIME NOW =     5     TOP OF THE LIST =  1023
 ICL    1 1024   -0    0   -0    0   -0    0    0    0    0    0    0    0    0    0    0    0    3

INTERNAL CLOCK TIME OF INSERTION =     4     REAL     TIME OF INSERTION =    13LENGTH OF INTERNAL CLOCK =     9
ELEMENT BEING WORKED ON = IOD1 =     4
TIME NOW =     5     TOP OF THE LIST =  1022
 ICL    1 1024   -0    0   -0    0    1 1023    0    0    0    0    0    0    0    0    0    0    4
```

Fig. 26. Debug output.

Table 6. Definition of LCOUNT(I).

I	Subroutine
1	Output
2	FUNSIM
3	JKM1
4	DET1
5	COUN
6	SHRE
7	MOD1
8	SCHED
9	UNSCHE
10	COMLOG
11	NAN2
12	NAN3
13	NAN4
14	NAN8
15	NOR2
16	INVE
17	XOR2
18	OR2
19	AND
20	UNS
21	ADD4
22	DECO
23	IODEST
24	ERROR
25	READIN
26	INCOND
27	INITIAL
28	PLOT1

has been programmed into EVESIM to include these models. This has been done in order to facilitate the addition of these models.

The user of the program is reminded that the model is required to calculate when an event shall occur. The actual scheduling of the event is accomplished through the use of subroutine SCHED. Certain information must be available to the model. This information is passed via common Blocks. The required common Blocks are listed in Table 7.

The values of IOD1-IOD12 have been established by subroutine IODEST. IOD1 is the pointer to IDESOU(I,J) of the element being processed. IOD2-IOD12 are the pointers to IDESOU(I,J), which are the input elements to IOD1, listed in order. The value of the input is found by IDESOU(IOD_n, J05). The new outputs are calculated and the times of change are calculated. The times of change are placed in IDESOU($IOD1_n$, J06) and the new values are placed in IDESOUT($IOD1_n$, J07) and NOUT(N). Subroutine SCHED is called to schedule the events in the future.

In order to add more models than are listed in Table 6, additional changes must be made. The size of LTYAL(16) and LTNOU(16) must be increased to reflect the total number of models. In conjunction with this Block Data subroutine DES must be changed. LTYAL(n) must contain the new element name and LTNOU(n) must contain the number of outputs of the

Table 7. Common blocks required by subroutine models.

```
COMMON/ELEM/LECH, IETIME

COMMON/INCL/ICL(100,2), LIST(1024), LSP, LENCL, INUV(15,2)

COMMON/MACH/IND, INI

COMMON/DESCRP/IDESCR(100,19), IDESOU(350,17), JI1, JI2, JI3,
JI4, JI5, JI6, JI7, JI8, JI9, JI10, JI11, JI12, JI13, JI14,
JI15, JI16, JI17, JI18, JI19, J01, J02, J03, J04, J05, J06,
J07, J08, J09, J010, J011, J012, J013, J014, J015, J016,
J017, LISTCL(50)

COMMON/WORK/IOD1, IOD2, IOD3, IOD4, IOD5, IOD6, IOD7, IOD8,
IOD9, IOD10, IOD11, IOD12, II, NOUT(10), NGATE, IEX, LFF,
LOM, LTYAL(16), LTNOU(16), NAVL, IODL, NOW, NSCTI, LERROR,
NEXIN, LFATAL

COMMON/VALE/LOW, LHIGH, LUNK, IFAN, LIMIT, LOVER

COMMON/DEBUG/ LBU, LCOUNT(40)

COMMON/WID/NCLWID, NPLWID, NCLW2, NPCW2
```

new element. NAVL must also be adjusted to reflect the total number of models available. With these changes, additional models can be added as explained above.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

EVESIM has been found to be an effective program for analyzing a digital logic circuit. EVESIM provides a means for the designer to quickly verify his logic circuit designs. EVESIM also gives the designer a tool with which his design can be optimized and quickly re-verified. The time consuming task and manually tedious operation of design verification of logic circuits has been reduced to a computer program.

Conclusions

The original objectives as stated in Chapter 1 have been accomplished. EVESIM is a level mode simulator that uses element level models. The efficiency gained in using a time-driven event simulator have allowed the original objectives to be expanded to include the use of three valued logic and the individual assignment of delays depicting whether an element output is rising or falling. In addition, the use of an iterative process during initialization has allowed a limited capability for circuit stability analysis.

The three valued logic is used to identify illegal conditions or unknown conditions generated in a circuit.

This information is used by the designer to improve his design or identify flaws to be corrected in his design. The three valued logic is also used during the initialization process. The initialization of the circuit can then be accomplished within the program without any requirements for ordering of the elements of the circuit.

The assignment of delays relative to whether a signal is rising or falling presents a much improved model over other techniques. The modeling accuracy gained is in keeping with the growing need for more accurate timing information as the speed of logic increases. This technique also insures that EVESIM will not grow old and useless as logic families change.

Iterations are made through the circuit during initialization until all unknown conditions are cleared and no element outputs are still toggling, or until the preset level of attempts to initialize is reached. If increasing the number of iterations does not result in the elimination of elements toggling, then those elements toggling are flagged as being part of an unstable condition. Certainly, this may be part of the design, and therefore EVESIM has an override capability. However, at a minimum, the user is made aware of the situation such that he can make a decision as to whether it will be allowed.

EVESIM has also provided the user with an improved ability to identify errors in his circuit through the use of

error messages. In conjunction with this, debugging information is available when it is needed.

It is hoped that most programming errors have been eliminated from EVESIM. However, it is a rule of programming that the elimination of one error is going to cause another error some other place in the program. It is time that EVESIM move from a development phase to a phase of wide usage. It is during this phase of wide usage that those hidden programming errors will be identified. It is hoped that EVESIM will continue to grow and be expanded to include the recommendations of the next section.

Recommendations

The process of developing a program is never ending. If time were not a factor, the author could continue forever adding to and improving the program. Areas of recommended improvement for EVESIM are discussed here in hope that someone may be able to include them in a future version.

Assignment of delay values can be partially automated. Certainly, it would be more realistic to assign delays automatically by a random number generator process. The limits of the random number generator could easily be the minimum and maximum values of delays as depicted in a data book. This would relieve the burden to the user of assignment of "typical" delay values. In conjunction with this process, the delay value assignment could also be

weighted by the fanout of the element. This would not be a major extension since EVESIM already generates a fanout description for other purposes.

A further extension of the delay information would be to add a fourth logic value. The intended use of this fourth value would be to indicate that an element output is changing in value. This could possibly be defined as a non-propagating unknown condition. It would represent the element output during the time from when an input change has occurred until the output was stable at its new value.

The timing mechanism for EVESIM is no longer very efficient if the length of the internal clock is forced to become very long as the result of a single long delay. For example, a circuit containing a single one-shot element would require the clock to be of the length of the one-shot. This problem can be overcome by establishing an events queue for events that are beyond the length of the clocking mechanism. If these events are placed in a stack and ordered by time, then after each pass through the clocking mechanism, the event at the top of the stack is checked to see whether the event will fit into the next pass through the clocking mechanism. If it will fit in the time frame of the next pass, it is then inserted appropriately into the normal timing mechanism for processing. This technique would be much more efficient than extending the length of the clock to accommodate a single element.

The final recommendations is probably a much longer range goal than the previous recommendations. The above recommendations require minimal changes to EVESIM. However, it is considered that an extension to include an ability to perform critical race analysis would be an ultimate and final addition to EVESIM. The inclusion of this ability along with the existing SCRITSS test sequence generator and the completion of the MACRO AHPL compiler would provide maximum automation to logic circuit design.

REFERENCES

Anwaruddin. "Efficient Simulation of Logic Networks." Thesis, Department of Electrical Engineering, University of Arizona, 1969.

Breuer, M. A. "Functional Partitioning and Simulation of Digital Circuits," IEEE Transactions on Computers, C-19:1038-1048 (November, 1970).

Chappell, S. G., C. H. Elmendorf, and L. D. Schmidt. "LAMP: Logic-Circuit Simulators," The Bell System Technical Journal, Vol. 53, No. 8 (October 8, 1974).

Components Group, Engineering Staff. The TTL Data Book for Design Engineers. First Edition. Texas Instruments Incorporated, 1973.

Hayes, Gwendolyn G. "Computer-Aided Design: Simulation of Digital Design Logic," IEEE Transactions on Computers, C-18:1-10 (January, 1969).

Hill, F. J., and G. R. Peterson. Introduction to Switching Theory and Logical Design. 2nd ed. New York: John Wiley and Son, 1974.

Larson, R. P., and M. M. Mano. "Modeling and Simulation of Digital Networks," Communications of the ACM, 8: 302-312 (May, 1965).

Stang, D. R. "Simulation of a Small Logic System with a FORTRAN Program," Computer Design, 7:56-60 (January, 1968).

Stockwell, G. N. "Computer Logic Testing by Simulation," IRE Transactions on Military Electronics, Mil-5:275-282 (July, 1962).

Szygenda, S. A., and E. W. Thompson. "Digital Logic Simulation in a Time-Based, Table-Driven Environment, Part 1, Design Verification," Computer, Vol. 8, No. 3, 24-36 (March, 1975).

Ulrich, E. G. "Exclusive Simulation of Activity in Digital Networks," Communications of the ACM, 12:102-110 (February, 1969).

Weingarten, F. W. "Simulation of Computer Logic by FORTRAN ARITHMETIC," Communications of the ACM, 8:516-517 (August, 1965).

Williams, T. L. "Logic Design, Part 1," Digital Design, 5:118-121 (April, 1975a).

Williams, T. L. "Logic Design, Part 2," Digital Design, 5:58-65 (May, 1975b).