A COMPUTER-AIDED
PROCEDURE FOR DETERMINATION OF STATES OF DIGITAL LOGIC
CIRCUITS AROUND OVERLAPPED TRANSITIONAL INPUTS

By
SANJOY KUMAR Poddar

A Thesis
submitted to the department of Computer Engineering,
Bangladesh University of Engineering and Technology, Dhaka,
in partial fulfilment of the requirements for the degree
of
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

July, 1985
DECLARATION

I do hereby declare that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

Sanjay Kumar Paddar
Candidate 10.8.85
Accepted as satisfactory for partial fulfilment of the requirements for the degree of M.Sc. in Computer Engineering.

Board of Examiners

(Dr. Mr. Mujibur Rahman)
Professor
Department of Electrical & Electronic Engineering
BUET, Dhaka, Bangladesh.

(Dr. A. K. M. Mahfuzur Rahman Khan)
Professor & Head
Department of Computer Engineering
BUET, Dhaka, Bangladesh.

(Chairman
(Supervisor)

(Dr. Shamsuddin Ahmed)
Professor & Dean
Faculty of Electrical & Electronic Engineering
BUET, Dhaka, Bangladesh.

(Dr. Syed Mahbubur Rahman)
Assistant Professor
Department of Computer Engineering
BUET, Dhaka, Bangladesh.

Member

Member

Member

Member (External)
ACKNOWLEDGEMENTS

This thesis was prepared under constant guidance and keen supervision of Dr. Md. Mujibur Rahman. To him the author owes his deepest gratitude. He expresses his sincere gratitude to Dr. Abdul Matin Patwari, Vice-Chancellor and Dr. Shamsuddin Ahmed, Dean of the faculty for their constant encouragement. He is grateful to Dr. Mahfuzur Rahman Khan, Head of the Computer Engineering Department for his advice and rendering all possible help. He is also grateful to Dr. Syed Mahbubur Rahman and Dr. Lutfar Rahman for their advice in carrying out the work.

The author offers his sincere thanks to the staff of Computer Centre of BUET for their co-operation in using the Computer.

The author expresses his sincere gratitude to all those people who have collaborated in this work. Among those he wishes to mention a few but thanks all others not mentioned here.
A software algorithm has been developed and implemented to find the states of digital logic circuits around overlapped transitional inputs. The algorithm considers differences of propagation delays, inertial delay of logic element and undefined state so that, for very fast transitional and overlapped excitation signals, the node states can be determined with safe accuracy. It is simple and poses very little restrictions to any complexity of the circuit with basic gates and flip-flops, any order of gate connection description and any sequence of the excitation signals. The program developed inputs the circuit and environment descriptions in the form of a pseudo-language and gives a variety of output, so that, circuit status like signal levels, hazards and logic ambiguities can be easily detected. The output also includes some lists for future reference.
CONTENTS

Acknowledgements                      i
Abstract                                     ii

Chapter 1  INTRODUCTION

1.1 General                                1

1.2 Types of Simulation

   1.2.1 Comparison of Asynchronous and Synchronous Simulation  2

1.3 Asynchronous Simulation and introduction of propagation delay  5

1.4 Simulation Models

   1.4.1 Compile Code model                       7
   1.4.2 Table driven model                      9
   1.4.3 Comparison of two models                 10

1.5 Literature survey                      11

1.6 Scope of the present work              13

Chapter 2  REVIEW OF SYSTEM MODELS

2.1 Introduction                           14

2.2 Language definition and commands

   2.2.1 Logic elements and expressions

      2.2.1.1 Gates                                 14
      2.2.1.2 Flip-flops                           15

   2.2.1.3 Macro and Black-Box expression         18

2.2.2 Simulation commands

2.3 Translation and simulation

   2.3.1 Data dependent code model                23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4</td>
<td>Special Software techniques</td>
<td>24</td>
</tr>
<tr>
<td>2.4.1</td>
<td>General purpose system simulators</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2</td>
<td>PERT</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Specialized computer techniques</td>
<td>26</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>DEVELOPMENT OF SIMULATION MODEL</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Timing specification</td>
<td></td>
</tr>
<tr>
<td>3.2.1</td>
<td>Basic gate</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Exclusive-Or gate</td>
<td>29</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Flip-flop gates</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Special considerations on timing data</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Considerations for algorithm development</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Model selection</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Table generation for simulation</td>
<td>37</td>
</tr>
<tr>
<td>3.7</td>
<td>Time mapping for event control</td>
<td>41</td>
</tr>
<tr>
<td>3.8</td>
<td>Algorithm for simulation</td>
<td>41</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>SYSTEM ORGANIZATION</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Process organization</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Units of simulation process</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Group (I) units</td>
<td></td>
</tr>
<tr>
<td>4.3.1.1</td>
<td>Basic interface unit</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1.2</td>
<td>Command processor</td>
<td>46</td>
</tr>
<tr>
<td>4.3.1.3</td>
<td>Data set processors</td>
<td>46</td>
</tr>
</tbody>
</table>
4.3.2 Group (II) units

4.3.2.1 Signal driving gate table generator

4.3.2.2 Signal state table generator

4.3.3 Group (III) units

4.3.3.1 Initial condition setter

4.3.3.2 Level-delay setter

4.3.3.3 Printout unit

4.3.3.4 Time outer

4.3.3.5 System reference time incremener

4.3.3.6 Gate reference time decremener

4.3.3.7 Output determination unit

4.3.3.8 Output setter unit

4.3.3.9 Update signal unit

4.4 State diagram of simulation process

Chapter 5  SOFTWARE DEVELOPMENT AND APPLICATION

5.1 Introduction

5.2 Language selection and program description

5.3 Circuit testing by the program

Chapter 6  CONCLUSION AND SUGGESTION FOR FUTURE WORK

6.1 Conclusion

6.2 Suggestion for future work
## Appendices:

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix I</td>
<td>Communication with the simulator</td>
<td>64</td>
</tr>
<tr>
<td>Appendix II</td>
<td>Tables used by the program</td>
<td>71</td>
</tr>
<tr>
<td>Appendix III</td>
<td>Program flowchart</td>
<td>77</td>
</tr>
<tr>
<td>Appendix IV</td>
<td>Gate timing specifications</td>
<td>103</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>107</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION
1. INTRODUCTION

1.1 GENERAL

Determination of signal states in a digital logic circuit becomes difficult when the excitation signals change in random fashion and their durations are comparable to delays of gates and interconnection lines. In these conditions, the circuit becomes completely asynchronous and the stray delays experienced by the signal need to be explicitly considered. An extensive timing analysis of the circuit is necessary to detect all the probable timing failures such as hazards, spikes, logic ambiguities etc. Timing diagrams and recommended operating conditions are given in data books but tracing signals along various paths, maintaining timing constraints, is a complex process and the digital computer can be an useful tool in such analysis.

Computers are used to simulate a realistic model of the logic circuit with sets of input stimuli to generate signal-versus-time behaviour. This can assist in system development by checking the logic, detecting timing problems and using alternate circuits and components, and obtaining detailed circuit operational reference data.
1.2 TYPES OF SIMULATION

Simulations are carried out in a number of ways depending on the requirements of the particular application. These may be classified as either synchronous or asynchronous, according to whether or not the response time is tied to the clock time. In the first type, logic operation of the entire circuit is performed under control of a real or hypothetical clock and gated inputs to flip-flops are used only when the clock is ON. In the second type, the circuit is considered asynchronous or fundamental mode circuit and indicates a clockless or free-running circuit. Excitation signals can change at any time and the circuit can reach a stable condition depending on the delays of the gate.

1.2.1 COMPARISON OF ASYNCHRONOUS AND SYNCHRONOUS SIMULATION

A synchronous simulator considers the circuit as a zero-delay network. All state changes are assumed to occur at the time at which a hypothetical clock pulses. This is only possible in a practical circuit if excitation to any node makes the entire circuit stable before a change of any excitation signal occurs. That is, the cumulative delay along any path of the on going signal in the circuit has to be less than the excitation signal duration. In such a modelling, therefore, the circuit behaviour which is actually dependent upon the relative delay through the different logic paths may not be
accurately depicted. However, for low speed operation and for long overlapped period of the excitation signals, the clock speed and signal variations are sufficiently slow in relation to gate delays such that the output of one gate is always available when needed as input by a succeeding gate.

A synchronous simulation may fail to detect the specific instance when too much time is required by an input to a gate to switch its state, or a race condition. Small time variations to state change of input signals to a gate can generate unwanted output pulse between clock times. These conditions cannot generally be detected by synchronous simulation. This leads to race condition of the circuit. In such a condition, the output of a gate depends on the precise order of arrival of the input signals. The example of AND gate in Fig. 1-1 is part of a complex control circuit.

![a) Circuit](image1)

![b) Timing diagram](image2)

**Fig. 1-1** Single spike condition.
Normally if the inputs go from (0,1) to (1,0), the output would be expected to be zero. Now if IN1 changes to 1 before IN2 changes to 0, for a short space of time both inputs remain 1; thus for this period output goes high, showing a short pulse or 'spike'. The duration of the short pulse may be sufficient enough to result in a false action of the control circuit.

Further, sometimes an output which is supposed to change state only once oscillates before finally settling down. In the circuit, Fig. 1-2, the order in which the input values IN1, IN2 and IN3 changes from (1,0,1) to (0,1,0) through the intermediate states (1,0,0) and (1,1,0) determines the intermediate values of the output OUT.

![Circuit Diagram](a) Circuit

![Timing Diagram](b) Timing diagram

Fig. 1-2 Several spikes condition.

The timing diagram shows the oscillation in the output. Sometimes this type of oscillation may also continue indefinitely as would happen in the case of a RS flip-flop,
if both inputs go to zero simultaneously and then high.

Asynchronous simulation can be used for both asynchronous and synchronous circuits and would also generally present a more useful test bed environment. It is better that even a synchronous circuit should first be tested using asynchronous simulation, so that any possible timing errors may be detected. After the asynchronous simulation, a safe zone of operation can be found so that small changes in the circuit will not create much of a problem to a later simulation in synchronous mode. Synchronous simulation is simpler and easy to implement since events can occur only at subsequent periodic clock times and need not concern itself with delays.

1.3 ASYNCHRONOUS SIMULATION AND INTRODUCTION OF PROPAGATION DELAY

In high speed application, the trouble with the logic circuit is the delay which is inherent in any electronic circuit. The delays are not constant and are difficult to implement in circuit analysis. Delays may be associated with gate or line or both. Line delay depends on interconnection length, type of insulation adjacent to the conductor, location of nearby conductors and grounds, number and states of gates which share the interconnection. Gate delay depends on the technology used to fabricate the gate, fanout, temperature and state of the gate.
For a practical circuit with fast transitional inputs and where output signals are to be picked up at or around the transition, both line and gate delays need to be considered. But the problem is that their introduction into simulation model fairly complicates the algorithm. Normally in asynchronous simulation only one of the delay is considered, which is the delay of the device itself (gate delay). This is supported by speed independent circuit theory\(^{(22)}\). If the line delay also becomes significant, dummy gates representing similar delays may be placed in the line to incorporate the effect\(^{(23)}\).

Two time units are normally used to specify gate delay. The first one is the propagation delay experienced by a signal change from the time a gate is excited to the time the gate output responds. This depends on the state of the gate and it has two distinct values; \( T_{PLH} \) — the delay for a signal to change the gate state from low to high and \( T_{PHL} \) — the delay for a signal to change the gate state from high to low. Sometimes one of the two values differ by even upto twice as much as the other. The second type of time unit is the setup delay \( T_{su} \), that defines a time unit for which an input change must persist to produce an effect at the output.

Normally asynchronous simulator assumes propagation delay as the average of \( T_{PHL} \) and \( T_{PLH} \). Therefore the time difference in switching a state of a signal between high-to-low and low-to-high is ignored. Some simulators do not consider setup delay and thus signal transition, however short it may be, is allowed to transmit through the gate. Sometimes asynch-
ronous simulators provide routines for flip-flops but avoid conforming signal specifications for proper operation. All the above assumptions provide a very straightforward algorithm and simulation time is greatly reduced. This however may result in faulty analysis of the circuit.

1.4 SIMULATION MODELS

The actual computer executable model of a digital network is formed by translating the input statements which describe the physical circuit. This translation basically results in any of two types of models. The first type of model is generated by compilation of input statements into a set of coded subprograms, each of which performs the function of a logic element\(^1,2\). The second type of model is composed of a group of data structures or tables which contain detailed information about the circuit\(^12,13\). The two types of models are discussed in sections 1.4.1 and 1.4.2.

1.4.1 COMPILE CODE MODEL

In this type the compiler of the simulator produces a model in the form of executable code. These codes are normally the machine code instruction of the host computer. Almost every computer has a set of machine code instructions that perform the basic Boolean operations. This set of instructions
are utilized to carry out the simulation. To illustrate this, a circuit which implements the equations, $E = (A \cdot B \cdot C)'$ and $F = (E \cdot D)$ is coded in the form shown in Fig. 1-3.

![Diagram of circuit](image)

**Table a) Circuit**

<table>
<thead>
<tr>
<th>Label</th>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL A</td>
<td>1</td>
<td>Load value of A in accumulator.</td>
</tr>
<tr>
<td>ANA B</td>
<td></td>
<td>A.B</td>
</tr>
<tr>
<td>ANA C</td>
<td></td>
<td>A.B.C</td>
</tr>
<tr>
<td>COM</td>
<td></td>
<td>(A.B.C)'</td>
</tr>
<tr>
<td>SLW E</td>
<td></td>
<td>(A.B.C)'$. E</td>
</tr>
<tr>
<td>TRA 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table b) Compiled code for NAND gate (gate G)**

<table>
<thead>
<tr>
<th>Label</th>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL E</td>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>ANA D</td>
<td></td>
<td>E.D</td>
</tr>
<tr>
<td>SLW F</td>
<td></td>
<td>E.D → F</td>
</tr>
</tbody>
</table>

**Table c) Compiled code for AND gate (Gate H)**

It is seen that, two different collections of coded statements or subprograms are required for representing the equations. Each collection of statements performs a basic gate operation.
1.4.2 TABLE DRIVEN MODEL

In this type, logic expressions are represented by data tables. The table given in Fig. 1-4 is the representation for the circuit in Fig. 1-3(a). The linkages between logical elements in the table corresponds to their structural connection in the network.

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAND</td>
</tr>
<tr>
<td>AND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element</th>
<th>Input table address</th>
<th># Fan-in</th>
<th>Output table address</th>
<th># Fan-out</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 1-4 Table-driven model

For a given logic element, e.g., gate G, the table defines its input signals (A, B, and C), the type of operation (NAND), and the output (E). Exercising the model is performed by an interpreter program. The interpreter utilizes the entries in the table to perform the required operations. The entry in the type column (e.g. NAND) determines the specific operation (NAND operation) to be performed. The operand (A, B, C)
addresses are obtained from the input table address and their number from the fan-in entry. The operation is performed by specific subroutine and output value is set. The address of output (E) is obtained from the output table address. The number of output is obtained from the fan-out entry.

In carrying out the operation only one set of machine code is used by the program for each logic operation. The specific set of machine code would be referenced each time the interpreter desires to perform that particular logic operation. The computed output of a stage determines the link to be followed by the interpreter to reference the next element in the data table.

1.4.3 COMPARISON OF TWO MODELS

In the compiled code model (sec. 1.4.1) a preliminary sorting of the machine code is necessary to the order in which the logic elements within the circuit are actually activated along the signal paths. This sorted model results in a comparatively simple and straightforward computer program design with fast execution. But sorting during compilation takes a lot of time. Since the compiled code model is inherently synchronous, simulating nominal propagation delays and feedback between logic elements is usually not performed. In addition even if a small change in the circuit is made, a reassembly of the entire produced code is necessary.
In table driven models, any complex algorithm with various delays can be implemented. In most of the cases, no ordered description of the circuit is necessary, so that translation time is reduced but the execution time is increased. This is because, there is no direct execution of equivalent machine code. Unlike the compiled code model, circuit modification can be easily taken care of.

1.5 LITERATURE SURVEY

Digital circuit finds immense importance in computer systems, industrial systems, consumer applications etc. and as such a great deal of analysis and implementations of gate-level logic simulation are found in the literature. The algorithms suggested and implemented have many common as well as different characteristics, depending on the requirements of the particular circuits. Some of the more important simulation programs and their mentionable considerations are discussed in this section.

J.G McKinney(11) describes a simple two value synchronous simulator based on sorted compiled code model. R.M. McClure(9) has developed another simulator, which contains declarations for registers as well as for flip-flops and gates. Gwendolyn G. Hays(5) developed a synchronous interpreter which can simulate delay lines and 18-bit, 4K computer memory, apart from normal gates and flip-flops.
Usually a small percentage of the gates in a logic system changes at any instant of time, in response to a new input. Therefore, in setting output of gates, only the excited gates need to be considered. A number of asynchronous simulators have been developed based on this concept. Authors of such simulators are E.G. Ulrich, A.R. McKay, F.H. Hardie and R.J. Suhocki, case et al, amongst others. Details of their work can be found in the literature\(^{14,10,4}\). Of these McKay's simulator is the fastest one, which uses a special purpose computer with dedicated hardware.

F.H. Hardie and R.J. Suhocki\(^4\) and R.E. Wolfe\(^\textbf{15}\) introduced the concept of using all bits of a computer word for processing multiple conditions. In this way Hardie and Suhockey's simulator can carry out thirty three cases in one pass of the simulation run. J.S. Jephson, R.P. McQuarrie and R.E. Vogelsberg in their paper\(^6\) and L. Shafer and B.H. Scheff in paper\(^{12}\) discussed the concept of using a third value—(in addition to 0 and 1)—called X-state, to represent an initial unknown state of the circuit. This is of importance in detecting the time required for a definite state (0 or 1) to be reached starting from initial conditions. Sometimes this is also important in detecting hazards.
1.6 SCOPE OF THE PRESENT WORK

It has been mentioned already that for short pulse width of the excitation signals, the propagation delays and setup delay should all be considered in analysis of the circuit. In this work asynchronous table-driven model has been selected as the basis for the simulation algorithm. Most of the important concepts discussed previously, such as inclusion of X-state, evaluation of only gates that might change, etc. have been included in the algorithm.

The work starts with a review of the available system models, logic elements and expression descriptions, input-output description and use of special purpose software and hardware techniques.

Next a review of timing specification of gates and flip-flops relevant to circuit analysis has been made. An algorithm has been developed taking into account the delay constraints. The tables which are manipulated by the algorithm have also been defined.

In the next step the system units that will be activated during simulation, conforming to the algorithm, have been defined. The system flow chart showing the interlinking among the system units are shown.

Finally, a computer program has been developed based on the algorithm and it is then applied to analyse two circuits.
CHAPTER 2
REVIEW OF SYSTEM MODELS
2. REVIEW OF SYSTEM MODELS

2.1 INTRODUCTION

All the gate-level logic simulation systems, so far in use, developed around a definite integrated structure suitable for communication with the logic designer and to translate input commands and expressions to an executable model. An evaluation of the systems show that they have two different sections. First one is the language definition section. This establishes the syntactic and semantic description for the logic elements and their basic interconnections. This also includes commands for simulation control, signal specifications and output control. Second one is the core of the system which translates the input statements to an executable model. The commands are translated to set simulation control and finally carry out the simulation.

In this chapter details of important simulation system characteristics are discussed.

At the end of this chapter, a short discussion on two general purpose simulation programs and their use in logic circuit design is given. Finally specialized computer techniques to solve the problem are also presented.

2.2 LANGUAGE DEFINITION AND COMMANDS

2.2.1 LOGIC ELEMENTS AND EXPRESSIONS

Two types of logic element and expression description can be noted among the simulators. Of the two types, the first
has a tendency to utilize a language derived from the Boolean circuit representations while the other uses a definite format form directly convertible from the logic diagram.

2.2.1 GATES

In pseudo-Boolean language format, gate operations are represented by mathematical operators, accents, or special characters, connecting the operands upon which predefined logical operations are made. Normal practice is to represent AND by a dot(·) or an asterisk(*) OR by a plus sign, and inversion or negation by a dash or apostrophe. But there exist languages which deviate from this normal practice. Examples of four different conventions, as used in four different languages to describe a two-legged NAND gate operation, are given below:

1. \[ C = '(A * B) \]
2. \[ C = *A + *B \]
3. \[ C = (A * B) \]
4. \[ C = (A.B)* \]

The first three representations \(^{12,9,8}\) do not follow directly the exact conventions used by digital logicians. The fourth convention \(^{11}\) is more direct with operation being written in their normal order. These form of gate expressions allow easy extension of Boolean equations to a large expression in a single statement \(^{12}\). This is shown in Fig. 2-1.
\[
\text{OUT} = \text{IN1} \cdot \text{IN2} + \text{IN2} \cdot \text{IN3}
\]

(a) Circuit
(b) Expression

Fig. 2-1 Use of Boolean expression to define large circuit. All these conventions are suitable for translating inputs to compiled code model form but introduction of delays are difficult.

In logic diagram translation form each logic gate with an explicit name and operands as inputs, outputs and delays are expressed in a single statement, which describes only one gate. This type of expression is suitable for table driven model and in cases where delays are to be considered. The following two expressions show this:

\[
\begin{align*}
1. & \quad \text{OUT} \quad \text{INV} \quad 0 \quad \text{AIN} \\
2. & \quad \text{N} \quad \text{AIN} \quad \text{BIN} \quad \text{OUT}
\end{align*}
\]

In the first statement the format is output-gate-delay-inputs and in the second gate-inputs-outputs. Sometimes function name followed by its arguments as in FORTRAN statements are used as a direct subroutine call. The same two-legged NAND gate discussed before may be expressed as \( G = \text{NAND}(A, B) \).

2.2.1.2 FLIP-FLOPS

Actually flip-flops are not basic gates, rather a combination of gates. Because of their frequent use like basic
gates with unique characteristics, many simulators declare them as basic gates. However their logic operations are defined in a somewhat complicated way. Various conventions are used in different simulators to express them, for example

1. TOGGLER/S = /TOGGLER*T
   TOGGLER/R = TOGGLER*T
2. JKFLIP(2,A,ABAR, IN1*IN2,A*IN4,PULSE,DCSET,DC_RESET)
3. FF1(FT,FF) = XX*B,AB+Y,CLK,DCR
4. S1FF1 = XX*B

The first expression denotes a toggle flip-flop which reacts to an input signal T(9). The S and R identify the set and reset functions which are stated in a Boolean form. In the second expression function subroutine notation is used(8). The subroutine called performs the function of a JK flip-flop, using as argument the number of outputs, the list of outputs, and the inputs for set, reset, pulse DC set, and pulse DC reset.

A more convenient notation(12) is expression 3. Here pseudo-Boolean format identifies the true output (FT), the false output (FF), the set signal (XX*B), the reset signal (AB+Y) the clock signal (CLK), and the DC reset (DCR) of circuit FF1.

The simulation in reference(11) uses a separate equation for each input terminal. A typical input equation for a set input is shown by S1, as given in expression 4. A reset input would be identified by R1.
The second form of expression for flip-flops uses definite format, and an example\(^{(5)}\) would be:

```
FF JINP KINP CLOCK DQSET DQRESET QOUT NQOUT
```

Here, the type of gate is defined first, followed by J input, K input, clock input, DC set and DC reset input.

### 2.2.1.3 MACRO AND BLACK-BOX EXPRESSION

Some simulators have provision to define and store commonly used circuit expressions (e.g. flip-flops, two stage counters) by the users or by the designers of the simulator. These expressions are stored in a library called macro library and the group of expressions under each definition is called macro or black-box transfer vector. These expressions can be inserted in the main expression stream just by calling their names. These macros help in easy coding of recurrent circuits. Sometimes a macro can call another macro. In the example of the Fig. 2-2, the logic of NAND flip-flop, denoted by RSFF, is being defined initially by detailed expressions and is later used in the two stage shift circuit by specifying only the inputs and outputs of RSFF. The explicit description of the flip-flop is not repeated. In turn, SHIFT as well as RSFF can be recursively nested in larger subsystems.
a) Circuit RSFF:

\[
\begin{align*}
&\text{RSFF:} \\
&\text{OUT/TRUE, FALSE; } \\
&\text{INPUT/IN1, IN2, IN3;} \\
&\text{LOGIC/TRUE} = '(\text{FALSE} \cdot \text{IN1}); \\
&\text{FALSE} = '(\text{TRUE} \cdot \text{IN2} \cdot \text{IN3}); \\
&\text{END}
\end{align*}
\]

b) Expression for circuit (a)

c) Two-stage shift circuit using circuit (a)

\[
\begin{align*}
&\text{*SHIFT} \\
&\text{OUT/SHIFTOUT;} \\
&\text{INPUT/CLOCK, P1, P2;} \\
&\text{LOGIC/} \\
&\text{E(E1, E2) = P2, P1, P1;} \\
&\text{B = CLOCK} \cdot \text{E1;} \\
&\text{C = CLOCK} \cdot \text{E2;} \\
&\text{D(SHIFTOUT) = B, C, C;}
\end{align*}
\]
ALLOCATE/
    E = RSFF
    D = RSFF
END

d) Expression for circuit C
Fig. 2-2 Use of macro in circuit expression.

2.2.2 SIMULATION COMMANDS

These are special purpose statements used to communicate with the simulator to control the simulation environment. These commands may be grouped under three headings:
a) Simulation control  b) Signal specification  c) Output. The commands used in the examples may be different for different simulators, but these functions are normally done by every simulator command repertoire.
a) Simulation control:

SIMULATE  Initiates the simulation.
EXIT      Signifies the end of simulation commands.
RESTART   Reinitializes the simulation program and circuit characteristics for the next run.

b) Signal specifications:

HIGH      Sets listed signals to logic-level 1
LOW       Sets listed signals to logic-level 0.
SCHEDULE  Causes: (1) Inversions of listed input signals at successive time increments; (2) Printout at the time periods indicated in parentheses after PRINT statement;
(3) Conclusion of the simulation at the time given in parentheses after STOP statement.

**DELAY**

Sets the delays of listed gates (ns). Default values are nominal values.

c) **Output**

**PRINT**

Prints the listed signals.

**CHART**

Specifies timing-chart on simulation printout (maximum number depends on the width of the paper)

**CHANGES**

Indicates a full-line of simulation printout is to occur every time a signal on the output list changes.

**NOCHANGE**

Opposite of changes; printout occurs only when explicitly specified.

The following example illustrates the use of some of the above commands:

HIGH/CLOCK, INPUTA

LOW/SIGNAL, INPUTB, GND

SCHEDULE/CLOCK(300,P), SIGNAL (100, 200, P)

INPUTA(1000, 2000, 1000)

INPUTB(1500, 1200, 1000)

STOP(4000)

PRINT/CLOCK, SIGNAL

CHART/

SIMULATE/

EXIT
The explanation of the statements are:

1. HIGH initially sets the listed signal CLOCK, and INPUTA to 1.
2. LOW initially sets the listed signals to zero; GND is a permanent ground.
3. SCHEDULE schedules timing information. CLOCK(300,P) generates a CLOCK signal with 300-ns (300 time unit) pulse width (alternately high and low).
4. INPUTA is a signal starting with high value and goes zero at 1000 time unit, then high at 2000+1000 time units etc.
5. STOP (4000) shows stopping of execution at 4000 time unit.
6. SIMULATE begins simulation and EXIT ends the simulation.

2.3 TRANSLATION AND SIMULATION

This section starts with checking of the input statements and commands. The error free statements are then translated to an executable form upon which simulation begins. Translation indeed depends on the form of the output code that is to be made. For example, in the conversion to compiled code model, the input expressions are normally Boolean equations and the output code that is to be made depends on whether data-dependent or data-independent code are necessary. For table driven model,
the tables are made depending on the simulation algorithm devised. After translation actual simulation begins and continues until stop condition is met. Data independent code model and table driven model have already been discussed (see chapter 1). Data dependent code model is discussed here.

2.3.1 DATA DEPENDENT CODE MODEL

Data dependent code model uses codes that takes advantage of the characteristics of the logic element. For example, the output of a three input NAND gate will be 1 if any one of its inputs is 0. The following instruction set takes advantage of this fact:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SON D</td>
<td>Store one in D</td>
</tr>
<tr>
<td>NZT A</td>
<td>A = 0? IF A ≠ 0, Skip next instruction</td>
</tr>
<tr>
<td>TRA OUT</td>
<td>Yes</td>
</tr>
<tr>
<td>NZT B</td>
<td>B = 0? Yes</td>
</tr>
<tr>
<td>TRA OUT</td>
<td>C = 0? Yes</td>
</tr>
<tr>
<td>NZT C</td>
<td>Store zero in D.</td>
</tr>
</tbody>
</table>
Thus if input A is zero, only first three instructions are executed. As a result of instruction bypassing, number of instructions actually executed in this case is less than the data independent coding (chapter 1), where all the five instructions would be executed.

2.4 SPECIAL SOFTWARE TECHNIQUES

Logic behaviour can be simulated with any general purpose simulator that models a discrete environment. Although usually applied to study system level behaviour, they are directly applicable to modeling digital logic. Using the general simulation language, it is possible to express the structure of the network, its behaviour, and the simulation conditions. By so doing, the advantage of the features such as timing controls, output formats, and diagnostics which have been built into the programming system, can be taken. Two much used programming systems, the general purpose system simulators and PERT can be used in logic design.

2.4.1 GENERAL PURPOSE SYSTEM SIMULATORS

Of the two general purpose system simulators, GPSS and SIMSCRIPT, GPSS is more suitable to logic design. GPSS(3), defines its basic entities as:
1. Transaction which move through the system, generating prescribed actions;
2. Facilities which are operated upon by transactions;
3. Blocks which provide the rules by which transactions flow;
4. Statistical queues on tables.

For logic simulation, these concepts represent signal pulses, the digital elements (gates, flip-flops), and the signals-paths of the network. To use GPSS, logic environment are described by GPSS language structures. From these statements, GPSS generates and monitors the flow of digital pulses through the circuit. Gates are put into an event que when their inputs change; and replaced when their outputs switch-state. GPSS clock is free-running, so timing of events is asynchronous and it advances according to the time at which next event is to occur. Delay values can be expressed as a statistical distribution. The output and report generation describes both system flow and state changes and provide summary statistics.

There are several drawbacks which actually precludes using it for digital simulation. Some of these points are:

1. It depicts a statistical model of a system and is too detailed to provide a valuable assistance in conceptual design. Each gate representing each block in GPSS takes much execution time.
2. Much understanding of the syntax and semantics is necessary for application in simulation.

2.4.2 PERT (Project Evaluation and Review Technique)

This provides basically a statistical estimate of end dates in large projects. A modification of this technique has been used to obtain worst-case delay which would not be easily available from a logic simulator[7]. This is valuable for a statistical estimate of a complex path to be critical. After a detection of critical path, the path can be studied in detail with a logic simulator.

2.5 SPECIALIZED COMPUTER TECHNIQUES

Normally simulators are operated on standard, general-purpose computers for logic simulation. But dedicated systems are also possible to carry out simulation. These computers have several processing units called cells which operate in parallel. Each cell acts as a small computer and contains a particular routine for a definite logic operation. These units also maintain the status of the signals. The controlling processor contains a system clock and maintains the list of future events to communicate with the appropriate cell. The controller activates the proper cell and provides input values for computation. The cells inform the controlling processor whether or not the elements
under their cognizance are actually switched. The controlling processor updates the clock and rearranges the events list.

A combined logic gate and register level simulation has also been suggested and implemented (10). This consists of four special purpose parallel programmable hardware elements and a general purpose computer. The special purpose hardware performs logic element simulation, while the general purpose computer performs register level simulation. The computer processes input and output statements and controls the hardware.
CHAPTER 3
DEVELOPMENT OF SIMULATION MODEL
3. DEVELOPMENT OF SIMULATION MODEL

3.1 INTRODUCTION

In this work it is proposed that, the algorithm developed will carry out extensive timing simulation, so that the monitored node states can be found at any time - at or around the input excitation signals' switching points. This model, therefore, considers the differences in propagation delays, setup delay for basic gates and the recommended timing specification for some selected flip-flops. In this chapter, first timing charts of basic gates and flip-flops are reviewed and then the tables and the algorithm for simulation are discussed.

3.2 TIMING SPECIFICATIONS

3.2.1 BASIC GATES

For most basic gates (AND, OR, INV, NAND, NOR gates) manufacturers specify two values of propagation delays: \( T_{PLH} \) and \( T_{PHL} \). These values are specified for a certain ambient temperature (normally 25°C) and at certain resistive and capacitive loading. Values for other environments can be obtained from propagation delay Vs. load and propagation delay Vs. temperature graphs. Some are shown in Table 3-1 for TTL gates by Texas Instruments.
Not only TTL gates, other logic families show this same type of characteristics and typical values for these are given in Appendix IV.

3.2.2 EXCLUSIVE-OR GATE

Exclusive-Or (XOR) gate is not a basic gate and its timing table (Table 3-2) shows that propagation delay for any one input signal setting depends on the state of the other input.

<table>
<thead>
<tr>
<th>Type of Delay</th>
<th>From</th>
<th>Other Input</th>
<th>Typ</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{PLH} )</td>
<td>A/B</td>
<td>low</td>
<td>12</td>
<td>18</td>
</tr>
<tr>
<td>( T_{PHL} )</td>
<td>A/B</td>
<td>low</td>
<td>39</td>
<td>50</td>
</tr>
<tr>
<td>( T_{PLH} )</td>
<td>A/B</td>
<td>high</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>( T_{PHL} )</td>
<td>A/B</td>
<td>high</td>
<td>42</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 3-2

However, these two values do not differ much (e.g. 12 and 14, 39 and 42 etc.) and consideration of the higher value will provide a reasonable analysis.
3.2.3 FLIP-FLOP GATES

Several types of flip-flops in their operational features (T, D, SR, JK flip-flops and D latch) are seen in use. Of these, JK flip-flop is widely used and many others can be made from JK flip-flops. Again, different types of JK flip-flop construction are found with different timing characteristics. In this work, some selected JK flip-flops are considered. The flip-flops considered in this work may be positive or negative clock edge triggered and also level triggered. For level triggered flip-flop the only restriction imposed is the time specification required for the clock access after Reset/Set input goes high. If the setup time is chosen to be zero, it means that the clock action on setting the flip-flop can instantaneously be activated as soon as the Reset/Set goes high. Another important consideration for using flip-flops is that, all the setting of output will take place by $T_{pHL}$ and $T_{pLH}$. This means that, only one value of each could be specified for a gate. But several gates are found having different values for $T_{pHL}$ or $T_{pLH}$ depending on other input signal conditions (see section 3.2.2). Examples of flip-flop are 70, 72, 73, 76, 107, 74, 111 etc. of 54/74 series. Some of their timing specification are given in Appendix IV.
3.3 SPECIAL CONSIDERATIONS ON TIMING DATAS

From the typical timing data for basic gates it is evident that for some gates $T_{PLH}$ can be even twice as much higher than $T_{PHL}$. In these cases if $T_{su}$ becomes less than $T_{PHL}$ then it may be possible to cancel out the effect of one signal variation if the excitation signal changes state before $T_{su}$. To illustrate this, let us assume that an inverter gate has $T_{su} = 3$, $T_{PLH} = 6$ and $T_{PHL} = 3$ time units, and its output is going from low to high. After 3 time units, if the excitation signal changes state so as to make the output high to low, the output will not go ultimately high. So, for each excitation signal to have an effect on the output, $T_{su}$ should be greater than $T_{PHL}$ by one time unit.

The values stated for flip-flops show that if the input signals conform to the recommended specifications, the flip-flop will operate properly. Otherwise the flip-flop may enter an undefined state. Eight time units are important for guaranteed operation of the JK flip-flops under consideration, with asynchronous Set and Reset inputs. These time units are minimum low pulse width for set and reset, minimum clock high and low pulse width, setup and hold time for $J$ and $K$ input, and propagation delay times $T_{PLH}$ and $T_{PHL}$ for $Q$ and $Q'$. 
3.4 CONSIDERATIONS FOR ALGORITHM DEVELOPMENT

The use of time units with respect to gate simulation is illustrated in Fig. 3-1 for basic gates.

![Diagram](image)

**Fig. 3-1**

Here two AND gates are cascaded with same timing characteristics. Both the gates have \( T_{su} = 3 \text{ t.u.} \), \( T_{pu} = 2 \text{ t.u.} \), and \( T_{pl} = 4 \text{ t.u.} \). A, B, and C are three-input periodic test signals with periods 3 t.u., and their initial values are 1, 1 and respective.

From these informations it is evident that at least three tables are necessary. First table will contain gate types, inherent timing data \( (T_{su}, T_{pu}, T_{pl}) \) and the signal node names associated with each gate. The second table will contain signal states of all signals. And the third table will contain information for switching time points for input excitation signals.

Now at the start or at zero system reference time (SRT), A, B, C have values 1, 1, 1 and D, E have undefined value X. To condition outputs D and E, the inputs A, B, and C are to be entered for the relevant gate. Since, at SRT = 0, inputs to
the 1st gate are 1 and 1 but output will appear not at SRT=0. So, input states at SRT = 0 are to be remembered. To decide at what SRT output goes to 1, it is seen that D is in X (either 1 or 0) state and D will surely go to 1. If D were 1, no need of output setting would be necessary. But it may be 0 as well. So, a reference time called delay reference time (DRT) field of time unit \( T_{PLH} \) will provide a safe result. This time is also to be remembered. To check that the time difference between state changes of a signal conforms to the setup time, another DRT for setup time is necessary for each input signal to a gate.

For the second gate, C is known and D is unknown. C is 1 but D may be zero or 1. Since C is 1, it will try to set the output high. Therefore, it is better to set DRT for C at \( T_{PLH} \) (because D may be 0). This consideration should be made independent of other inputs because these may not change at the same time. Now after \( T_{PLH} \), A, B, and C go to 0.

To set this conditions to the gates, it is evident that the effect of the first variation has not yet reached the output, although this variation does not violate \( T_{SU} \). So, another field for state and DRT is necessary for each input signal.

Here DRT is selected by noting that whatever the value of other inputs, the AND gate output will go to 0 if any input is 0.
So, DRT for A and B is made equal to $T_{PHL}$. These states and DRTs are defined by BUFFER ONE (B1), DELAY ONE (D1), BUFFER TWO (B2), DELAY TWO (D2). Setup time for each signal is called reference setup time.

For the second gate, BUFFER TWO will contain 0 and DELAY TWO will be $T_{PHL}$. When SRT becomes 4 t.u., D1's of A and B times out and indicates that the instant to take an action to set the output has been reached. Now, looking at the B1's of inputs is difficult for setting the output because, some D1's may not be in a time out state. Therefore, another buffer called value to be used is necessary for every input. At the time out of D1, the value of B1 is to be pushed into the value to be used buffer. It should be remembered that the true state of any signal will reside in the signal states table. And the buffers contain image states.

It is simple to look at the value to be used buffer for input state and a decision can be made to set the output. For the first gate, this will be 4. For the second gate, though, time out condition for C input has already been reached and the output can not be set because although the D input at SRT = 4 is 1, time out condition for D has not yet been reached. Now, D input condition for the 2nd gate is set. Time out condition for each input is made by decrementing all DRT's.
upto zero at every increment of SRT. This decrementation is called reference time decrement and setting of output because of time out/called iteration to set output.

Searching value to be used of inputs of a gate can be made short by checking each input value. This is done by taking advantage of the characteristics of the gates, that is, output will always be 0, 1, 0, 1 for basic gates (AND, NAND, NOR and OR) for any input 0, 0, 1, 1 respectively, irrespective of other inputs. The above discussion of signal setting is shown in Fig. 3-2.

For flip-flop gates, inherent time data (Reset setup time, set setup time, clock low pulse width, clock high pulse width, setup delay for J and K input, hold delay for J and K input, propagation delay $T_{p LH}$ and $T_{p HL}$ for Q and $\bar{Q}$ signal) and signals (Reset, set, clock, J, K) with only one reference time (DRT) for each signal is necessary to validate their inherent time specifications. But it requires two buffers for state and delay for each output. When an input signal changes state, it is first tested to see if DRT is zero. If this is not so, ambiguous operation of the flip-flop will occur. A valid change of input sets Q and $\overline{Q}$ with logical value setting.

Logical setting means setting Q and $\overline{Q}$ by consideration of the truth table at the time of reference.
<table>
<thead>
<tr>
<th>Gate type</th>
<th>T_{Sa}</th>
<th>T_{DHL}</th>
<th>T_{DLH}</th>
<th>No. of input signal</th>
<th>value</th>
<th>Ref. setup time</th>
<th>BUF1</th>
<th>DEL1</th>
<th>BUF2</th>
<th>DEL2</th>
<th>Input signal</th>
<th>value</th>
<th>Ref. setup time</th>
<th>BUF1</th>
<th>DEL1</th>
<th>BUF2</th>
<th>DEL2</th>
<th>Output signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 3 2 4 2</td>
<td>A</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>X</td>
<td>0</td>
<td>B</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>X</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>C</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>X</td>
<td>0</td>
<td>D</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>A</td>
<td>X</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>0</td>
<td>B</td>
<td>X</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>C</td>
<td>X</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>0</td>
<td>D</td>
<td>X</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>0</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>A</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>B</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>C</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>D</td>
<td>X</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>X</td>
<td>0</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>C</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>D</td>
<td>X</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>X</td>
<td>0</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND 3 2 4 2</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A conflict will appear at setting D, because its setup time has not timed out.

To avoid this, setup time should be made short, so that this change of D can be pushed to BUF2.

**Fig. 3-2 Simulation example**
All the above discussion for setting output are to be carried out sequentially. It is seen that, each gate is completely independent of other gate settings and hence gates may be placed in any order. No signal-setting of input to any gate does make an instantaneous setting of output. So, a sequential setting of input conditions to all connected gates, a sequential decrementation of all DRT's of all gates, a sequential iteration to set the output, all occurs independently and only one type of operation occurs at one time. This concept makes the algorithm very simple and automation of simulation is simplified.

Now to simplify the algorithm and to reduce the time of simulation some points are important. Input signals may change slowly, so a setting of input may take time even after time out of DRT's to a particular gate. This point can be used to avoid checking of DRT's of a particular gate by attributing a gate reference time decrement flag (called gate decrement flag). If this flag is reset, checking for decrementation can be avoided. All DRT's of all gates may become zero. In that case, checking any gate for DRT decrementation can be avoided by introducing another flag called system decrement flag.

Another point to note is that, time out for iteration to set output occurs infrequently. So, an iteration flag
for each gate helps avoiding unnecessary checking of all gates. A 'system iteration flag' will completely avoid checking of any gate for iteration.

3.5 MODEL SELECTION

From the discussion in section 3.4, it is evident that a huge information are to be manipulated for complete asynchronous simulation. Therefore, table driven model has been selected for this study.

3.6 TABLE-GENERATION FOR SIMULATION

After checking the input data stream, the simulator translates these data to generate several tables. Values and significance of the tables and table entries implemented here are discussed below:

TABLE 1:

This table keeps data for 1) basic gates:

- a) Type of gate and its status
- b) Setup time
- c) \( T_{pL} \)
- d) \( T_{pH} \)
- e) Number of inputs
- f) Input signal number along with its status
- g) Output signal number

For 2) Flip-flop gates:

- a) Gate type and its status
b) Setup time for Reset, Set, J and K inputs

c) Hold time for J and K inputs

d) Clock high and low pulse width duration

e) Reset, Set, Clock, J input and K input signal number with respective delay reference time field

f) Q and Q̅ signal number

g) Q and Q̅ signal values and flags

h) Clock access DRT

The explanation of the data items for basic gate is:

a) The type of basic gate means a code name for any of AND, OR, NAND, NOR, INVERT, XOR gate. The status contains gate decrement flag and gate iteration flag

b), c), d) and e) are self-explanatory

f) Signal number is a unique identification number particular to a signal. Status indicates the behaviour of this signal in relation to this particular gate. Status contains information like value of the signal to be used to set output at the time of reference, setup delay to indicate minimum stay of a signal, queued value one (B1) and the propagation delay for this value (D1), queued value two (B2) and the propagation delay (D2) for this value. An iteration flag is
associated with each input to show whether the input in consideration can take part in output signal setting.

g) This shows a unique identification number for a signal which is the output of this gate

The explanation for flip-flop gate data items are:

a) This shows the type of gate as either positive or negative clock edge triggered flip-flop. Status has flags like gate decrement flag and iteration flag. It has also a special flag which shows whether the Clock is allowed to set outputs. This is because Set/Reset activity prohibits clock action.

b), c), d) are self-explanatory

e) These are unique identification numbers for each input signal with their respective reference time of transition indicator.

f) Self-explanatory

g) Two queued state and delay reference time values for each $Q$ and $\overline{Q}$ signal along with one iteration flag for each output. Iteration flag points that setting time for $Q/\overline{Q}$ is reached.

h) This delay reference time indicates that a clock can set output after timeout of this field. The field is set after Set/Reset goes high from low.
TABLE 2:
This keeps data of the input excitation signals. This contains information like initial value, switching time points etc. Signals may be periodic, non-periodic or constant d.c. level.

TABLE 3:
This keeps time for periodic printing.

TABLE 4:
This keeps time for stopping simulation. When system reference time equals this end time, simulation stops.

TABLE 5:
This keeps data for input-output relation for gates AND, OR, NOR, NAND. This means, e.g., a zero input to an AND gate makes output zero irrespective of other input states.

TABLE 6:
This keeps states of all signals updated at the time of reference as 0, 1 or X.

TABLE 7:
This keeps number of gates and the gate number driven by each signal.
TABLE 8:
This keeps signal number and name of signals that are to be monitored at simulation.

TABLE 9:
This keeps variable name in order of their first occurrence in the circuit description.

3.7 TIME MAPPING FOR EVENT CONTROL
Time mapping means that time is quantized, i.e., any event such as, change of input test signal, change of output signal of a gate etc. can only occur at the end of a quantized period. Time is thus quantized into constant &t time intervals; each instant of time \( t_1, t_2, t_3 \ldots \ldots \ldots t_i, t_{i+1} \ldots \ldots \ldots t_n \) at which an event can occur is defined by \( t_{i+1} = t_i + \Delta t \).

3.8 ALGORITHM FOR SIMULATION
This algorithm manipulates data in the table discussed in section 3.6 and ultimately carries out simulation.

Step 1: Reset system reference time and all the flags.
Step 2: Initialize signal states table by input test signal and set levels and delay reference time, (B's and D's) of all connected gates. Set the system and gate decrement flags.
Step 3: Check to see if printing is necessary. If not, bypass this step.
Step 4. Check to see if system reference time equals end time. If so, stop simulation, otherwise continue.

Step 5. Increase system reference time.

Step 6. Check to see if system decrement flag is set. If this flag is reset, by-pass this step. Otherwise reset this flag and start checking the individual gate decrement flag commencing with the first gate. If an individual gate decrement flag is set, reset it and decrement all the delay reference time. If any field still remains non-zero, set the individual and system decrement flags. In this way check all gates up to the last gate. For flip-flop gate, if clock access delay reference time becomes zero at the time of decrement, reset the clock access flag.

At the time of decrement of delay reference time of inputs for basic gates and $Q$ and $\bar{Q}$ for flip-flop gates, if any delay reference time becomes zero, set the system and individual iteration flags.

Step 7. Check to see the condition of system iteration flag. If it is reset go to step 9. If it is not, reset it and start checking individual gate iteration flags. If any individual gate iteration flag is reset, bypass the gate, search the next one and continue up to the last gate. If any individual flag is set, reset it and try to establish an output from the input conditions for basic gates. Check to compare this output state for basic gates, or $Q$ or $\bar{Q}$ for a flip-flop gate, to their previous states. If these values are
different, place the signal number and the value in a transient buffer. Continue checking & storing in this way up to the last gate.

Step 8. Take the signal number and level from the transient buffer area of step 7 and set the value and delay reference time for all connected gates. Set also the signal states in the signal state table. Set the system and individual gate decrement flags. Repeat this for all the signals in the transient buffer area.

Step 9. Check to see if any input test signal requires change of its state. If any switching time points occur, set the value and delay reference time for all connected gates. Set the signal state in the signal state table. Set the system and individual gate decrement flag. Repeat this for all input test signals. Go back to step 3.
CHAPTER 4

SYSTEM ORGANIZATION
4. SYSTEM ORGANIZATION

4.1 INTRODUCTION

The simulation system proposed and implemented in the present work has two distinct phases. The first phase defines the logic elements, their interconnections and command description to communicate with the simulation software. The second phase is the simulation process that inputs information defined by the first phase and carries out the simulation. Only the simulation process subphases will be discussed in this chapter. The description of the first phase is given in appendix 1. The state diagram of the simulation process is given at the end of the chapter (Fig. 4-1).

4.2 PROCESS ORGANIZATION

The simulation process begins with the data inputs describing the logic circuit and the commands. The commands actually define the type of data sets and guide the simulation software to move through the input data streams to carry out the intended work. In this system, data and commands are kept completely separate. Each command invokes a particular unit (routine) to process data and take necessary actions. A number of distinct execution units are called into play in the simulation process. Their logical organization and functional characteristics are discussed here.
4.3 UNITS OF SIMULATION PROCESS

The different processor units of the simulation system perform one or more of the following functions:

1) Checking for syntax errors and issuing messages if any errors are detected; 2) Generation of different tables from the error free data; 3) Carrying out simulation and printing simulation status acting on the generated tables. These functions are distinct and are normally done by one particular unit, but some units may carry out two functions simultaneously (e.g. (1) and (2)) whenever it seems suitable to combine two functions. Accordingly these units are grouped as type (I) type(II) or type(III) units.

4.3.1 GROUP (I) UNITS

4.3.1.1 BASIC-INTERFACE UNIT (BIU)

This is the first unit that initiates the communication with the input data stream and other simulation units. It detects the first character of an input data record and determines whether it is a command, comment or an invalid data record. If it is a comment card, it just puts the information in the card to the output list file. If it detects an invalid starting character (if not a comment or a command character), messages are issued to indicate that. If it is a command card it transfers control to the command processor.
4.3.1.2 COMMAND PROCESSOR (C.P)

It detects, for a command card, the existence, format, validity, repetition and the order of this command. The order means that, some commands must appear in a definite sequence. For example, a macro command must appear before a circuit description command. If any errors are found, control is given over to the BIU with necessary syntax error messages. If a valid command exists, control is directed to the particular unit for processing that particular command. These units are defined as data set processors.

4.3.1.3 DATA SET PROCESSORS (D.S.P)

These units check and process each particular type of data that follows the command.

Name processor: This routine processes the name of the circuit which is to be simulated. At the output list, the name of the circuit helps in identification of the simulated circuit. At the end control is given to BIU.

Option processor: This unit checks the proper format of the option operands, validity of the operand, repetition of and the operand sets the option flags depending on the operand(s) specified. The meaning of these operands can be found in appendix- 1. At the end control transfers to BIU.

Macro processor: This checks and stores a correct macro definition, operands and delays of macro definition,
and circuit description defined by the macro in macro library. This also checks duplication of macro name. The end of macro processing is detected by a command card and control is transferred to command processor.

Statement format processor: This checks each statement describing the circuit. It checks valid gate identification, delays and I/Os. Input-output shorted for basic gate is also detected. From these statements signal table and gate-input-output table are generated. It also accepts macro call and searches library and if found, the circuit description defined by the macro definition is expanded with replacible operands. Error messages are issued if any errors are detected. At the end, if it is a normal end, control is given to BIU. If, however, a command is found control goes to the command processor.

Input test signal processor: This unit checks signal name with the signal table to whether this signal is connected to any gate. This also checks the initial value of the signal and the signal waveform. At the time of checking, if no errors are found, a table of input test signal is also created. This routine can handle any form of signal waveform — d.c.-level, periodic and nonperiodic. This routine ends with the detection of a command card, where control is transferred to the command processor.
Print signal processor: This unit processes the signals that are to be monitored during simulation. A checking of these signals is made to see whether this signal is connected to any gate. A table is generated from these signals. At the end control is given to BIU.

Print time processor: This checks and generates a table of time at which printing of the signal states are to be made. Only a periodic time specification is allowed. At the end control is given to BIU.

End time processor: This is the last routine of the first group. This checks and stores the end time which is the termination time for simulation. At the end if no syntax error(s) are detected and the execution option flag is set, control is given to group (II) routines. Otherwise, simulation stops.

4.3.2 GROUP (II) UNITS

These units are used to generate tables for simulation. At the beginning, some prinout of circuit statistics is made. These units are always executed and are option independent.

4.3.2.1 SIGNAL DRIVING GATE TABLE GENERATOR

This unit generates a table (TABLE 7) that contains the number of gates and the gate number driven by each signal.
4.3.2.2 SIGNAL STATE TABLE GENERATOR

This table keeps status of all signals (initial values for the signals are set to X).

4.3.3 GROUP (III) UNITS

These units actually carry out simulation by acting on the tables generated by GROUP (I) and GROUP (II) units.

4.3.3.1 INITIAL CONDITION SETTER

This clears system reference time, resets system decrement and iteration flags. This also sets the level and delay reference time in Table (1) for the input test signals.

4.3.3.2 LEVEL-DELAY SETTER

This unit is called by other units when a change of signal states takes place. This unit, supplied with signal number and level, sets the level and delay reference time of all gates driven by the relevant signal. It also sets the gate decrement flag of all the driven gates. At the time of setting, this precisely checks the set-up time for both basic and flip-flop gates. Any abnormal condition detection is instantly notified. If signal specification is not met, execution stops and error messages are issued, containing the gate number at which a signal (its name is printed) fails to confirm to proper timing and also the time of reference of the error.
The level in the signal states table is also updated. At the end it returns control to the calling unit by setting the system decrement flag. If the option for continuous printing is not provided then this signal is checked with print signal table. If a match is found a print flag is set.

4.3.3.3 PRINTOUT UNIT

This unit first checks the option for continuous printing. If continuous printing is needed, it checks the print time table. If time for printing is reached, a states printout in both truth table and wave forms is listed. The print time table is updated at the same time. If continuous printing option is not asked for, then print flag is checked to see if any signal of print signal table has changed state. If the flag is set, printing is performed and then flag is reset.

4.3.3.4 TIME OUTER

This checks the system reference time with end time. If a match is found, simulation stops.

4.3.3.5 SYSTEM REFERENCE TIME INCREMENTER

This increases the system reference time by one time unit.
4.3.3.6 GATE REFERENCE TIME DECREMENTER

This routine first checks the system decrement flag. If this flag is reset, it transfers control to the update signal unit. If not, the flag is reset. Then the routine starts checking individual gate decrement flags. If any individual gate decrement flag is reset, the gate is bypassed and the next gate is checked. If any gate decrement flag is set it is reset. It then starts checking the reference times for decrement associated with that gate. If any non-zero reference time is found, it is decremented by one time unit and checked to see if it is still non-zero. If non-zero, individual as well as system decrement flags are set. If zero, three types of actions will be taken depending on the types of time fields.

If it is the time field for an input signal for a basic gate, the individual gate and system iteration flags are set. If it is the time field for $Q$ and $\bar{Q}$, of a flip-flop gate, iteration flags are set. If it is the clock access delay reference time of a flip-flop gate, clock access flag is reset. All other reference times are tested and a similar action is taken. The process is repeated for all the gates.

4.3.3.7 OUTPUT DETERMINATION UNIT

This unit first checks the system iteration flag. If this flag is reset, it transfers control to the update signal unit
If not, it is reset. Then it starts checking individual gate iteration flags. If any individual gate iteration flag is reset, the gate is bypassed and the next gate is checked. On finding a set condition for any gate iteration flag, the relevant flag is reset and the type of gate is noted. For a basic gate, the unit tries to establish a output from the value to be used states of all inputs. A successful establishment of the output makes a further check of the state of the output signal in the signal state table. If these two values differ, then the output signal number along with the new states is stored in a transient buffer area. For flip-flop gates, the $Q$ or $\bar{Q}$ state is checked with the signal state table. A detection of difference makes the unit store the signal number and value in the transient buffer area. No storing is made if no difference is found. In this way all gates are checked and control transfers to the output setter unit.

4.3.3.8 OUTPUT SETTER UNIT

This unit calls level-delay setter unit for each signal placed in the transient buffer area. At the and control is given to update signal unit.

4.3.3.9 UPDATE SIGNAL UNIT

This unit checks all the input test signals for any change of states of the signals that may have occurred
as a result of switching time point conditions. If a change of state is noted, this unit calls the level-delay, setter. Status of the test signal in the input test signal routine is also updated. At the end of all signal setting and updating, control is transferred to the print out unit.

4.4 STATE DIAGRAM OF SIMULATION PROCESS

The different units discussed in section 4.3 are called into play when they are needed for different phases of the simulation process. The transition of system flow among the processing units is shown in the state diagram of Fig. 4.1
Option EXECUT and no syntax error(s) detected.

Table generator

Entry to actual simulation

Initial condition setter

Printout unit

Time outer

If end time reached

System reference time incrementer

Level-delay setter

If error

STOP

Gate reference time decremener

Output setter

Update signal unit

Fig. 4-1 state diagram of the simulation system.
CHAPTER 5
SOFTWARE DEVELOPMENT AND APPLICATION
5. SOFTWARE DEVELOPMENT AND APPLICATION

5.1 INTRODUCTION:

The algorithm developed and most of the concepts discussed in the previous chapters have been implemented as a computer program. The communication with the program --- the commands and circuit description procedure --- is described in Appendix-1. Error and warning messages and their meanings are mostly self-explanatory. In this chapter a very short description of the program is given. Test run with this program to analyse two circuits are also presented.

5.2 LANGUAGE SELECTION AND PROGRAM DESCRIPTION

The program developed works like a pseudo-language. The units which constitute the simulator have been described in section 4.3. The tables organized in section 3.6 by description are represented in tabular form in Appendix II. In the first phase of this process validity checks are made on the input data. The input data may consist of a single or of multiple characters. The tables generated do not contain large or complex numbers but rather simple and small integers. A good number of status flags are maintained in the process. The program actually manipulates a huge amount of data and status of the table and does not involve any complex arith-
The work is very specialized and is required to be carried out fast. These facilities can be effectively extracted from ASSEMBLY language. Therefore, ASSEMBLY language has been chosen to develop the program.

The program checks each input card and issues error/warning messages under the corresponding line of the program listing. All the input data are checked sequentially up to the last card. At the time of checking some tables are also generated.

The description may appear in any order except that some data, which have precedence, must appear in a definite sequence. The last card is the end time descriptor after which the program flow does not search any further input card. The flow chart of the simulation system is given in Appendix-III.

The facilities that provide the power of this program is evident from the commands and circuit description techniques. The formats of the data are strictly fixed which makes a faster checking and therefore requires short time for simulation. The program can simulate as many as 32 K gates and nodes and 127 storage macros, provided that sufficient is allocated. All the delays must be less than 127 time units. These are sufficient to test most of the simple circuits.

5.3 CIRCUIT TESTING BY THE PROGRAM

The program developed is applied to two circuits to
demonstrate its effectiveness. The first circuit is an UP/DOWN counter. The circuit is shown in Fig. 5-1(a).

To simulate this circuit, it is first necessary to break the circuit diagram as a set of input and output nodes acted upon by standard logic gates. The circuit description expression along with other test condition descriptions are given in Fig. 5-1(b). Signal cross-reference (Fig. 5-1(d)), fan-out list (Fig. 5-1(c)), system truth table (Fig. 5-1(e)) and signal-versus-time (Fig. 5-1(f)) show the various options obtainable for the circuit, for documentation and analysis.

At first, a periodic clock of period 30 time units has been applied to the circuit. Reset input/been made zero to set the counter output QOUT, QQQQ2 and QQQQ1 to zero. In the printout of Fig. 5-1(e/f) it is seen that error occurs at time unit 30 at gate number 6 because of 'setup time short'. This means that the clock changes state before settling down of the J input. Now J and K inputs are NNN11, and to see why NNN11 'setup time short' occurs, this signal name is entered as one of the monitored signal. Fig. 5-1(g) shows that J input is in the undefined state up to 29 t.u. And perhaps, it changes at 30 t.u. when the clock pulse also occurs.

Therefore, it is evident, that clock must change state at least 30+20 t.u. later from zero system reference.
time. To see whether it actually requires that much time, a period of 39 t.u. has been chosen. But, even then the output shows an error condition (Fig. 5-1(h)). Clock period is then made 50 t.u. and it is seen in Fig. 5-1(i) that, trouble with the 'setup time' does not occur any more.

A close look at the circuit will make it clear why setup time J was short. The output of gate 2 changes to 0 at 18 t.u. and the output of gate 3 appears at \((18+6)=24\) t.u. Therefore, \(N\) goes to zero at \((24+6)=30\) t.u. Now, for this flip-flop, at least 20 t.u. are to be allowed for the changes to settle down. This requires \((30+20)=50\) t.u. so that clock must appear at or after 50 t.u.

Several points can be noted from Fig. 5-1(e). After the clock changes to 1, the output of flip-flops of the circuit will go to 1 from 0 after 27 t.u. \((T_{PLH})\) and to 0 from 1 after 18 t.u. \((T_{PHL})\). At time unit 150: the clock input sets the gate 2 and 6 outputs to 0 and respectively. Since \(T_{PHL}\) is less than \(T_{PLH}\), from time unit 168 to 176 both output remains 0. Therefore, certain precautions must be taken to avoid this consequence. We can not get a continuous counting up or down with this type of flip-flops. Only those flip-flops that have equal \(T_{PLH}\) and \(T_{PHL}\) will give continuous counting output.
Fig. 5-1(a): Synchronous binary up/down counter.
Fig. 5-1(b) Listing of circuit description and signal specification.

Fig 5-1(c) Fan-out list.

Fig 5-1(d) Signal cross-reference list.
Fig 5-1(e) System truth table.

Fig. 5-1(f) Signal Vs. time plot.
Fig. 5-1(g) Test simulation demonstrating timing error condition (Clock pulse width 30 t.u.)
SYNTAX ANALYSIS FOR CIRCUIT DESCRIPTIONS AND COMMANDS BEGINS ***

1) START OF DATA
2) OPTION: EXECUTE COUNT
3) EXECUTE UP-DOWN COUNTER
4) ERASE
5) INPUT B N O N
6) INPUT B N O N
7) INPUT B N O N
8) INPUT B N O N
9) INPUT B N O N
10) INPUT B N O N
11) INPUT B N O N
12) INPUT B N O N
13) INPUT B N O N
14) INPUT B N O N
15) INPUT B N O N
16) INPUT B N O N
17) INPUT B N O N
18) INPUT B N O N
19) INPUT B N O N
20) INPUT B N O N
21) INPUT B N O N
22) INPUT B N O N
23) INPUT B N O N
24) INPUT B N O N

000 SYSTEM TRUTH TABLE FOR UP-DOWN COUNTER

<table>
<thead>
<tr>
<th>TIME</th>
<th>E</th>
<th>R</th>
<th>C</th>
<th>O</th>
<th>Q</th>
<th>U</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

ERROR: J SETUP TIME SHORT
MESSAGE: SIGNAL CLOCK FAILED TO COMPLETE PROPER TIME SPECIFICATION AT GATE NUMBER 6 AT TIME 39.
ABNORMAL SIMULATION TERMINATED

*** END OF THIS SIMULATION RUN

Fig. 5-1(h) Test simulation demonstrating timing error condition (Clock pulse width 39 t.u.).
Fig. 5-1(i) Simulation for proper counting operation.
(Clock pulse width 50 t.u.).
In Fig. 5-1(i), both up and down counting is shown. The computer printout shows a message like 'CLOCK ACTION IS SUPPRESSED FOR Q'. Now looking at the system truth table, it is evident that when the clock goes to 1, Reset goes to 0. Since in the test signal description, clock has been defined before the Reset; clock action has been considered first. It sets the level and delay reference time by considering J and K inputs. So, whenever Reset goes to 0, it sees the non-zero delay-reference time for Q/Q and the clock in a high state. Therefore, it cancels out the clock action.

In the second example, a binary FULL-ADDER, used in several IBM general purpose digital computers(20), is taken for analysis. The circuit is shown in Fig. 5-2(a) and the program for the circuit is shown in Fig. 5-2(b). Looking at Fig. 5-2(c) it is evident that the response at SUM and CARRY outputs appear at different times depending on the input conditions and the node conditions. For example, for the first time for all inputs 0 condition, the output appears at 11 t.u. But for a CCCCC input 1, the results appear at 29 t.u., i.e. 21 t.u. later!

A manual analysis will show the same thing. The states of other nodes printed will help manual tracing of signals. It is interesting to note that, when SUM goes to 0,
Fig. 5-2(a): Full adder used in IBM machines.
Fig 5-2(b) Listing of circuit description and signal specification.

Fig 5-2(c) Fan-out list

Fig 5-2(d) Signal cross-reference.
**Fig. 5-2(e) System truth table.**
Fig. 5-2(f) Signal Vs. time plot.
Fig. 5-2(g) Test simulation demonstrating addition operation.
CCCCC input has already changed thrice by this time. This fast changing of signal actually complicates the picking up the necessary output. This run finally results in an error. This is because the model is very sensitive to setup time. Any anomaly in setup time results in an error condition and the simulation stops. It can be noted that, setup time for gate 7 is 7 t.u. The difference in propagation delays $T_{PLH}$ and $T_{PHL}$ results in zero excitations to gate 6 by input 11111, 22222, 33333 at time 48. Therefore a short spike of one time unit duration excites gate 7 and results in an error condition. To avoid this consequence, zero setup time can be set. But then, the last excitation will be cancelled by the same input signal if both buffers are filled up.
CHAPTER 6
CONCLUSION AND SUGGESTION FOR FUTURE WORK
6. CONCLUSION AND SUGGESTION FOR FUTURE WORK

6.1 CONCLUSION

A detailed study of gate-level logic simulation, particularly the effect of delays, has been presented. The algorithm developed is very effective when the circuits are to be studied with delays. The algorithm developed is simple and, when implemented, can accommodate varieties of logical approach to circuit analysis. It is inherently time-dependent and therefore can not analyse zero-delay networks.

The program developed based on the above algorithm is simple to use and can simulate circuits with various options. The circuit description can be entered by looking at the circuit diagram and change of gate elements can be easily effected. This helps in fast buildup of the circuit. The program can simulate circuits almost like a synchronous simulator by attributing one time unit for every delay.

Actually, the program developed is an attempt to study logic circuit timing characteristics. Special attention has been given to develop the whole system —— converting the user-oriented circuit descriptions and commands to the tables and finally carrying out the simulation. Because of its distinguishing nature for interpreting delays, it is very sensitive to timing parameters. In setting any one input, the level
delay setting unit of the program does not consider the other inputs. This can, in some cases provide faulty result for OR and NAND gate, and advocates the use of equal propagation delays ($T_{pLH} = T_{pHL}$) for these gates.

The program sometimes issues messages that need careful attention for interpretation. For example, let us consider that at one time J input and clock are changing state. Now, if clock is going to set the state and delay reference time first, it will perhaps see that J input reference setup time is zero. After that, when J goes to set the state and delay reference time in table 1, an error message will probably be issued like "J changed before setup/hold delay". But, if J comes first, and then the clock, the message issued will be: "J setup time short". However in both cases simulation stops.

6.2 SUGGESTION FOR FUTURE WORK

Future work on logic circuit simulation could concentrate on the following:

1) The program can be made much more user oriented and commands for reinitialization of the circuit and test conditions can be adopted.

2) Routines can be added to process other flip-flops and JK flip-flops with other time specifications.
3) A preliminary consideration of all input settings to a particular gate can avoid the situation that arises for OR and NAND gates.

4) Routines for wired logic and delay lines can be added.
APPENDIX I

COMMUNICATION WITH THE SIMULATOR
The commands to communicate with the simulator program are classified into two groups.

GROUP A: These commands direct the control to routines that take care of the succeeding data set of the circuit description and test conditions. For example, a macro command invokes the macro processor and all the succeeding cards define the macro and the circuit under the macro name.

GROUP B: These commands contain information on the command card itself. For example, the card $ OPTION EXECUT is an option command card and the options are stated directly with the command.

The following syntax are followed for writing the commands:

1) Every commands must start with a $ (dollar)

2) Words with capital letters indicate the command itself. All commands must start after a $ separated by one blank.

3) A card starts with an asterisk (*) is assumed to be a comment card and may appear anywhere in the data stream except after a continuation character.
4) Most of the commands can appear in any order except when one data set is dependent on the preceding data sets. For example, a macro command must precede a statement format command.

5) All time fields must be filled up with full characters. No blanks are allowed.

EXPLANATION AND FORMATS OF THE COMMAND

GROUP-A COMMANDS:

<table>
<thead>
<tr>
<th>Command name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $ Y MACRO</td>
<td>This defines that all data following this command are for macro definition and statements.</td>
</tr>
<tr>
<td>2. $ Y SFORMAT</td>
<td>This is the statement format. All data after this command are statements to define circuit.</td>
</tr>
<tr>
<td>3. $ Y ITSIG</td>
<td>All data following this command are for test signal characteristics description.</td>
</tr>
<tr>
<td>4. $ Y PSIGN</td>
<td>The cards following the command specify which signal states will be printed for monitoring.</td>
</tr>
</tbody>
</table>

GROUP-B COMMANDS:

1. $ Y OPTION $ n1 $ n2 .........

where n1, n2 ......... may be one of the following:

- STTAEL = generate system truth table of the monitored signal.
- SGPLOT = generate signal-versus-state of the monitored signal.
- FNLIST = provides fan-out listing of each signal.
- XREFEN = provides cross-reference of each signal.
CONTON = specify that printing of truth table or plotting is to be done at each successive time units, specified by PTIME.
EXECUT = specify that complete simulation is needed.
MACGEN = give the list of statements expanded after macro call.

2. $ \& \text{ CKTNAM } \& \text{ name }

Where name is a 20 character in length name field which will be printed with system truth table, plotting of signal, fan-out listing and cross-reference listing for documentation. Any characters after 20 bytes will be truncated. This command is optional.

3. $ \& \text{ PTIME } \& \text{ t1 } R \& \text{ t2 }

Where t1 and t2 are five character numeric number and specify starting and repeat times of printing. R indicates that repeat factor is followed by R.

4. $ \& \text{ ETIME } \& \text{ t }

Where t is a five character numeric number showing the end time for simulation. This is the last command.

---CIRCUIT CONNECTION DESCRIPTION

Macro definition: The macro statements must start with a macro definition. This is defined as,

\[ \& \text{ name } \& \text{ s1 } \& \text{ s2 } \& \text{ s3 } \ldots \& \text{ d1 } \& \text{ d2 } \& \text{ d3 } \& \text{ d4 } \ldots \]
Where macro name must start with a dot (.).

'name' is a six character name of the macro.

s1, s2 ..... are signal node name each five characters in length. These operands will be replaced during macro call.

'/' is an indicator of separation between signal node name and delay operand.

d1, d2, ..... are delay operands each three numeric character in length.

A macro ends with a .¥ MEND statement.

All macro name must be different.

CIRCUIT DESCRIPTION:

All circuit description cards whether defined by a macro or by a statement format command have the following formats.

For basic gate,

\[ \text{GID} \backslash d1 \backslash d2 \backslash d3 \backslash s1 \backslash s2 \backslash s3 \] ...

Where GID (Gate IDentification) is the gate identification name and indicates one of the gates of table Appendix 1-1.

d1, d2 and d3 are setup, \( T_{\text{PHL}} \) and \( T_{\text{PLH}} \) delays.
s1 is the output node of the gate.
s2, s3 ..... are the input nodes of the gate.

For flip-flop gate,

\[ \text{GID} \backslash d1 \backslash d2 \ldots \ldots \backslash d8 \backslash s1 \backslash s2 \ldots \ldots \backslash s7 \]

Where GID is the gate identification name and indicates any one of the three flip-flops of table App. 1-1.
d1, d2, ....... d8 are the Reset and Set setup delay, clock minimum high and low pulse width time, setup and hold delays of J and K inputs, propagation delays for Q and $\overline{Q}$.

s1, s2, ....... s7 are the Reset, Set, Clock, J input, K input, Q output, $\overline{Q}$ output.

MACRO CALL;

Macro is called by the following statement.
& $\&$ name $\&$ s1 $\&$ s2 $\&$ s3 ....... $\&$ / $\&$ d1 $\&$ d2 .......

Where,

'name' is any six character name and must be defined in macro definition.

s1, s2, s3 are signal nodes that will replace the nodes in statements defined by the macro.

d1, d2, d3 ....... are delays that will replace the delays in statements defined by the macro.

The number of nodes and delays in macro call must be same as in macro definition. All the operands are positional.

INPUT TEST SIGNAL SPECIFICATION

The test signal description has the following format,
Signal $\&$ iv $\&$ t1 $\&$ t2 $\&$ t3 ....... $\&$ x $\&$ t$\overline{m}$

Where,

'Signal' is the name of the input test signal. This must be one of the node name of the circuit.

'iv' is the initial value of the signal and may be 0 or 1.
$t_1, t_2, \ldots$ are the switching time points and must be in ascending order.

$X$ may be $R$ or $D$. When $X$ is $R$, it indicates repetition. This means that the signal becomes periodic after the last switching with a period $t_n$. When $X$ is $D$, it means that the signal becomes constant after getting a $D$ and the signal does not change its state at the last time point before $D$.

PRINT SIGNAL SPECIFICATION

The format for signals to be displayed is $s_1 \gamma s_2 \gamma s_3 \ldots \gamma s_n$

Where $s_1, s_2, \ldots$ are signals to be displayed or monitored.
<table>
<thead>
<tr>
<th>Type of gate</th>
<th>Gate Identification</th>
<th>Actual code in program</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND</td>
<td>00</td>
</tr>
<tr>
<td>NAND</td>
<td>NND</td>
<td>02</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>01</td>
</tr>
<tr>
<td>INVERT</td>
<td>INV</td>
<td>04</td>
</tr>
<tr>
<td>NOR</td>
<td>NOR</td>
<td>03</td>
</tr>
<tr>
<td>EXCLUSIV-OR</td>
<td>XOR</td>
<td>05</td>
</tr>
<tr>
<td>Positive edge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triggered (Clock)flip-flop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative edge triggered (clock) flip-flop</td>
<td>JKP</td>
<td>21</td>
</tr>
</tbody>
</table>

For level triggered flip-flop use JKP/JKN for gate identification and zero setup delay. The hold delay should be equal to the high/low pulse width time of the respective clock.

Table App-1-1. Gate identification definition.
APPENDIX II

TABLES USED BY THE PROGRAM
For Basic gate

<table>
<thead>
<tr>
<th>Gate type</th>
<th>Byte count</th>
<th>$T_{su}$</th>
<th>$T_{dHL}$</th>
<th>$T_{dLH}$</th>
<th>No. of input</th>
<th>Input signal no.</th>
<th>Value to be used</th>
</tr>
</thead>
</table>

Relative byte count from +0 address

<table>
<thead>
<tr>
<th>Ref. setup time</th>
<th>Value 1</th>
<th>DRT1</th>
<th>Value 2</th>
<th>DRT2</th>
<th>Iteration flag</th>
<th>Input signal no.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Output signal no.</th>
<th>Gate type</th>
</tr>
</thead>
</table>

Gate type byte: X X X X X X X X

<table>
<thead>
<tr>
<th>Bit meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-bit — gate iteration flag</td>
</tr>
<tr>
<td>R-bit — gate decrement flag</td>
</tr>
<tr>
<td>N/FF-bit — 0 for basic gate / 1 for flip-flop gate.</td>
</tr>
</tbody>
</table>
**TABLE 1 (Contd.)**

**CA-bit** — if flip-flop gate, this is the clock access flag

**GI-bits** — this group determine the type of gate

**Iteration flag** : This flag indicates that the associated input's value to be used can take part in signal setting.

For flip-flop gate

<table>
<thead>
<tr>
<th>Gate type</th>
<th>Byte count</th>
<th>Reset setup time</th>
<th>Set setup time</th>
<th>Clock low pulse width time</th>
<th>Clock high pulse width time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Setup delay for J, K and clock</th>
<th>Hold delay for J and K</th>
<th>T_{DLH}</th>
<th>T_{DHL}</th>
<th>Reset signal no.</th>
<th>Delay ref. time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set signal no.</th>
<th>Delay ref. time</th>
<th>Clock signal no.</th>
<th>Delay ref. time</th>
<th>J signal no.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T_{su} / T_h for J</td>
<td>K signal no.</td>
<td>T_{su} / T_h for K</td>
<td>Q signal no.</td>
<td>Q̅ signal no.</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>27</td>
<td>25</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q2</th>
<th>DRT2</th>
<th>IT</th>
<th>Q̅1</th>
<th>DRT1</th>
<th>Q̅2</th>
<th>DRT2</th>
<th>IT</th>
<th>Clock setup time from clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
</tbody>
</table>

IT byte: Iteration flag for setting Q/Q

Byte count: This counts the number of bytes a gate uses.
TABLE 2 (iSIG field)

Input test signal table.

<table>
<thead>
<tr>
<th>No. of test signal</th>
<th>Test signal no.</th>
<th>Byte count</th>
<th>Status</th>
<th>N</th>
<th>SW1</th>
<th>SW2</th>
<th>.....</th>
<th>X</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test signal no. ...........
2

Status byte: X X X X X X X X X

\[iv R/ DC/ Presently not\]
\[NR AC/ Presently not\]
used

Bit meaning;

R/NR - this flag when set, indicates that the signal becomes periodic

AC/DC - if this flag is set, it indicates that the signal has a constant DC level.

iv - initial value.

N byte: this shows which switching field is to be compared with system reference time.

X byte: this may be R or D, indicating a periodic or DC signal shaping.

K byte: is the repeat factor for periodicity.
TABLE 3 (PT field)

<table>
<thead>
<tr>
<th>Ref. time</th>
<th>Repeat time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

TABLE 4 (ETIME field)

<table>
<thead>
<tr>
<th>End-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+ 2</td>
</tr>
</tbody>
</table>

TABLE 5 (IDONGT field)

<table>
<thead>
<tr>
<th>00 00 80 80 00 80 80 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+ 1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

These relations are for AND, OR, NAND and NOR gates respectively. All through the program a hex 80 indicates high and a hex 00 indicates low state.

TABLE 6 (SIGSTS field)

<table>
<thead>
<tr>
<th>Total no. of signal</th>
<th>ST1</th>
<th>ST2</th>
<th>.......</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

ST1, ST2 .... are the states of signals 1, 2, .... respectively.
**TABLE 7 (SIGDG field)**

<table>
<thead>
<tr>
<th>Byte count</th>
<th>No. of gate connected</th>
<th>Gate no.</th>
<th>Gate no.</th>
<th>\ldots \ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Byte count \ldots \ldots

0+  This is for next signal

**TABLE 8 (DISIOS field)**

<table>
<thead>
<tr>
<th>No. of signal to be displayed</th>
<th>Signal no.</th>
<th>Signal name</th>
<th>Signal no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td>1</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Signal name \ldots \ldots

10 15

**TABLE 9 (VTAB field)**

<table>
<thead>
<tr>
<th>No. of variables</th>
<th>Name</th>
<th>Name</th>
<th>\ldots \ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+</td>
<td>2</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

APPENDIX III
PROGRAM FLOWCHART
Get a card and print it with line number

Is it a comment or command card?

Error: Starting Character should be a "$" or a "*"

Check the validity

Any error?

Issue message

Determine the type of command and branch to respective data set processor

Check the validity
Replace the operands if possible in macro statements

Option command processor

Check the operands

Yes

Any error?

No

Issue message

BIU

Set the option flags as specified

BIU
Input test signal processor

Get a card and print it with line number

Is it a command? Yes
No

Any signal processed? COM

Error: Command found before any input test signal specification

Is it a comment? Yes
No

Check the statement

Any error? Yes
No

Issue message

Store signal number and waveform specification

ITP
Print signal processor

Get a card and print it with line number

Check the card

Yes

Any error?

No

Issue message

BIU

Store the signal in the print signal table

BIU

Print time processor

Check the card

Yes

Any error?

No

Issue message

BIU

Store the start & repeat interval

BIU
Yes

Issue message

Any error?

Yes

Error: Fatal error(s) detected

Yes

Generate Signal driving gate table

Generate signal state table and make all stage undefined

STOP

No

Store the end time

Yes

Any error occurs in all the previous statements

Yes

Option execution specified?

Yes

STOP

No

No

Yes

Check the card
Cleal's SRT.

N=No. of test
  signal

Set pointer 1 to first signal in input
test signal table

SUBROUTINE; SETLRF
  (Level delay setter)

Advance pointer 1 to next signal

N=N-1

N:0

PRT

Yes
  Option for printing:
  :continuous

No

Yes
  Print time
  reached?

No
  Print flag set?

Reset print flag

Print the signal
  states of all signal
  in DISIOS table
System ref. time:
end time

Increase system ref
time by one time unit

Yes System decrement
flag set?

No

UPS

Reset system
decrement flag

N=number of
gate

Set pointer 1 to
first gate in IGOT

Yes Gate decrement flag set?

No

GTA

Advance pointer 1
to next gate

N= N-1

IT N:0

Reset the flag

Check ref. setup time for each Reset,
Set, Clock, J and K input. If any one
has non-zero value, decrement it by
one t.u. and if it still remains non-
zero set gate and system decrement
flag
Increment DRTs of $Q$ and $\bar{Q}$ if any one is non-zero. If first DRT of $Q/\bar{Q}$ becomes zero at the time of decrement, set the associated gate and $Q/\bar{Q}$ iteration flag. Set also system iteration flag.

Yes

Clock access ref. time: 0

GTA

Decrement it by one t.u.

Yes

Is it zero?

No

Reset clock access flag

GTA

NMD

NMR

Set gate & system decrement flag

Set pointer 2 to first input

GTA

Yes

Ref: setup time : 0

N=No.of input

No

Decrement it by one t.u.

Yes

Is it still non-zero?

No

Set gate and system decrement flag

C
No DRT1 : 0

DRT1 = DRT1 - 1

No DRT1 : 0

DRT2 : 0

DRT2 = DRT2 - 1

Value1 = Value2

DRT1 = DRT2

DRT2 = 0

Set gate and system iteration flag

Yes No

Value to be used = Value 1

No DRT2 : 0

Set system and gate decrement flag

Yes

NMA

Advance pointer to next input

M = M - 1

M : 0

Set system and gate decrement flag

NMA

NMA
Yes System iteration flag set?

- Reset the flag
- N = Number of gate
- Set pointer 1 to first gate

Gate iteration flag set?

- Yes
- Reset the flag
- Basic
- Get type?
  - FF
  - FFI
- No
  - Advance pointer to next gate
  - N = N - 1
  - STS
  - N = 0

M = No. of input

L = 0

Set pointer 2 to first input signal

BSI

Yes Iteration flag for input set?

BSB

No BSA
Advance pointer 2 to next input

M = M - 1

M : 0

BSS = L : No. of input signal

Gate type?

AND/OR/ NAND/NOR

Value to be used: gate input characteristics value

Yes

Output value in the signal state table: gate output charts. value

Yes

Store output signal no. and its value in temporary storage
Gate type?

Select output as invert of gate characteristics value

Xor value to be used of both inputs with each other

Compare result with output signal stage table

Yes

Value same?

No

Select output as invert of the value to be used

INV

BSM
Yes

Iteration flag for Q set?

No

Yes

Reset it

Q1 : state of Q output in signal state table?

Store signal no. and state in temporary storage

DRT2 : 0

Move Q2 to Q1 and DRT2 to DRT1

DRT2 : 0

Take the same action for Q as Q

ITA
N = total no. of signal in the temporary storage

Set pointer 1 to first signal

Load signal no. and its value from temporary storage

SUBROUTINE SETLRF (Level delay setter)

Advance pointer 1 to next signal

N = N - 1

N = 0

UPS

N = no. of test signal

Set pointer 1 to the first signal

UPR
Test to switching time points and update status

Switching time point occurs?

No

Load signal number and value

SUBROUTINE; SETLRF (Level-delay-setter)

Yes

Advance pointer 1 to next signal

N = N - 1

VRT = N : 0
Set pointer 2 to first input

\( M = \text{No. of inputs} \)

**Yes**

Signal: this input

**SET**

**NINP**

Advance pointer 2 to next input

\( M = M - 1 \)

**No**

\( M : 0 \)

Advance pointer 1 to next gate driven by the signal

\( N = N - 1 \)

\( N : 0 \)

**TG**

**SET**

Ref. setup time: 0

**Yes**

Delay ref. time 1:0

**S1**

**No**

Error: Signal changed before setup delay

STOP

**Gate type?**

AND/OR/NAND/NOR

XOR/INV

\( \text{Buffer2} = \text{signal value} \)

\( \text{NORM} \)
Yes, Signal value: Characteristics input value

Delay ref. time 2 = delay to set output to characteristics output

NINP

No

Gate: XOR

Delay ref. time 2 = delay to set output to inverse of the input value

NINE

Yes

Delay ref. time 2 = delay to set output to inverse of the output value

NINP

No

Delay ref. time 2 = delay to set output to characteristics output

NINP

The same action as on buffer 2 except the values are pushed to buffer 1 and delay ref. time 1

NINP
Clock access flag set?

-ve Clock: -ve/+ve edge triggered

Yes

DRS

Delay ref. time 1:0

No

Set Q delay ref. time 1

Set Q value 1

Set Q delay ref. time 2

Set Q value 2

Msg: Clock action is suppressed for Q output because of Set/Reset signal

DRSA

DRSE

Yes

Yes

DRS

Clock signal value: 0

DRSD

No

No

Clock signal value: 1

DRSE

DRSA
Set clock access flag

Similar action is taken for Q value & delay ref. time setting

Yes

Signal: Clock

Clock: -ve/+ve triggered

No

Clock signal value

Yes

Clock: -ve/+ve triggered

No

Clock ref. setup time: 0

Clock ref. setup time = clock high pulse width time

Error: Negative edge triggered low pulse width short

JIN

CLK

Clock ref. setup time = clock high pulse width time

JIN

STOP

Error: Negative edge triggered low pulse width short

STOP

Yes

Clock ref. setup time: 0

CLK

Yes

Clock ref. setup time = clock low pulse width time

CKB

No

Clock ref. setup time = clock high pulse width time

STOP
No Clock access allowed?

Yes

Data ref. setup time for J: 0

No

Error: J ref. setup time short

No

Data ref. setup time for K: 0

Yes

Error: K ref. setup time short

Set ref. hold delay for both J & K input

STOP

From J and K input value: decide Q and Q value and respective delay

Yes

DRT1 of Q: 0

No

Set Q value 1

No

Set Q value 2

Set Q DRT 1

Take similar action for Q and DRT setting

JIN
The same action as negative edge triggered clock except ON condition is when clock goes high.

**CKP**

Yes

**JIN**

Yes

***Signal: J input***

Yes

**J ref. setup time : 0**

Set ref. Setup delay

No

Error: J changed before ref. Setup /hold delay

**KIN**

Yes

**STOP**

No

**AG**

**Yes**

***Signal: K input***

Yes

**K ref. setup time : 0**

Set ref. setup delay

No

Error: K changed before ref. setup /hold delay

**AG**

**STOP**
APPENDIX IV

GATE TIMING SPECIFICATION
<table>
<thead>
<tr>
<th>LOGIC FAMILY</th>
<th>OPERATING SUPPLY VOLTAGE</th>
<th>MINIMUM LOGIC &quot;1&quot; INPUT</th>
<th>MAXIMUM LOGIC &quot;1&quot; OUTPUT</th>
<th>MINIMUM LOGIC &quot;0&quot; INPUT</th>
<th>MAXIMUM LOGIC &quot;0&quot; OUTPUT</th>
<th>MAXIMUM LOGIC &quot;1&quot; INPUT CURRENT</th>
<th>MAXIMUM LOGIC &quot;0&quot; OUTPUT CURRENT</th>
<th>MAXIMUM PROPAGATION DELAY (ns)</th>
<th>MAXIMUM COUNTER FREQUENCY</th>
<th>MAXIMUM FANOUT</th>
<th>LOADING FACTOR</th>
<th>TYPICAL POWER DISSIPATION PER GATE</th>
<th>TYPICAL f&lt;sub&gt;T&lt;/sub&gt; 10-90%</th>
<th>SPECIFIC DEVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7400</td>
<td>4.75 5.25</td>
<td>2.0 / 0.8</td>
<td>2.4 / 0.4</td>
<td>1.6 mA / 40 mA</td>
<td>16 mA / 400 mA</td>
<td>8 ns</td>
<td>13 ns</td>
<td>35 MHz</td>
<td>10</td>
<td>1</td>
<td>10 mW</td>
<td>15 ns</td>
<td>7400</td>
<td></td>
</tr>
<tr>
<td>74H</td>
<td>4.5 5.5</td>
<td>2.0 / 0.8</td>
<td>2.4 / 0.4</td>
<td>2.0 mA / 50 mA</td>
<td>20 mA / 500 mA</td>
<td>6.2 ns</td>
<td>5.9 ns</td>
<td>50 MHz</td>
<td>12-74 10-74H</td>
<td>1.15</td>
<td>22 mW</td>
<td>7 ns</td>
<td>74H00</td>
<td></td>
</tr>
<tr>
<td>74L</td>
<td>4.75 5.25</td>
<td>2.0 / 0.8</td>
<td>3.5 / 0.4</td>
<td>0.18 mA / 10 mA</td>
<td>20 mA / 1 mA</td>
<td>5 ns</td>
<td>5 ns</td>
<td>3 MHz</td>
<td>12-74 10-74L</td>
<td>0.26</td>
<td>1 mW</td>
<td>30 ns</td>
<td>74L00</td>
<td></td>
</tr>
<tr>
<td>74S</td>
<td>4.75 5.25</td>
<td>2.0 / 0.8</td>
<td>2.7 / 0.4</td>
<td>0.36 mA / 3 mA</td>
<td>5 mA / 20 mA</td>
<td>8 ns</td>
<td>8 ns</td>
<td>50 MHz</td>
<td>12-74 10-74S</td>
<td>1.25</td>
<td>18 mW</td>
<td>3 ns</td>
<td>74S500</td>
<td></td>
</tr>
<tr>
<td>74LS</td>
<td>4.75 5.25</td>
<td>2.0 / 0.8</td>
<td>2.7 / 0.4</td>
<td>0.36 mA / 3 mA</td>
<td>5 mA / 20 mA</td>
<td>8 ns</td>
<td>8 ns</td>
<td>50 MHz</td>
<td>12-74 10-74LS</td>
<td>0.5</td>
<td>2 mW</td>
<td>15 ns</td>
<td>74LS00</td>
<td></td>
</tr>
<tr>
<td>PEC II</td>
<td>-4.7 -5.7</td>
<td>-0.950 -1.600</td>
<td>-1.650 -0.950</td>
<td>-0.5 mA / 165 mA</td>
<td>50 mA / 500 mA</td>
<td>2 ns</td>
<td>2 ns</td>
<td>50 MHz</td>
<td>12-74 10-74II</td>
<td>0.35</td>
<td>16 mW</td>
<td>5 ns</td>
<td>MC1642</td>
<td></td>
</tr>
<tr>
<td>PEC III</td>
<td>-4.7 -5.7</td>
<td>-0.950 -1.600</td>
<td>-1.650 -0.950</td>
<td>-0.5 mA / 165 mA</td>
<td>50 mA / 500 mA</td>
<td>2 ns</td>
<td>2 ns</td>
<td>50 MHz</td>
<td>12-74 10-74III</td>
<td>0.5</td>
<td>2 mW</td>
<td>15 ns</td>
<td>MC10101</td>
<td></td>
</tr>
<tr>
<td>74L</td>
<td>4.75 5.25</td>
<td>2.0 / 0.8</td>
<td>3.5 / 0.4</td>
<td>0.36 mA / 3 mA</td>
<td>5 mA / 20 mA</td>
<td>8 ns</td>
<td>8 ns</td>
<td>50 MHz</td>
<td>12-74 10-74L</td>
<td>0.25</td>
<td>1 mW</td>
<td>30 ns</td>
<td>74L000</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE A-5-1**  
Comparison of Common Logic Family Characteristics

- CMOS 3.3 Vee 1/3 Vee
- Typical values may vary depending on specific applications and conditions.
SERIES 54H/74H
HIGH-SPEED TRANSISTOR-TRANSISTOR LOGIC

TYPICAL CHARACTERISTICS

PROPAGATION DELAY TIME,
LOW-TO-HIGH-LEVEL OUTPUT

FREE-AIR TEMPERATURE

Fig. A-5-1: Interdependencies between propagation delays and free-air temperature.
**Fig. A-5-2**: Interdependencies between propagation delays and load.

**Fig. A-5-3**: Timing diagram for 74175 of National Semiconductor, shows typical interrelationships among different timing parameters.
Table A-5-2 (Contd.)

recommended operating conditions

<table>
<thead>
<tr>
<th>SERIES 54H/74H</th>
<th>'H71</th>
<th>'H72, 'H73, 'H76</th>
<th>'H74</th>
<th>'H78</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MIN</td>
<td>NOM</td>
<td>MAX</td>
<td>MIN</td>
<td>NOM</td>
</tr>
<tr>
<td>Supply voltage, VCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Series 54H</td>
<td>4.5</td>
<td>5</td>
<td>5.5</td>
<td>4.5</td>
<td>5</td>
</tr>
<tr>
<td>Series 74H</td>
<td>4.75</td>
<td>5</td>
<td>5.25</td>
<td>4.75</td>
<td>5</td>
</tr>
<tr>
<td>High-level output current, I0H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Series 54H</td>
<td>400</td>
<td>-</td>
<td>1000</td>
<td>400</td>
<td>-</td>
</tr>
<tr>
<td>Series 74H</td>
<td>400</td>
<td>-</td>
<td>1000</td>
<td>400</td>
<td>-</td>
</tr>
<tr>
<td>Low-level output current, I0L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock high</td>
<td>12</td>
<td>12</td>
<td>15</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Clock low</td>
<td>28</td>
<td>28</td>
<td>35</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Clear or preset low</td>
<td>16</td>
<td>16</td>
<td>25</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Pulse width, tW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setup time, tSU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-level data</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Low-level data</td>
<td>0</td>
<td>0</td>
<td>51</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hold time, tH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating free-air temperature, T_A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Series 54H</td>
<td>-55</td>
<td>125</td>
<td>-55</td>
<td>125</td>
<td>-55</td>
</tr>
<tr>
<td>Series 74H</td>
<td>0</td>
<td>70</td>
<td>0</td>
<td>70</td>
<td>0</td>
</tr>
</tbody>
</table>

The arrow indicates the edge of the clock pulse used for reference: † for the rising edge, ‡ for the falling edge.

switching characteristics, VCC = 5 V, T_A = 25°C

<table>
<thead>
<tr>
<th>PARAMETER‡</th>
<th>FROM (INPUT)</th>
<th>TO (OUTPUT)</th>
<th>TEST CONDITIONS</th>
<th>'H71, 'H72, 'H73, 'H76</th>
<th>'H74</th>
<th>'H78</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MIN</td>
<td>NOM</td>
<td>MAX</td>
<td>MIN</td>
</tr>
<tr>
<td>t_{max}</td>
<td>Preset</td>
<td>Q</td>
<td>C_L = 25 pF, R_L = 280 Ω, See Note 2</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>43</td>
</tr>
<tr>
<td>t_{PLH}</td>
<td>Preset</td>
<td>Q</td>
<td></td>
<td>6</td>
<td>13</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>t_{PLH}</td>
<td>Clear</td>
<td>Q</td>
<td></td>
<td>6</td>
<td>13</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>t_{PHL}</td>
<td>Preset</td>
<td>Q</td>
<td></td>
<td>12</td>
<td>24</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>t_{PHL}</td>
<td>Clock</td>
<td>Q or Q</td>
<td></td>
<td>14</td>
<td>21</td>
<td>8.5</td>
<td>15</td>
</tr>
<tr>
<td>t_{PHL}</td>
<td>Clock</td>
<td>Q or Q</td>
<td></td>
<td>22</td>
<td>27</td>
<td>13</td>
<td>20</td>
</tr>
</tbody>
</table>

‡ t_{max} = maximum clock frequency; t_{PLH} = propagation delay time, low-to-high-level output; t_{PHL} = propagation delay time, high-to-low-level output.
REFERENCES


