NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS RÉCUÉ
A MULTIPLE DELAY SWITCH-LEVEL SIMULATOR FOR MOS/LSI CIRCUITS

by

Kia Liang Chuah

A thesis submitted to the School of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Applied Science in the Department of Electrical Engineering University of Ottawa

Ottawa, Ontario, 1985

© Kia Liang Chuah, Ottawa, Canada, 1985.
The University of Ottawa requires the signatures of all persons using or photocopying this document. Please sign below, and give address and date.
ACKNOWLEDGEMENT

I wish to express my sincere appreciation to my supervisor, Dr. K. W. Chiang, for his advice and encouragement throughout the preparation of this thesis, and also for his financial support.
ABSTRACT

In this thesis a new multiple delay switch-level simulator for MOS LSI circuits is presented with the following features: table-driven, time queue with macro event list for simulation with multiple rise and fall delays, selective tracing of active paths, and event scheduling and rescheduling on time queue including spike detection. The delay model used is organized around subnetworks, instead of logic gates. The use of subnetworks permits both logic gates and pass transistors to be handled in a uniform fashion. The delay model takes into account the effects of not only the output loading capacitances but also the input waveforms. An improved algorithm is described and implemented for computing the states and delays of a subnetwork.
# TABLE OF CONTENTS

ACKNOWLEDGEMENT ........................................ iv

ABSTRACT ........................................... v

Chapter I: INTRODUCTION ................................ 1

Chapter II: MOS LSI TECHNOLOGY ......................... 6
   Introduction ........................................ 6
   MOS Transistor .................................... 7
   Analysis of the MOS Transistor ................. 10
      Current-Voltage Characteristics .......... 11
      Capacitances of the MOS Transistor .... 15
   MOS Circuits ..................................... 17
   Transient Response of an NMOS Inverter .... 21
      Definition of Transient Characteristics .. 22
      Equivalent Capacitance Calculations ..... 25
      Rise Time and Rise Delay ................. 27
      Fall Time and Fall Delay ................. 28
   Transient Response of NMOS Circuits ....... 31
   Delay Calculations With Linear Ramp Input ... 44
      Delays of an Inverter ....................... 45
      Delays of a Pass Transistor Chain ....... 51

Chapter III: LOGIC SIMULATION ......................... 57
   Introduction ...................................... 57
   Input Circuit Description ..................... 59
   Delay Modeling .................................. 62
   Multiple Logic Values ......................... 64
   Simulator Structures ......................... 65
      Event Directed Simulation ............... 65
      Event Scheduling ......................... 66

Chapter IV: MULTIPLE DELAY SWITCH-LEVEL SIMULATION ALGORITHM ......................... 68
   Introduction .................................... 68
   Network Model .................................. 69
      Network Representation .................... 72
      Partitioning of the Network ............. 73
# FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>NMOS Transistor</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>NMOS device cross-section showing depletion region and induced channel</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>I-V characteristics for a MOS transistor</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>MOS transistor capacitances</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>The Miller capacitance</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>NMOS depletion-load inverter</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>3-input NOR gate</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>3-input NAND gate</td>
<td>20</td>
</tr>
<tr>
<td>2.9</td>
<td>Steering logic circuit</td>
<td>21</td>
</tr>
<tr>
<td>2.10</td>
<td>An equivalent circuit of Figure 2.9</td>
<td>21</td>
</tr>
<tr>
<td>2.11</td>
<td>Transient model for an NMOS inverter</td>
<td>22</td>
</tr>
<tr>
<td>2.12</td>
<td>Definitions of transition and delay times</td>
<td>24</td>
</tr>
<tr>
<td>2.13</td>
<td>Lumping of capacitances at inverter output nodes</td>
<td>26</td>
</tr>
<tr>
<td>2.14</td>
<td>A pass transistor</td>
<td>39</td>
</tr>
<tr>
<td>2.15</td>
<td>Pass transistor chain</td>
<td>39</td>
</tr>
<tr>
<td>2.16</td>
<td>An inverter driven by different input waveforms</td>
<td>44</td>
</tr>
<tr>
<td>2.17</td>
<td>NMOS inverter chain</td>
<td>46</td>
</tr>
<tr>
<td>2.18</td>
<td>Delay times $t_{PLH}$ and $t_{PHL}$ for various capacitive load conditions</td>
<td>50</td>
</tr>
<tr>
<td>2.19</td>
<td>A pass transistor chain driven by an inverter</td>
<td>52</td>
</tr>
<tr>
<td>2.20</td>
<td>A pass transistor tree driven by an inverter</td>
<td>54</td>
</tr>
<tr>
<td>3.1</td>
<td>Circuit generated from Boolean equation</td>
<td>59</td>
</tr>
<tr>
<td>3.2</td>
<td>Macro representation of a 32-bit register</td>
<td>60</td>
</tr>
<tr>
<td>3.3</td>
<td>Element representation of a JK flip-flop</td>
<td>60</td>
</tr>
<tr>
<td>3.4</td>
<td>Input and output timing diagrams of an</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.5</td>
<td>List structure scheduler</td>
<td>67</td>
</tr>
<tr>
<td>4.1</td>
<td>Illustrations of the node states and node strengths of a circuit</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>A MOS transistor</td>
<td>71</td>
</tr>
<tr>
<td>4.3</td>
<td>Network partitioning</td>
<td>74</td>
</tr>
<tr>
<td>4.4</td>
<td>Searching of direct paths</td>
<td>77</td>
</tr>
<tr>
<td>4.5</td>
<td>Simulation time flow mechanism (Time Queue)</td>
<td>80</td>
</tr>
<tr>
<td>4.6</td>
<td>Input and output timing diagrams of an error condition</td>
<td>81</td>
</tr>
<tr>
<td>4.7</td>
<td>Event-Driven Simulation Algorithm</td>
<td>82</td>
</tr>
<tr>
<td>4.8</td>
<td>Side capacitances are assumed to be connected to the branch nodes</td>
<td>99</td>
</tr>
<tr>
<td>5.1</td>
<td>Overall table configuration and interaction</td>
<td>103</td>
</tr>
<tr>
<td>5.2</td>
<td>Node table</td>
<td>104</td>
</tr>
<tr>
<td>5.3</td>
<td>Description of node table</td>
<td>105</td>
</tr>
<tr>
<td>5.4</td>
<td>Transistor table</td>
<td>106</td>
</tr>
<tr>
<td>5.5</td>
<td>Description of transistor table</td>
<td>107</td>
</tr>
<tr>
<td>5.6</td>
<td>Fanoutset table</td>
<td>108</td>
</tr>
<tr>
<td>5.7</td>
<td>Interconset table</td>
<td>108</td>
</tr>
<tr>
<td>5.8</td>
<td>Timing table</td>
<td>109</td>
</tr>
<tr>
<td>5.9</td>
<td>Simulation time flow mechanism</td>
<td>110</td>
</tr>
<tr>
<td>5.10</td>
<td>Input and output waveforms for Example 5</td>
<td>113</td>
</tr>
<tr>
<td>5.11</td>
<td>Input and output waveforms for Example 6</td>
<td>115</td>
</tr>
</tbody>
</table>
Chapter I

INTRODUCTION

The rapid progress of LSI technologies enables a designer to produce denser and more complex integrated circuits, especially in the MOS area. Design verification in each design step of logic systems has particularly become an important technology in the LSI design process. Hardware prototyping and software simulation are effective means for design verification. The latter has been extensively used because of the following reasons:

1. A logic simulator can be used to check if the present design can perform the intended function before committing it to hardware.

2. A timing simulator can be used to predict the timing relations among signals closely by taking the actual device parameters and loading conditions into account.

3. A circuit simulator can be used to evaluate the electrical characteristics with variations in device parameters and operating conditions.

A variety of software simulators for MOS LSI circuits is currently available; they include circuit simulators [15] [20], timing simulators [10] [11] [19], logic simulators [8] [9] [18] [21] and combinations of these [1] [12], each providing a certain degree of accuracy.
The primary concern of circuit simulators is accuracy: the ability to simulate exactly the real-world behavior of devices. A circuit is modeled by a collection of differential equations, and the equations are solved to predict the circuit's behavior. This approach produces highly accurate results, and has resulted in widespread usage of circuit simulators as a design tool. Unfortunately, the accuracy of the circuit models comes at a high price in execution time: circuit simulators typically require several seconds of CPU time per transistor. As a result, it is impractical for today's state-of-the-art VLSI circuits, which contain tens or hundreds of thousands of transistors.

In timing simulator, circuits are described at the transistor level. The concept of gate here deviates from that used in logic simulation to represent specific structures of transistors. These structures are either a configuration of driver and load devices or pass transistors, and are in general functionally more complex than logic gates. Timing simulator evaluates its gates by finding the current in each device and then integrating the net charging current flowing into the gate output capacitance. The gate voltages obtained in this way are propagated to other gates by techniques similar to those used in logic simulators. By using tables to store device I-V characteristics, and through simplified model of the transistors and their associated capacitances, timing simulators are faster than circuit simula-
tors. Because of the amount of detail the timing simulators have to examine, the number of components in a circuit that these simulators can handle within reasonable amount of time and space is limited to less than a few thousands gates.

In a logic simulator, circuits are described at the gate level, whose inputs and outputs are binary valued and the signal propagation delay through the device can be specified. Logic simulators exist with capability of dealing with circuits up to tens of thousands of gates. However, the delay models used are generally insufficient for the analysis of MOS LSI circuits, because the circuit delay times are affected not only by the output loading but also by the input loading and input waveforms. The mixed-mode simulators, which aim at accuracy as in circuit simulation and efficiency as in logic simulation, still have the drawback that accurate timing may be lost in interfacing between timing and logic levels.

The multiple delay simulator for MOS LSI circuits proposed by Nham and Bose [16] makes significant use of device level information. Their approach has a number of limitations as pointed out by the authors, including the inability to handle "multiple input to output delays", "delay of a gate driving the data input of a pass transistor", "slope of the input waveform", and "overlapping transition".

Perhaps, a better tool for MOS LSI logic verification is a switch level simulator such as MOSSIM [5], and others [3]
However, currently such simulators use a unit delay model which can only verify logic without providing any timing information. Switch level simulators model the network in a way which corresponds closely to the actual circuit. Transistors are viewed as switches with no assigned direction of information flow. Nodes can either provide strong signals to the network or can store charge dynamically. Switch level simulators can model a large class of designs, because the basic logic elements of the simulator correspond directly to the basic components of the circuit. The circuit for a switch level simulation can be extracted directly from the layout masks; it requires no knowledge of logic design or the intended functions of the system. The combination of this automatic network extraction followed by switch level simulation may uncover many errors in both the logic design and the layout design. This check may prove to be invaluable.

The multiple delay switch level simulator presented in this thesis is developed to bridge the gap between logic and timing simulators and to provide a compromise between their respective advantages of speed and accuracy. The simulator makes significant use of the device level information; transistors are evaluated logically, and variable rise and fall delays are associated with the transitions. The algorithm used is based on Bryant's algorithm [5], but it is table-driven event-directed and therefore allows for multiple
delays. This simulator also helps in removing most of the limitations mentioned above for that proposed by Nham and Bose [16].

The organization of the thesis is as follows:

Chapter 2 introduces MOS LSI technology. Basic theory and characteristics of the MOS devices are discussed. Delays for the MOS circuits are derived.

Chapter 3 describes some of the techniques used in a conventional logic simulation.

Chapter 4 describes a new multiple delay simulator for MOS LSI circuits. Algorithms for the simulator are presented.

Chapter 5 describes the implementation issues of the simulator.
Chapter II

MOS LSI TECHNOLOGY

2.1 Introduction

Metal oxide semiconductor (MOS) very-large-scale integration (VLSI) has emerged as the most significant development in the electronic industry. MOS technology is the basis for most of the LSI/VLSI digital memory and microprocessor circuits.

The most important advantage of MOS circuits over bipolar circuits is that more transistors and more functions can be successfully fabricated on a single chip. The reasons are threefold: First, an individual MOS transistor occupies less chip area. Second, the fabrication process for MOS circuits involves fewer steps, and thus, results in fewer defects per unit area. This makes it feasible to produce larger chips. Third, dynamic circuit techniques that require fewer transistors to realize a given circuit function are practical and commonly used in MOS technology. The result of these differences is that, for a given function, MOS LSI circuits are significantly cheaper to produce than bipolar circuits. This, accompanied by the fact of increased familiarization of the design techniques for MOS circuits since the introduction of structured VLSI design methodology, accounts for
the trend of steadily increasing fraction of the total digital IC market being taken up by MOS LSI/VLSI circuits.

Even though bipolar circuits are traditionally thought of as very high speed applications, the speed advantage is gradually disappearing as minimum feature size of advanced MOS technology is being scaled down into submicron range [14].

2.2 MOS Transistor

A MOS field-effect transistor (FET) features extremely high input impedance and voltage-controlled output current. A diagrammatic cross-section of an n-channel MOS (NMOS) transistor is shown in Figure 2.1. It consists of two regions of heavily doped n-type (n⁺) silicon, introduced in a p-type substrate by diffusion or ion-implantation. By convention, during circuit operation the n⁺ region with more positive voltage is referred to as the drain while the other as the source. Because of the symmetry, the source and drain are interchangeable. The p-type area between the source and the drain is the channel region. On top of the channel region is a thin layer of dielectric material, typically silicon dioxide (SiO₂), and a layer of conductor, traditionally metal but largely replaced by polycrystalline silicon (polysilicon) more recently. This forms the gate terminal of the transistor. Note that the term MOS (metal-oxide-semiconductor) is derived from the device structure.
Figure 2.1: NMOS Transistor.

The thin layer of dielectric material is used to insulate the gate from the rest of the MOS device and has a capacitive effect. If a positive voltage is applied to the gate terminal, an electric field is set up which will draw electrons from the substrate into the channel region. If the applied gate voltage is large enough, the p-type material in the channel region will be changed into n-type, thus creating an n-type channel. This n-type channel becomes a conduction path between the source and drain regions through which
the current flows. The direction of the current flow depends on the voltages on the source and the drain. Since the current can flow in either direction through the channel, a MOS transistor is a bilateral device.

A p-channel MOS (PMOS) transistor can be obtained by using an n-type substrate and diffused or ion-implanted p-type source and drain. The p-channel MOS transistor has inferior operating performance, because the charge carriers, holes, move much slower than those (electrons) in n-channel transistors: the mobility of holes is roughly 1/3 of that of electrons. With improvement in IC processing such as ion-implantation, self-alignment due to the use of polysilicon gates, and the use of depletion-mode load devices, NMOS has been widely used.

CMOS (complementary-symmetric MOS) technology incorporates both p-channel and n-channel transistors in the formation of functional circuits. It features extremely low static power consumption and good noise immunity. However, due to the necessity of device isolation, functional circuits usually occupy larger silicon area than their NMOS counterparts. Further, the fabrication process is more complicated. Nonetheless, CMOS is conceived as the technology for very dense ICs where power density and heat dissipation are premium considerations. To overcome the silicon area penalty, recent trend in CMOS technology is to use n-channel devices to form the functional circuits while p-channel devices are
used sparingly, mainly as the load devices. In this way, the "pseudo-CMOS" technology is essentially a variation of NMOS.

The thesis will concentrate on simulation aspects of NMOS circuits, and with slight modification, the results can be used with pseudo-CMOS circuits as well.

2.3 Analysis of the MOS Transistor

The concepts which are necessary for an understanding of the operation of MOS transistors will be discussed in this section. A more detailed theory and derivation of equations can be found in many digital integrated electronic textbooks.

In MOS devices, the conducting channel does not form for very small positive gate voltages because electrostatic potential at the surface of the p-type channel region must be made positive by application of 0.5V to 2V of gate-source voltage. The gate voltage needed to initiate formation of a conducting channel is termed the threshold voltage $V_T$ [13]. In many practical MOS devices, the channel doping concentration is modified by ion-implantation to alter the threshold voltage. A p-type implant will make the threshold more positive. On the other hand, an n-type implant will make the threshold voltage more negative. A large n-type implant can make the n-type channel exist even with a zero gate-source voltage. The threshold voltage in this case is negative.
NMOS transistors that have no conducting channel at zero gate-source voltage are termed enhancement-mode devices. A device which has a conducting channel at zero gate-source voltage is termed a depletion-mode device. For an NMOS depletion device, a negative gate-source voltage is required to deplete the conducting channel. Threshold voltages of enhancement and depletion devices are denoted as \( V_{TR} \) and \( V_{TD} \), respectively.

For many NMOS processes, the enhancement threshold voltage, \( V_{TE} \), is usually set at roughly \( 0.2V_{DD} \) to leave sufficient noise margin, and the depletion threshold voltage, \( V_{TD} \), is set at \( -0.8V_{DD} \) for optimizing electrical characteristics of functional circuits. We shall use these figures for illustration purposes in this thesis. However, it must be noted that they are process dependent parameters.

2.3.1 Current-Voltage Characteristics

Within an NMOS device, the charge carriers (electrons) move away from the source and towards the drain region through the channel (see Figure 2.2). The conventional direction of current flow within the device is from drain to source. Figure 2.3 shows typical I-V characteristics of n-channel MOS devices. In Figure 2.3(a) the MOS device characteristics refer to an enhancement device. In such a device there is no channel between source and drain at \( V_{GS} = 0V \). No drain-to-source current \( I_{DS} \) flows until the gate-to-source voltage exceeds the threshold voltage \( V_T \). In Figure 2.3(b)
the MOS device characteristics refer to a depletion device. Here a channel exists when \( V_{GS} = 0 \) V and the threshold voltage \( V_T \) is negative.

![Diagram of NMOS device cross-section showing depletion region and induced channel.](image)

**Figure 2.2:** NMOS device cross-section showing depletion region and induced channel.

Either transistor type may operate in the linear region or in the saturation region. In the linear region there is a continuous channel between source and drain, and \( I_{DS} \) varies almost linearly with \( V_{DS} \) for a fixed \( V_{GS} \). In the saturation region, the channel is pinched off at the drain when \( V_{GS} - V_T \leq V_{DS} \), the current \( I_{DS} \) remains nearly constant.

In the linear region, the drain current is given by the equation

\[
I_{DS} = \beta [2( V_{GS} - V_T ) V_{DS} - V_{DS}^2 ] \quad \text{for} \quad 0 < V_{DS} < V_{GS} - V_T
\]
(a) Enhancement device

(b) Depletion device

(c) Defining voltages and current

Figure 2.3: I-V characteristics for a MOS transistor.
In the saturation region, the drain current is given by the equation

$$I_{DS} = \beta \left( V_{DS} - V_T \right)^2$$

for $0 < V_{DS} - V_T < V_{DS}$

The constant $\beta$ is often referred to as the transistor gain factor, which is defined as

$$\beta = \frac{\mu_n C_{ox} W}{L}$$

where
- $\mu_n$ - mobility of electrons in the channel
- $C_{ox}$ - gate oxide capacitance per unit area
- $W$ - channel width
- $L$ - channel length

In a p-channel device, where the carriers are positive (holes), $V_{SD}$ and $I_{SD}$ are positive. Furthermore, a channel forms to allow current $I_{SD}$ to flow, when the source-to-gate voltage $V_{SG}$ exceeds the threshold voltage $|V_T|$. In the linear region, the drain current is given by the equation

$$I_{SD} = \beta \left[ 2(V_{SG} - |V_T|) V_{SD} - V_{SD}^2 \right]$$

for $0 < V_{SD} < V_{SG} - |V_T|$

In the saturation region, the drain current is given by the equation

$$I_{SD} = \beta \left( V_{SG} - |V_T| \right)^2$$

for $0 < V_{SG} - |V_T| < V_{SD}$
The above equations are approximations and do not include all higher-order effects. However, they are entirely adequate for a first-order analysis.

2.3.2 Capacitances of the MOS Transistor

Since the MOS transistor is a field-effect device, the switching speed of MOS circuits is limited by the time required to charge and discharge the capacitances at the driving and driven devices' electrodes and the interconnecting lines. Within LSI circuits all of these capacitances are so small (0.01 to 1 pF) that they are difficult to measure. For circuit analysis, these capacitances are calculated from process dependent data and the size of devices.

Figure 2.4 shows the capacitances among the electrodes of a MOS transistor. The capacitances $C_{sb}$ and $C_{ab}$ are the $n^+p$ junction capacitances. Theoretical equations for these capacitances can be derived from the basic physical model of the device. Hogdes [13] gives this derivation and the resultant equation is

$$C_{J_0} = \sqrt{\frac{2}{2\pi}} \frac{E_sN}{E_o}$$

and the built-in junction potential $\phi$ is given by

$$\phi = \left(\frac{kT}{q}\right) \ln\left(\frac{N_A N_D}{n_i^2}\right)$$

where $k$ - Boltzmann's constant
$T$ - absolute temperature
$q$ - electronic charge
\[ N_A, N_D - \text{doping concentration} \]
\[ n_i - \text{intrinsic carrier concentration of the silicon} \]
\[ \varepsilon_{si} - \text{permittivity of the silicon} \]

The capacitance from the gate to other electrodes, to a first-order approximation, can be calculated as the sum \( C_j \) of \( C_{gs}, C_{gb}, \) and \( C_{gd} \), which is nearly a constant and equal to

\[ C_j = W \cdot L \cdot C_{ox} = W \cdot L \cdot \frac{\varepsilon_{ox}}{t_{ox}} \]

where
- \( W \) - channel width
- \( L \) - channel length
- \( \varepsilon_{ox} \) - permittivity of the gate dielectric
- \( t_{ox} \) - thickness of the gate dielectric

![MOS transistor capacitances diagram](image)

**Figure 2.4:** MOS transistor capacitances.

In an integrated circuit, capacitances of circuit nodes are due not only to the capacitance of gates connected to the nodes but also to the distributed capacitance of the
signal paths and other stray capacitances. These capacitances, called parasitic capacitances, are not negligible. While gate capacitances are typically an order of magnitude greater per unit area than capacitances of the signal paths, the signal paths are often much larger in area than the associated gate regions. Therefore, a substantial fraction of the delay encountered may be accounted for by stray capacitance rather than by the inherent properties of the active transistors.

There is another type of parasitic capacitance, however, which is not easily accounted for. The Miller capacitance is a feedback capacitance between the drain edge of the gate and the drain node, see Figure 2.5. In Section 2.5-2, we show how these capacitances are used in delay calculations.

![Figure 2.5: The Miller capacitance.](image)

2.4 MOS Circuits

In MOS technology, digital circuits are commonly constructed entirely from transistors. The simplest digital circuit is an inverter. A MOS inverter exhibits all the essential features of a MOS logic circuit.
Figure 2.6 shows a basic NMOS depletion-load inverter circuit; together with drain current-voltage characteristics and load line. The single voltage supply $V_{dd}$ is typically 5 volts. The input, or driver, transistor is an enhancement-mode device with a positive threshold voltage (typically $0.2V_{dd}$), and the load transistor is a depletion-mode device with a negative threshold voltage (typically $-0.8V_{dd}$).

(a) Depletion-load inverter. (b) Drain characteristic and load line.

Figure 2.6: NMOS depletion-load inverter.

Positive logic is used with NMOS digital circuits. Logic values 0 and 1 are at voltage levels near the ground potential $0V$ and $V_{dd}$, respectively. The logic operation of the inverter circuit is as follows. When the input voltage $V_I$ is at the logic value 1, the driver transistor is conducting. The relatively low on-resistance of the driver transistor will pull the output voltage $V_O$ down to nearly $V_d$ the ground
potential and thus has a logic value 0. When $V_g$ is at the logic value 0, the driver transistor is non-conducting, and the off-resistance is very high. Thus the output voltage $V_o$ is pulled up through the load transistor to nearly $V_{dd}$ and has a logic value 1.

NMOS NOR and NAND gates are easily constructed from the simple inverter configuration. NOR gates are formed simply by paralleling additional driver transistors, as illustrated in Figure 2.7. The parallel connection of driver transistors in the NOR gates allows the output to rise to $V_{dd}$ only when all driver transistors are turned off. Otherwise, the output is pulled down to nearly the ground potential. NAND gates are formed simply by connecting additional driver transistors in series, as shown in Figure 2.8. The series connection of driver transistors in the NAND gates allows the output to be pulled down to nearly the ground potential only when all driver transistors are turned on. Otherwise, the output is pulled up to nearly $V_{dd}$.

![Diagram](image)

Figure 2.7: 3-input NOR gate.
Figure 2.8: 3-input NAND gate.

The transmission gates, or pass transistors, can be used in a way similar to switches to realize logic functions. As shown in Figure 2.9, signals are stored in the input capacitors of a driver transistor in some logic gates in the form of charge. Gates can be subsequently activated by providing suitable control signal so that the stored signals may be used. The transmission gate of Figure 2.9 has the advantage that each additional input requires only a minimum sized transistor. However, the voltage at the input of the inverter will be one threshold voltage lower than $V_{dd}$ when the input voltage at $X$ is high at $V_{dd}$ due to the saturation of the pass transistor. We can transfer pass transistor networks used for steering purpose into the driver part of complex gates. A functional equivalent for the circuit of Figure 2.9 is shown in Figure 2.10.
Figure 2.9: Steering logic circuit.

Figure 2.10: An equivalent circuit of Figure 2.9.

2.5 Transient Response of an NMOS Inverter

For an understanding of the transient behaviour of the NMOS inverter, the circuit shown in Figure 2.11 will be analyzed.

To simplify the analysis, the following assumptions are made:

1. All capacitances at the output node are lumped into a single load capacitor, $C_L$. This allows an analytical
solution to be developed that is adequate for most static circuits. In some dynamic circuit configurations, a more complex equivalent circuit is required due to the consideration of the voltage dependence of various capacitances.

2. The input voltage of the inverter is a step function. Theoretical expressions will be derived for the rise time and the fall time of the inverter.

![Figure 2.11: Transient model for an NMOS inverter.](image)

2.5.1 Definition of Transient Characteristics

Before analyzing the transient response of an NMOS inverter, it is necessary to define the transition and propagation delay times [13].

The waveforms at the input and output nodes are shown in Figure 2.12(a), where $V_I$ is the input node voltage, $V_O$ is the output node voltage, $V_{OH}$, $V_L$ and $V_{PP}$ are the output high, output low levels and the magnitude of the voltage swing, respectively. Rise and fall times, $t_r$ and $t_f$, are defined as the time duration in which the output node voltage swings between 10% and 90% points.
There are two propagation delays associated with transitions: the rise delay $t_{PLH}$ for the output to rise from low voltage level to high voltage level, and the fall delay $t_{PHL}$ for the output to drop from the high voltage level to the low voltage level. Both delays are defined as the time between the 50% points of the input and output waveforms.

The following expressions for the various delay times are derived based on the assumption of ideal step input waveform as illustrated in Figure 2.12(b). This ideal step input waveform is positioned with its edge at the 50% point of the actual input waveform. Therefore, the propagation delay times are measured from the edge of ideal step input waveform to the 50% point of the output waveform.
(a) Actual transient voltages.

(b) Idealized transient voltages.

Figure 2.12: Definitions of transition and delay times.
2.5.2 Equivalent Capacitance Calculations

As described in Section 2.3-2, each MOS transistor has five separate voltage-dependent capacitances connecting its four electrodes. An analytical solution that takes the voltage dependence into consideration is very difficult, if not impossible, without the use of computer aids. However, an approximate solution can be obtained if all capacitances are lumped into a single capacitance $C_T$ at the output node.

Three of the capacitances, $C_{gd}$, $C_{gd}$, and $C_{gs}$, can be approximated as a sum $C_g$, as described in Section 2.3-2. Voltage dependent junction capacitances, $C_{ds}$ and $C_{ss}$, can be replaced by an equivalent linear capacitance $C_{eq}$, as follows:

$$C_{eq} = K_{eq} \cdot C_{jo}$$

where $C_{jo}$ is the junction capacitance per unit area as given in Section 2.3-2. The dimensionless parameter $K_{eq}$ is derived in Hogdes [13]. A typical value of $K_{eq}$ is 0.55 for a supply voltage $V_{dd}$ of 5V and a body voltage $V_{ss}$ of 0V.

Figure 2.13 shows how the device capacitances in a circuit comprising two cascaded inverters can be lumped at the inverter output nodes. First, all capacitances across which there is no voltage change are ignored, since they have no effect on circuit performance. The device capacitances which must be considered are shown in Figure 2.13(a). In this figure, $C_L$ is the capacitance associated with interconnecting wires at the inverter outputs.
The capacitances $C_{T1}$ and $C_{T2}$ are calculated as follows:

$$C_{T1} = K_{eg} \left( C_{d61} + C_{s6A} \right) + C_{L1} + C_{g3}$$

$$C_{T2} = K_{eg} \left( C_{d63} + C_{s6A} \right) + C_{L2}$$

The result of the linearization of junction capacitances using $K_{eg}$ is a minor distortion of the shape of transient voltage waveform. However, the propagation delays are not affected significantly by this approximation.

(a) Inverter chain capacitances.

(b) Simplified circuit of Figure 2.13(a).

Figure 2.13: Lumping of capacitances at inverter output nodes.
2.5.3 Rise Time and Rise Delay

To derive the rise time \( t_r \) and the rise delay \( t_{PLH} \), consider the situation represented in Figure 2.11. Initially, the driver transistor TD is ON and the output voltage \( V_o \) (\( V_o = V_c \), the capacitor voltage) is near ground potential. Then the driver transistor TD is turned OFF by the input voltage step at \( t=0 \). As a result, for \( t > 0 \), \( I_o \) is assumed to be equal to zero and the currents \( I_L \) and \( I_c \) are given by:

a) When \( V_{ds} - V_o = V_{ds} > V_{gs} - V_{th} \) and \( V_{gs} > V_{th} \), the load transistor TL is in saturation region, and the currents are given by

\[
I_L = \frac{\mu_n C_{ox}}{2} \frac{W_L}{L_L} (V_{gs} - V_{th})^2
\]

\[
I_c = C_L \frac{dV_o}{dt}
\]

b) When \( V_{ds} - V_o = V_{ds} < V_{gs} - V_{th} \) and \( V_{gs} > V_{th} \), the load transistor TL is in linear region, the currents are given by

\[
I_L = \frac{\mu_n C_{ox}}{2} \frac{W_L}{L_L} [2(V_{gs} - V_{th})V_{ds} - V_{ds}^2]
\]

\[
I_c = C_L \frac{dV_o}{dt}
\]

From these equations we can derive the rise time \( t_r \) and the rise delay \( t_{PLH} \). Details are shown in Appendix A. The resultant equations are as follows:

\[
t_r = \frac{2C_i}{\lambda_n C_{ox} \frac{W_L}{L_L} V_{ds}} \left[ \frac{V_{ds}}{-V_{th}} \left( \frac{V_{ds} - V_i}{-V_{th}} \right) + \frac{V_{ds}/2}{-V_{th}} \cdot \frac{L_n \left( \frac{2}{V_{ds}} - \frac{2}{V_{ds} + X_s} \right)}{V_{ds} - X_s} \right] \tag{2.1}
\]

\[
t_{PLH} = \frac{2C_i}{\lambda_n C_{ox} \frac{W_L}{L_L} V_{ds}} \left[ \frac{V_{ds}}{-V_{th}} \left( \frac{V_{ds} - V_i}{-V_{th}} \right) + \frac{V_{ds}/2}{-V_{th}} \cdot \frac{L_n \left( \frac{2}{V_{ds}} - \frac{2}{V_{ds} + X_s} \right)}{V_{ds} - X_s} \right] \tag{2.2}
\]
where $W_l$ - width of the load channel

$L_l$ - length of the load channel

$X_1$, $X_2$ and $X_3$ - 10%, 90% and 50% points of the voltage swing

In a typical process, where $\mu_nC_{ox} = 2 \times 10^{-5}$ A/V$^2$, $V_{PD} = -0.8V_{DD}$, $V_{DD} = V_{DD}$ and $V_{DD} = 5V$,

$$t_r = 37.0 \times 10^3 \cdot \frac{(L_l/W_l) \cdot C_L}{(2.3)}$$

$$t_{PHL} = 16.1 \times 10^3 \cdot \frac{(L_l/W_l) \cdot C_L}{(2.4)}$$

2.5.4 Fall Time and Fall Delay

To derive the fall time $t_f$ and the fall delay $t_{PHL}$, consider the situation represented in Figure 2.11. Initially, the driver transistor TD is OFF and the output voltage $V_o$ is at $V_{DD}$. Then the driver transistor TD is turned ON by the input voltage step at $t=0$. As a result, for $t > 0$, the currents $I_d$, $I_L$ and $I_C$ are given by:

a) When $V_{DD} > V_o > V_{DD} - V_{PD}$, load transistor TL is in linear region and driver transistor TD is in saturation region, and the currents are given by:

$$I_L = \frac{\mu_nC_{ox}}{2} \cdot \frac{L_l}{W_l} \cdot \left[ \frac{1}{2}(-V_{PD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]$$

$$I_d = \frac{\mu_nC_{ox}}{2} \cdot \frac{L_l}{W_l} \cdot (V_{DD} - V_{PD})^2$$

$$I_C = C_L \cdot \frac{dV_o}{dt}$$
From the current equations given above, we can derive the time delay $\tau_d$. Details are shown in Appendix B. The resultant equations are as follows:

When $V_o < V_o < V_o + V_o$, the transistors are in the linear region, and the currents are given by:

\[
I_c = \frac{2(2V_o - V_o) - 2V_o}{2(2V_o - V_o) - 2V_o}
\]

\[
I_c = \frac{2(2V_o - V_o) - 2V_o}{2(2V_o - V_o) - 2V_o}
\]

\[
I_c = \frac{2(2V_o - V_o) - 2V_o}{2(2V_o - V_o) - 2V_o}
\]

\[
I_c = \frac{2(2V_o - V_o) - 2V_o}{2(2V_o - V_o) - 2V_o}
\]
\[
\tan^{-1} \left( \frac{\sqrt{V_{dd}^2 - V_{dd}^2}}{k(V_{dd}^2 - V_{dd}^2)^{1/2}} \right) - \tan^{-1} \left( \frac{V_{dd}}{k(V_{dd}^2 - V_{dd}^2)^{1/2}} \right) + \frac{V_{dd}}{k(V_{dd}^2 - V_{dd}^2)^{1/2}} \left( \ln \left| \frac{a_4 x_5 + a_2 - (a_4^2 - 4a_2 a_3)}{2a_4 x_5 + a_2 + (a_4^2 - 4a_2 a_3)} \right| \right) \]

\[2.6\]

and

\[a_1 = (k-1)\]
\[b_2 = 2V_{rB} + 2V_{dd} - 2k(V_{dd} - V_{FE})\]
\[c_2 = -2V_{rB} V_{dd} - V_{dd}^2\]
\[a_3 = k\]
\[b_3 = -2k(V_{dd} - V_{FE})\]
\[c_3 = (-V_{FE})^2\]

where

\[k = \frac{\omega_c}{\omega_s} \frac{L}{L_s}\]

In a typical process, where \(V_{FE} = 0.2V_{dd}\) and \(V_{rB} = -0.8V_{dd}\), the above equations become as follows:

i) for \(k = 4\),
\[t_f = 4.0624 \left( \frac{1}{\sqrt{C_{ox} V_{dd}}} \right) \left( \frac{1}{\omega_s} \right) \cdot C_L \]
\[t_{PHL} = 1.6197 \left( \frac{1}{\sqrt{C_{ox} V_{dd}}} \right) \left( \frac{1}{\omega_s} \right) \cdot C_L \]

ii) for \(k = 8\),
\[t_f = 3.8758 \left( \frac{1}{\sqrt{C_{ox} V_{dd}}} \right) \left( \frac{1}{\omega_s} \right) \cdot C_L \]
\[t_{PHL} = 1.6209 \left( \frac{1}{\sqrt{C_{ox} V_{dd}}} \right) \left( \frac{1}{\omega_s} \right) \cdot C_L \]
There is a set of equations for each value of $k$, since the discharging current also depends on the load transistor's geometric aspect ratio.

2.5.5 Transient Response of NMOS Circuits

In the previous section, we have derived all the delay time equations for an NMOS inverter. Here, we are going to extend the concepts to any multiple inputs gate such as NAND, NOR and any combination of series and parallel driver transistors. This is done by using the following steps.

First, transform the multiple inputs gate into an equivalent inverter. Second, obtain an equivalent aspect ratio, $(W/L)_{eq}$, for the driver transistors. Then, with this equivalent aspect ratio, we can use the equations for inverter to estimate the rise and fall times, rise and fall delays.

The following shows how NAND and NOR gates are being transformed into an equivalent inverter and the procedure for calculating the equivalent aspect ratio. For the NOR gate, only the active driver transistors are included for calculation of the equivalent aspect ratio.

For several devices connected in series whose geometric aspect ratios are $W_1/L_1, W_2/L_2, \ldots, W_n/L_n$,

$$
(W/L)_{eq} = \frac{1}{\frac{1}{L_1/W_1} + \frac{1}{L_2/W_2} + \ldots + \frac{1}{L_n/W_n}}
$$
The above expression can be proved (for $n=3$) as follows:

Case 1: When $V_i - V_{FE} \geq V > 0$, all transistors operate in the linear region.

$$V = V_{DS1} + V_{DS2} + V_{DS3}$$
\[
I_1 = \frac{1}{L_1} \left[ (V_i - V_{ds1} - V_{ds2} - V_{ds3}) V_{ds1} - \frac{V_{ds1}^2}{2} \right]
\]
\[
I_2 = \frac{1}{L_2} \left[ (V_i - V_{ds2} - V_{TE}) V_{ds2} - \frac{V_{ds2}^2}{2} \right]
\]
\[
I_3 = \frac{1}{L_3} \left[ (V_i - V_{TE}) V_{ds3} - \frac{V_{ds3}^2}{2} \right]
\]
Since \( I_1 = I_2 = I_3 \),
\[
\left( \frac{L_1}{L_1} \right) I = 1 \frac{1}{L_1} \left[ (V_i - V_{TE}) V_{ds1} - V_{ds1} \cdot V_{ds2} - V_{ds1} \cdot V_{ds3} - \frac{V_{ds1}^2}{2} \right]
\]
\[
\left( \frac{L_2}{L_2} \right) I = 1 \frac{1}{L_2} \left[ (V_i - V_{TE}) V_{ds2} - V_{ds2} \cdot V_{ds3} - \frac{V_{ds2}^2}{2} \right]
\]
\[
\left( \frac{L_3}{L_3} \right) I = 1 \frac{1}{L_3} \left[ (V_i - V_{TE}) V_{ds3} - \frac{V_{ds3}^2}{2} \right]
\]
Adding up the three equations, we have
\[
\left( \frac{L_1}{L_1} + \frac{L_2}{L_2} + \frac{L_3}{L_3} \right) I = 1 \frac{1}{L_1} \left[ (V_i - V_{TE}) (V_{ds1} + V_{ds2} + V_{ds3}) \right]
\]
\[
- \frac{L_1}{L_1} \left( V_{ds1}^2 + V_{ds2}^2 + V_{ds3}^2 + 2V_{ds1} \cdot V_{ds2} + 2V_{ds1} \cdot V_{ds3} + 2V_{ds2} \cdot V_{ds3} \right)
\]\n\[
= 1 \frac{1}{L_1} \left[ (V_i - V_{TE}) V - \frac{V^2}{2} \right]
\]
\[
I = \frac{1}{L_1} \left[ (V_i - V_{TE}) V - \frac{V^2}{2} \right]
\]

\[\left( \frac{L_1}{L_1} \right) \varphi = \frac{1}{L_1} \left[ (V_i - V_{TE}) V - \frac{V^2}{2} \right]
\]
\[
\left( \frac{L_1}{L_1} \right) \varphi = \frac{1}{2/L_1 + 2/L_2 + 2/L_3}
\]

- 33 -
Case 2: When $V > V_{th} > 0$, transistor $T_1$ saturates while $T_2$ and $T_3$ operate in the linear region.

$$I_1 = \mu_n C_{ox} \cdot \left( \frac{W_1}{L_1} \right) \cdot \frac{1}{2} (V_i - V_{th} - V_{th2} - V_{th3})^2$$

$$I_2 = \mu_n C_{ox} \cdot \left( \frac{W_2}{L_2} \right) \cdot \left[ (V_i - V_{th1} - V_{th2}) V_{th3} - \frac{V_{th3}^3}{3} \right]$$

$$I_3 = \mu_n C_{ox} \cdot \left( \frac{W_3}{L_3} \right) \cdot \left[ (V_i - V_{th1}) V_{th2} - \frac{V_{th2}^3}{3} \right]$$

$$\left( \frac{dI_1}{dV_i} \right)\frac{1}{I_1} = \mu_n C_{ox} \cdot \left[ \frac{1}{2} (V_i - V_{th})^2 + \frac{1}{2} V_{th2}^2 + \frac{1}{2} V_{th3}^2 - (V_i - V_{th}) V_{th3} - (V_i - V_{th}) V_{th2} V_{th3} \right]$$

$$\left( \frac{dI_2}{dV_i} \right)\frac{1}{I_2} = \mu_n C_{ox} \cdot \left[ (V_i - V_{th1}) V_{th2} - V_{th2} V_{th3} - \frac{V_{th2}^3}{3} \right]$$

$$\left( \frac{dI_3}{dV_i} \right)\frac{1}{I_3} = \mu_n C_{ox} \cdot \left[ (V_i - V_{th1}) V_{th3} - \frac{V_{th3}^3}{3} \right]$$

Adding up the three equations, we have

$$\left( \frac{dI_1}{dV_i} + \frac{dI_2}{dV_i} + \frac{dI_3}{dV_i} \right) \frac{1}{I} = \mu_n C_{ox} \cdot \frac{1}{2} (V_i - V_{th})^2$$

$$I = \mu_n C_{ox} \cdot \left( \frac{W_1}{L_1} \right) \frac{1}{2} (V_i - V_{th})^2$$

$$\left( \frac{W}{L} \right)_{eq} = \frac{1}{\frac{W_1}{L_1} + \frac{W_2}{L_2} + \cdots + \frac{W_n}{L_n}}$$

For several devices connected in parallel whose geometric aspect ratios are $W/L_1, W/L_2, \ldots, W/L_n$,

$$\left( \frac{W}{L} \right)_{eq} = \frac{W_1}{L_1} + \frac{W_2}{L_2} + \cdots + \frac{W_n}{L_n}$$
The above expression can be proved (for n=3) as follows:

**Case 1:** When \( V_i - V_{FE} \geq V > 0 \), all transistors operate in the linear region.

\[
I_1 = \mu_n C_{ox} \cdot \left( \frac{W}{L} \right) \cdot [(V_i - V_{FE})V - \frac{V^2}{2}]
\]

\[
I_2 = \mu_n C_{ox} \cdot \left( \frac{W}{L_2} \right) \cdot [(V_i - V_{FE})V - \frac{V^2}{2}]
\]

\[
I_3 = \mu_n C_{ox} \cdot \left( \frac{W}{L_3} \right) \cdot [(V_i - V_{FE})V - \frac{V^2}{2}]
\]

Since \( I = I_1 + I_2 + I_3 \), adding up the equations, we get

\[
I = I_1 + I_2 + I_3
\]

\[
= \mu_n C_{ox} \cdot \left( \frac{W}{L} + \frac{W_2}{L_2} + \frac{W_3}{L_3} \right) \cdot [(V_i - V_{FE})V - \frac{V^2}{2}]
\]

\[
\frac{W}{L} = \frac{W}{L_1} + \frac{W_2}{L_2} + \frac{W_3}{L_3}
\]

**Case 2:** When \( V \geq V_i - V_{FE} > 0 \), all transistors operate in the saturation region.
\[ I_1 = \frac{C_{ox}}{Z_r} \left( \frac{V_{i}}{2} - V_{p/e} \right)^3 \]

\[ I_2 = \frac{C_{ox}}{Z_2} \left( \frac{V_{i}}{2} - V_{p/e} \right)^3 \]

\[ I_3 = \frac{C_{ox}}{Z_3} \left( \frac{V_{i}}{2} - V_{p/e} \right)^3 \]

Adding up the three equations, we get

\[ \bar{I} = I_1 + I_2 + I_3 \]

\[ = \frac{C_{ox}}{Z_r} \left( \frac{V_{i}}{2} + \frac{V_{i}}{Z_2} + \frac{V_{i}}{Z_3} \right) \left( \frac{V_{i}}{2} - V_{p/e} \right)^3 \]

\[ \frac{V_{i}}{Z} C_{eq} = \frac{V_{i}}{Z_r} + \frac{V_{i}}{Z_2} + \frac{V_{i}}{Z_3} \]

The circuit below shows an inverter driving a pass transistor chain. The transition and delay times of the inverter can be calculated as before, but with the substitution of an equivalent load capacitance \( C_{eq} \) as given below:

\[ C_{eq} = \left( C_1 + \frac{1}{3} C_2 + \frac{1}{13} C_3 + \ldots + \frac{1}{n+1} C_n \right) \]

The above expression gives a reasonable approximation of the delay times and it can be shown with some examples.
As an illustration, the transition and delay times of the following circuit are calculated using the above equations. The results are compared with SPICE simulation results.

Example 1:

Given process parameters:
\[
W_x/L_x = 5/10, \quad W_{\alpha}/L_{\alpha} = 10/5, \quad W_{\rho}/L_{\rho} = 5/5 \quad \text{(all in \( \mu \text{m} \))}
\]
\[
V_{p,p} = 5V, \quad \mu n C_{ox} = 2 \times 10^{-5} \text{ A/V}\cdot\text{s}
\]

The equivalent load capacitance,
\[
C_{eq} = \left(0.02 + \frac{0.02}{N_2^2} + \frac{0.1}{N_2^2}\right) = 0.09 \text{ pF}
\]

From eqs. (2.3) and (2.4),
\[
t_r = 37 \times 10^3 \cdot \left(\frac{L_0}{S}\right) \cdot 0.09 \times 10^{-2} = 6.7 \text{ ns}
\]
\[
t_{PLN} = 16.1 \times 10^3 \cdot \left(\frac{L_0}{S}\right) \cdot 0.09 \times 10^{-2} = 2.9 \text{ ns}
\]

From eqs. (2.7) and (2.8),
\[
t_f = 40.624 \times 10^3 \cdot \left(\frac{S}{L_0}\right) \cdot 0.09 \times 10^{-2} = 1.8 \text{ ns}
\]
\[
t_{PHL} = 16.197 \times 10^3 \cdot \left(\frac{S}{L_0}\right) \cdot 0.09 \times 10^{-2} = 0.7 \text{ ns}
\]

<table>
<thead>
<tr>
<th>(ns)</th>
<th>Approximate Results</th>
<th>SPICE Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_r)</td>
<td>6.7</td>
<td>7.0</td>
</tr>
<tr>
<td>(t_f)</td>
<td>1.8</td>
<td>1.2</td>
</tr>
<tr>
<td>(t_{PLN})</td>
<td>2.9</td>
<td>1.3</td>
</tr>
<tr>
<td>(t_{PHL})</td>
<td>0.7</td>
<td>0.3</td>
</tr>
</tbody>
</table>
In the above delay estimation, we assumed that the equivalent load capacitance is a fixed value $C_{eq}$. In general, the load which is driven by a MOS inverter composed of pn junction depletion capacitances, the input gate capacitance of driven gate(s), and the net capacitance of a large number of stray or parasitic capacitances. All these capacitance values are function of voltage; thus, as the inverter output voltage changes so does the load capacitance. In spite of the simple model used in the derivation and the assumptions that we have made, the estimated delays are pessimistic and with a right order of magnitude.

Figure 2.14 shows a pass transistor with load capacitance $C_L$. The derivation of the transition and delay time equations are given in Appendix C. The resultant equations are as follows:

$$t_r = 17.778 \left( \frac{1}{\ln \frac{C_{ox}}{V_f}} \right) \cdot \left( \frac{L_p}{W_p} \right) \cdot C_L$$

$$t_{PLH} = 2 \left( \frac{1}{\ln \frac{C_{ox}}{V_f}} \right) \cdot \left( \frac{L_p}{W_p} \right) \cdot C_L$$

$$t_f = 2.7438 \left( \frac{1}{\ln \frac{C_{ox}}{V_f}} \right) \cdot \left( \frac{L_p}{W_p} \right) \cdot C_L$$

$$t_{PHL} = 1.0986 \left( \frac{1}{\ln \frac{C_{ox}}{V_f}} \right) \cdot \left( \frac{L_p}{W_p} \right) \cdot C_L$$

where

$V_f$ - final value at output node

$L_p$ - channel length of pass transistor

$W_p$ - channel width of pass transistor
The above equations can also be used for a pass transistor chain, see Figure 2.15, but with the substitution of $C_2$ and $W/L_0$ with $C_{eq}$ and $(W/L)_{eq}$, respectively, as given below:

$$C_{eq} = \frac{1}{N^2} \left( C_1 + \sqrt{2} C_2 + \cdots + \sqrt{N} C_N \right)$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{N} \left( \frac{W_0}{L_0} \right)$$

Figure 2.15: Pass transistor chain.

The above expression can be proved (for $n=3$) as follows:
Static analysis

\[ I_{DS1} = \mu_n C_{ox} \left( \frac{W}{L} \right) \frac{I}{2} (V_{DS} - V_I - V_T F)^2 \]

\[ I_{DS2} = \mu_n C_{ox} \left( \frac{W}{L} \right) \frac{I}{2} (V_{DS} - V_I - V_T F)^2 - (V_{DD} - V_I - V_T F)^2 \]

\[ I_{DS3} = \mu_n C_{ox} \left( \frac{W}{L} \right) \frac{I}{2} (V_{DS} - V_I - V_T F)^2 - (V_{DD} - V_I - V_T F)^2 \]

Adding up the three equations, we have

\[ 3I = \mu_n C_{ox} \left( \frac{W}{L} \right) \frac{I}{2} (V_{DS} - V_I - V_T F)^2 \]

Thus,

\[ I = \frac{1}{3} \left[ \mu_n C_{ox} \left( \frac{W}{L} \right) \frac{I}{2} (V_{DS} - V_I - V_T F)^2 \right] \]

Then,

\[ (V_{DS} - V_I - V_T F)^2 = \frac{1}{3} \cdot (V_{DD} - V_I - V_T F)^2 \]

\[ (V_{DD} - V_I - V_T F)^2 = \frac{3}{2} \cdot (V_{DD} - V_I - V_T F)^2 \]

\[ V_I = \frac{I}{N_3} \cdot V_3 + \left( 1 - \frac{I}{N_3} \right) \cdot (V_{DD} - V_T F) \]

\[ V_2 = \frac{N_3}{2} \cdot V_3 + \left( 1 - \frac{N_3}{2} \right) \cdot (V_{DD} - V_T F) \]

Adding capacitors to the circuit used for static analysis
\[ i_r = C_1 \frac{dV_r}{dt} = C_1 \sqrt{\frac{V_r}{L}} \frac{dV_r}{dt} \]
\[ i_2 = C_2 \frac{dV_2}{dt} = C_2 \sqrt{\frac{V_2}{L}} \frac{dV_2}{dt} \]
\[ i_3 = C_3 \frac{dV_3}{dt} = C_3 \sqrt{\frac{V_3}{L}} \frac{dV_3}{dt} \]

\[ I = i_r + i_2 + i_3 = \frac{1}{\sqrt{3}} \left( C_1 + \sqrt{\frac{V_2}{L}} C_2 + \sqrt{\frac{V_3}{L}} C_3 \right) \frac{dV_3}{dt} \]

**Equivalent Circuit**

\[ C_{eq} = \frac{1}{\sqrt{3}} \left( C_1 + \sqrt{\frac{V_2}{L}} C_2 + \sqrt{\frac{V_3}{L}} C_3 \right) \]

\[ \left( \frac{dV_3}{dt} \right)_{eq} = \frac{1}{3} \left( \frac{dV_3}{dt} \right) \]

**Generalization**
\[ C_{eq} = \frac{1}{\sqrt{N}} \left( \frac{1}{C_1} + \frac{1}{\sqrt{N} C_2} + \frac{1}{\sqrt{N} C_3} + \cdots + \frac{1}{\sqrt{N} C_N} \right) \]

\[ \left( \frac{\mu}{\sigma} \right)_{eq} = \frac{1}{\sqrt{N}} \left( \frac{\mu}{\sigma} \right) \]

More on Generalization

\[ C_{eq} = \frac{1}{\sqrt{\sum_{k=1}^{N} \frac{1}{\mu_j}}} \left( \sum_{k=1}^{N} \frac{1}{\mu_j} C_k \right) \]

\[ \left( \frac{\mu}{\sigma} \right)_{eq} = \frac{1}{\sqrt{\sum_{j=1}^{N} \frac{1}{\mu_i}}} \left( \sum_{j=1}^{N} \frac{1}{\mu_i} \frac{\mu}{\sigma} \right) \]

As an illustration, the delays for the following circuit are calculated using the above equations and the results are compared with SPICE simulation results.
Example 2:

Given process parameters:

\[ V_{DD} = 5 \text{V}, \mu_n C_{OX} = 2 \times 10^{-5} \text{A/V}^2, \ W/L = 5/5 \text{ (in } \mu \text{m}) \]

\[ V_{REF} = 0.2V_{DD} \]

So, \[ V_f = V_{DD} - V_{REF} = 4 \text{V}. \]

\[ \frac{\Delta V}{V} = \frac{4}{5} \cdot \frac{5}{5} = \frac{4}{5} \]

\[ C_{GS} = \frac{1}{V} \cdot (0.02 + \frac{1}{2} \times 0.1) = 0.1141 \text{ pF} \]

\[ t_r = 17.778 \left( \frac{1}{2 \times 10^{-5} \times \frac{5}{4}} \right)(2)(0.1141 \times 10^{-\Delta V}) = 50.7 \text{ ns} \]

\[ t_{PLH} = 2 \left( \frac{1}{2 \times 10^{-5} \times \frac{5}{4}} \right)(2)(0.1141 \times 10^{-\Delta V}) = 5.7 \text{ ns} \]

\[ t_f = 2.7438 \left( \frac{1}{2 \times 10^{-5} \times \frac{5}{4}} \right)(2)(0.1141 \times 10^{-\Delta V}) = 7.6 \text{ ns} \]

\[ t_{PHL} = 1.0986 \left( \frac{1}{2 \times 10^{-5} \times \frac{5}{4}} \right)(2)(0.1141 \times 10^{-\Delta V}) = 3.1 \text{ ns} \]

<table>
<thead>
<tr>
<th>(ns)</th>
<th>Approximate Results</th>
<th>SPICE Results</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_r )</td>
<td>50.7</td>
<td>47.5</td>
<td>7</td>
</tr>
<tr>
<td>( t_f )</td>
<td>7.8</td>
<td>7.1</td>
<td>10</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>5.7</td>
<td>5.8</td>
<td>-2</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>3.1</td>
<td>3.3</td>
<td>-6</td>
</tr>
</tbody>
</table>
2.6 Delay Calculations With Linear Ramp Input

In Section 2.5, we have derived the delay equations for the NMOS circuits. The simple model used in the derivation can occasionally produce large errors in delay estimate. There are two sources of error in the simple model. One source of error is the lumping of equivalent capacitance when there are pass transistors at the output node. This tends to overestimate the delays. Fortunately, for most circuits, the number of pass transistors is usually small and the error produced is not very significant.

The second and more significant source of error in the simple model comes from the assumption that the input waveform is a step function. Consider the situation of Figure 2.16. In (b), the inverter is driven by a linear ramp input waveform; the inverter will take much longer to drive its output load to zero than in (a), because the gate voltage is lower than \( V_{DSS} \) when the output is being driven in (b). In general, the switching delay of a transistor is a function of both the input waveform and the load being driven.

The solution to this problem is to include the effects of the input waveform. Each input is characterized by its slope (rise/fall time) and the input capacitance, \( C_I \). A rise/fall time of zero corresponds to a step function, and a large rise/fall time corresponds to a slowly rising or falling signal. Typically, if a gate is driving a large output load, or has small input capacitance, only a very slow input rise/
fall time will affect its delays. If a gate is driving a small output load, or has very large input capacitance, its delay will be more sensitive to the rise/fall time of the input waveform.

The following subsections show how input waveform is included in delay calculations.

2.6.1 Delays of an Inverter

Here, we analyze an inverter and the results can be applied to other NMOS gates. Figure 2.17 shows an NMOS inverter chain; the various input waveforms to gate $G_n$ are obtained by varying $C_{n-1}$, the load capacitance of the preceding gate $G_{n-1}$.

Figure 2.18 shows delay times of gate $G_n$ for various values of $C_n$ and $C_{n-1}$. The rise delay $t_{PLH}$ depends largely on
the load capacitance \( C_n \) but less on the input capacitance. The fall delay \( t_{\text{PHL}} \) is largely affected by the input waveform and the load capacitance \( C_n \).

From here onwards we are going to use a superscript * to indicate rise time or fall time that would occur at the output if the input were driven by a step function, e.g. \( t_r^* \) means the rise time of a gate with step function input. Since input waveform is characterized by rise/fall time and the input capacitance, we use a subscript I to indicate it as an input rise time or fall time, e.g. \( t_{rI} \) means the input rise time of the gate, and \( C_{I} \) is the input capacitance.

With the above notation, the delay times of the inverter can be approximately given by the following empirical equations by replacing \( C_n \) with \( C_L \) and \( C_{n-1} \) with \( C_I \).
\[ t_r = t_r^* \left( 1 + 0.3 \left( \frac{t_{rI}^*}{t_r^*} \right)^{0.5} \right) \]

\[ t_f^* = t_f^* \left( 1 + 0.25 \left( \frac{t_{rI}^*}{t_f^*} \right)^{0.5} \right) \]

\[ t_{PLH} = t_r / \left[ 2.9^2 + 0.15(C_I/C_L)^2 \right]^{1/2} \]

\[ t_{PML} = t_f^* / \left[ 1.9^2 + 0.06(C_I/C_L)^3 \right]^{1/2} \]

Here, \( t_r^* \) and \( t_f^* \) are calculated as in Section 2.5. All the empirical constants are obtained from circuit simulation.

As an illustration, the delays for gate \( G_n \) in Figure 2.17 are estimated using the above empirical equations. First, assuming the input waveform to gate \( G_{n-1} \) is a step function, the output will have various waveform by varying \( C_{n-1} \). The following results are obtained by using equations from Section 2.5.

<table>
<thead>
<tr>
<th>( C_{n-1} ) (pF)</th>
<th>( t_r ) (ns)</th>
<th>( t_f^* ) (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>1.5</td>
<td>0.4</td>
</tr>
<tr>
<td>0.1</td>
<td>7.4</td>
<td>2.0</td>
</tr>
<tr>
<td>0.2</td>
<td>14.8</td>
<td>4.1</td>
</tr>
<tr>
<td>0.4</td>
<td>29.6</td>
<td>8.1</td>
</tr>
<tr>
<td>1.0</td>
<td>74.0</td>
<td>20.3</td>
</tr>
<tr>
<td>2.0</td>
<td>147.9</td>
<td>40.6</td>
</tr>
</tbody>
</table>

These output waveforms will become the input waveforms for gate \( G_n \). For gate \( G_n \), first, the rise time \( t_r^* \) and fall time \( t_f^* \) are calculated with equations from Section 2.5.
Then, using the empirical equations, the output transition and delays are calculated as follows:

Given process parameters:

\[ V_{bb} = 5V, \mu_n C_{ox} = 2 \times 10^{-5} \text{ A/V}^2 \]

\[ W/L = 5/10, W/L_{sh} = 10/5 \text{ (all in } \mu \text{m)} \]

\[ C_n = 0.2 \text{ pF} \]

So,

\[ t_{f^r} = 3.6976 \left( \frac{1}{\mu_n C_{ox} V_{bb}} \right) \left( \frac{W}{L} \right) C_n = 14.8 \text{ ns} \]

\[ t_f^d = 4.0624 \left( \frac{1}{\mu_n C_{ox} V_{bb}} \right) \left( \frac{W}{L} \right) C_n = 4.1 \text{ ns} \]

and

\[ t_r = 14.8 \left[ 1 + 0.3 \left( \frac{t_{f^r}}{14.8} \right)^{.5} \right] \]

\[ t_{PLH} = t_r / \left[ 2.9^a + 0.15(C_n-1/0.2)^a \right]^a \]

\[ t_f = 4.1 \left[ 1 + 0.25(t_{r^r}/4.1)^{.5} \right] \]

\[ t_{FHL} = t_f / \left[ 1.9^a + 0.06(C_n-1/0.2)^a \right]^a \]

The results are given in the following table and plotted in Figure 2.18.
The results obtained by SPICE simulation program are given in the following table and also plotted in Figure 2.18.

<table>
<thead>
<tr>
<th>( \frac{C_{n-1}}{C_n} )</th>
<th>( t_r ) (ns)</th>
<th>( t_f ) (ns)</th>
<th>( t_{PLH} ) (ns)</th>
<th>( t_{PHL} ) (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02/0.2</td>
<td>14.8</td>
<td>4.4</td>
<td>5.1</td>
<td>2.3</td>
</tr>
<tr>
<td>0.1/0.2</td>
<td>15.0</td>
<td>6.0</td>
<td>5.2</td>
<td>3.2</td>
</tr>
<tr>
<td>0.2/0.2</td>
<td>15.4</td>
<td>8.3</td>
<td>5.3</td>
<td>4.3</td>
</tr>
<tr>
<td>0.4/0.2</td>
<td>16.6</td>
<td>13.1</td>
<td>5.5</td>
<td>6.7</td>
</tr>
<tr>
<td>1.0/0.2</td>
<td>21.9</td>
<td>28.8</td>
<td>6.3</td>
<td>12.7</td>
</tr>
<tr>
<td>2.0/0.2</td>
<td>35.0</td>
<td>57.0</td>
<td>7.2</td>
<td>18.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \frac{C_{n-1}}{C_n} )</th>
<th>( t_r ) (ns)</th>
<th>( t_f ) (ns)</th>
<th>( t_{PLH} ) (ns)</th>
<th>( t_{PHL} ) (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02/0.2</td>
<td>14.9</td>
<td>4.3</td>
<td>5.1</td>
<td>2.3</td>
</tr>
<tr>
<td>0.1/0.2</td>
<td>15.5</td>
<td>6.4</td>
<td>5.1</td>
<td>3.9</td>
</tr>
<tr>
<td>0.2/0.2</td>
<td>16.3</td>
<td>9.0</td>
<td>5.1</td>
<td>5.4</td>
</tr>
<tr>
<td>0.4/0.2</td>
<td>18.3</td>
<td>14.0</td>
<td>5.3</td>
<td>7.0</td>
</tr>
<tr>
<td>1.0/0.2</td>
<td>24.0</td>
<td>28.0</td>
<td>6.0</td>
<td>11.5</td>
</tr>
<tr>
<td>2.0/0.2</td>
<td>35.0</td>
<td>50.5</td>
<td>6.8</td>
<td>17.0</td>
</tr>
</tbody>
</table>
Figure 2.18: Delay times $t_{\text{pH}}$ and $t_{\text{pHL}}$ for various capacitive load conditions.

The dotted lines are the results obtained by circuit simulation.
2.6.2 Delays of a Pass Transistor Chain

In an earlier section we have calculated the delay times for a pass transistor chain with step input. Here, a pass transistor chain which is driven by an inverter, as shown in Figure 2.19, is considered.

To calculate the delays of the pass transistor chain, first, the output transition times of the inverter are calculated. Then, with these transition times as the input of the pass transistor chain, the delays of the pass transistor chain can be approximately given by the following empirical equations:

\[ t_r = t_r^{\ast} \left[ 1 + 0.3 \left( \frac{t_{rI}}{t_r^{\ast}} \right)^{0.5} \right] \]

\[ t_{PLH} = \frac{t_r}{8.9} \]

\[ t_f = t_f^{\ast} \left[ 1 + \left( \frac{t_{fI}}{t_f^{\ast}} \right)^{0.5} \right]. \]

\[ t_{PHL} = \frac{t_f}{2.5} \]

Here, \( t_r^{\ast} \) and \( t_f^{\ast} \) are calculated as in Section 2.5. All the empirical constants are obtained from circuit simulation.
Figure 2.19: A pass transistor chain driven by an inverter.

As an illustration, we will show how the delay times of the circuit in Figure 2.19 are calculated using the above empirical equations.
Example 3

Given process parameters:

\[ V_{DD} = 5V, \mu n C_{OX} = 2 \times 10^{-5} \text{A/V}^2 \]
\[ W_L/L_L = 5/10, W_{oL}/L_{oL} = 10/5, W_P/L_P = 5/5 \text{ (in \mu m)} \]
\[ V_{P.D} = -0.8V_{DD}, V_{P.H} = 0.2V_{DD} \]

First, we calculate the transition times of the inverter. As we have shown in Example 1, the inverter rise and fall times are given by:

\[ t_r = 6.7 \text{ ns} \]
\[ t_f = 1.8 \text{ ns} \]

Next, the transition times of the pass transistor chain with step input waveform are calculated. As we have shown in Example 2, the rise and fall times of the pass transistor chain are given by:

\[ t_{r'} = 50.7 \text{ ns} \]
\[ t_{f'} = 7.8 \text{ ns} \]

With the above values and the empirical equations, the transition and delay times can then be calculated as follows:

\[ t_r = 50.7 \left[ 1 + 0.3 \left( \frac{6.7}{50.7} \right)^{0.5} \right] = 51.4 \text{ ns} \]
\[ t_{PLH} = 51.4/8.9 \approx 5.8 \text{ ns} \]
\[ t_f = 7.8 \left[ 1 + \left( \frac{1.8}{7.8} \right)^{0.5} \right] = 10.6 \text{ ns} \]
\[ t_{PHL} = 10.6/2.5 = 4.2 \text{ ns} \]

- 53 -
<table>
<thead>
<tr>
<th>(ns)</th>
<th>Approximate Results</th>
<th>SPICE Results</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_r$</td>
<td>51.4</td>
<td>48.5</td>
<td>6</td>
</tr>
<tr>
<td>$t_f$</td>
<td>10.6</td>
<td>10.5</td>
<td>1</td>
</tr>
<tr>
<td>$t_{PLH}$</td>
<td>5.8</td>
<td>5.5</td>
<td>5</td>
</tr>
<tr>
<td>$t_{PHL}$</td>
<td>4.2</td>
<td>5.0</td>
<td>-16</td>
</tr>
</tbody>
</table>

In this example, a different pass transistor structure, pass transistor tree, is considered. Figure 2.20 shows a pass transistor tree driven by an inverter. The delay calculation considers only the direct path between the pullup node (node 3) and the output node (node 7). All the side capacitances are assumed to be connected to the branch nodes. This will reduce the tree structure into a chain structure, and the previous equations can then be applied to delay calculation.

Example 4

![Diagram](image)

**Figure 2.20:** A pass transistor tree driven by an inverter.
Given process parameters:

\[ V_{DD} = 5V, \mu A C_{ox} = 2 \times 10^{-5} \text{ A/V}^2 \]

\[ W_c / L_e = 5/10, W_e / L_e = 10/5, W_p / L_p = 5/5 \text{ (in } \mu \text{m)} \]

\[ V_{RTOS} = -0.8V_{DD}, V_{REF} = 0.2V_{DD} \]

The first step is to calculate the equivalent load capacitance, and the rise and fall times of the inverter.

\[ C_{eq} = \left( 0.01 + \frac{W_c}{L_e} \times 0.06 + \frac{W_e}{L_e} \times 0.01 + \frac{W_p}{L_p} \times 0.1 \right) = 0.11 \text{ pF} \]

From equations (2.3) and (2.7),

\[ t_r = 37 \times 10^{-3} \times \left( \frac{V_c}{1.5} \right) \times 0.11 \times 10^{-12} = 8.1 \text{ ns} \]

\[ t_f = 40.624 \times 10^{-3} \times \left( \frac{V_c}{1.5} \right) \times 0.11 \times 10^{-12} = 2.2 \text{ ns} \]

The second step is to reduce the transistor tree into a chain structure, as shown below:

So, \( V_f = V_{RTOS} - V_{REF} = 4V \)

\[ \frac{(W/L)_{eq}}{2} = \frac{1}{3} \left( \frac{W_c}{L_e} \right) = \frac{1}{3} \]

\[ C_{eq} = \frac{1}{\sqrt{3}} \left( 0.06 + \sqrt{3} \times 0.01 + \sqrt{3} \times 0.1 \right) = 0.14 \text{ pF} \]

\[ t_r = 17.778 \left( \frac{V_c}{2 \times 10^{-3} \times 4} \right)(3)(0.14 \times 10^{-12}) = 93.3 \text{ ns} \]

\[ t_f = 2.7438 \left( \frac{V_c}{2 \times 10^{-3} \times 4} \right)(3)(0.14 \times 10^{-12}) = 14.4 \text{ ns} \]

Then, using the empirical equations,
\[
\begin{align*}
\tau &= 93.3 \left[ 1 + 0.3 \left( \frac{8.1}{93.3} \right)^{\frac{1}{3}} \right] = 94 \text{ ns} \\
\tau_{\text{PLH}} &= \frac{94}{8.9} = 10.6 \text{ ns} \\
\tau_f &= 14.4 \left[ 1 + \left( \frac{2.2}{14.4} \right)^{\frac{1}{3}} \right] = 18.3 \text{ ns} \\
\tau_{\text{PHL}} &= \frac{18.3}{2.5} = 7.3 \text{ ns}
\end{align*}
\]

<table>
<thead>
<tr>
<th>(ns)</th>
<th>Approximate Results</th>
<th>SPICE Results</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau)</td>
<td>94.0</td>
<td>81.0</td>
<td>16</td>
</tr>
<tr>
<td>(\tau_f)</td>
<td>18.3</td>
<td>16.5</td>
<td>11</td>
</tr>
<tr>
<td>(\tau_{\text{PLH}})</td>
<td>10.6</td>
<td>9.0</td>
<td>18</td>
</tr>
<tr>
<td>(\tau_{\text{PHL}})</td>
<td>7.3</td>
<td>8.3</td>
<td>-12</td>
</tr>
</tbody>
</table>
Chapter III

LOGIC SIMULATION

3.1 Introduction

Logic simulation is the process of building and exercising a model of a digital circuit on a computer. By exercising, it means the evaluation of signal values in the modeled circuit as a function of time for some applied input sequence [4].

There are two main applications for a logic simulator. The first is in the logic and/or design verification of a new design. In logic verification, the logic designers simply verify the logical correctness of the design. While in design verification, they are interested in testing for logical correctness, as well as timing and signal propagation characteristics.

A second application for logic simulation is in the area of fault analysis. Here, the logic designer may desire information related to what faults are detected by a proposed test sequence, what are the operational characteristics of the circuit under specific fault conditions or what degree of fault resolution is obtainable with a given test sequence? These and other questions can be dealt with effectively by the process of fault simulation.
The usefulness of a logic simulator can be evaluated with respect to the following attributes:

1. Accuracy - There should be a close correspondence between predicted signal value in time as calculated by the simulator, and that which occurs in the actual circuit.

2. Efficiency - The process of simulation must be cost effective.

3. Generality - The simulator should be able to handle a broad class of circuits.

In many situations, the accuracy considerations can be thought of as being primarily theoretical, and the cost effectiveness as primarily practical implementation consideration.

This chapter gives an overview of logic simulation, detailed discussion can be found in [4], and as a background for the understanding of the remaining chapters.
3.2 Input Circuit Description

The circuit description can be supplied to the simulator in various forms, and a preprocessor translates the description into a form (data structure) used internally. At least three possibilities exist for the input: Boolean equations, macro, and elements. The Boolean equation input would allow the user to describe the circuit in an equation form. For example:

\[ Z = A \cdot B + C \]

would generate the circuit shown in Figure 3.1, via a Boolean preprocessor.

![Diagram](image)

**Figure 3.1:** Circuit generated from Boolean equation

The macro input circuit description is in the forms of a procedure for highly repetitive logic circuits. For example the 32-bit register, as shown in Figure 3.2, is made up of 32 JK flip-flops. The macro allows the user to describe the flip-flop once at the gate level, and then use this description for each of the 32 bits of the register. This would result in a considerable simplification of the description process.
Figure 3.2: Macro representation of a 32-bit register

The element description is similar to the gate level description except that an element can have multiple inputs and outputs whereas a gate has a single output. Figure 3.3 shows an element representation of a JK flip-flop.

Figure 3.3: Element representation of a JK flip-flop
In a switch-level simulator, circuits are described by a set of nodes connected by transistor switches. The input circuit description consists of a list of transistors, each one consisting of a gate, a source, and a drain node, and other pertinent information, such as the width/length (W/L) aspect ratio for each transistor and the capacitance at each node. This input circuit description can usually be extracted directly from the mask layout; therefore, no knowledge of the logic function of the circuit is required. More detailed description on circuit representation in switch-level simulator is presented in Chapter 4.
3.3 Delay Modeling

Delay is an important attribute to the correct functional operation of many circuits. We can categorize most delay modeling techniques as being either pessimistic or optimistic models of the real circuit. Usually a more detailed delay model is used for design verification than is used for logic verification. In general, simulation delay models used can be categorized as zero, unit, rise/fall and precise delays.

As the timing precision of the model increases, so does the complexity of the implementation, and this complexity is often a major consideration. However, the implementation complexity should not be the dominant consideration. The principal applications area of the simulator is a more important consideration. Zero and unit (single delay value) delay simulators can be used for logic verification, but are not effective for design verification to catch timing errors. In the unit delay model, a single delay value is assigned to each type of logic element. Some devices, however, have different signal rise and fall times due to various electrical parameters such as input waveforms, input capacitances and output capacitances. Such devices can be modeled by assigning two delay values; one for a transition from 0 to 1, and another for a transition from 1 to 0. This model is referred to as a rise/fall delay model, which closely approximates the timing properties for some technologies.
Another timing model is the min/max delay model. These delays defined an ambiguous time region in which the actual signal value changes somewhere in the region. Yet another is called an inertia delay model. To switch a logic gate, some minimum energy is required. In the inertia delay model, this minimum energy is modeled as a minimum pulse width required to switch the logic gate. Figure 3.4 shows the input and output timing diagrams of an inverter for various delay models.

![Timing Diagram](image)

**Figure 3.4:** Input and output timing diagrams of an inverter for various delay models
3.4 Multiple Logic Values

The choice of particular delay parameters directly affects the number of logic values used in the simulator. Zero and unit delay models can use either two or three logic values for simulation. The values would be 0, 1 and X, where X is indeterminate or unknown. Two-valued (0 and 1) models are not sufficient in design verification. The problem is in the initialization of the circuit to be simulated. Since only two values exist, all signals would be initially set to 0 or 1. Consistency cannot be easily maintained. Also, the abnormal states due to timing errors cannot be represented. Thus, most simulators employ at least three logic values.

The rise/fall delay models use three or four logic values. The four logic values are 0, 1, X and E, where X is the initial unknown state and E represents errors. The min/max delay model requires two additional logic values of U (signal rising) and D (signal falling). For tri-state devices and transmission gates, a value Z is introduced to represent a high impedance state.

To model the circuits correctly, more logic values would be required, but this will generally lengthen the simulation time. Therefore, the selection of the number of logic values in implementation is dependent on the purpose of simulator.
This multiple delay simulator uses three logic values: 0, 1, and X, where X represents the initial unknown state and also the timing errors. A detailed description of this delay model is presented in Chapter 4.

3.5 Simulator Structures

There are two basic classes of simulators: compiled code and table-driven event-directed [4]. In compiled code simulators, the circuit to be simulated is actually represented internally as a sequence of computer instructions. The table-driven event-directed simulator is more versatile in handling delays and achieves computational efficiency. Therefore, table-driven event-directed simulation is described in the following section.

3.5.1 Event Directed Simulation

Simulation of a logic circuit starts from building a circuit model, which consists of the descriptions of circuit elements and their interconnections. Then an input sequence is introduced to the input terminals at predetermined times. The circuit will be simulated at discrete time steps by evaluating the response to the applied input stimuli. If any circuit elements change states, they are scheduled to change after some appropriate delay. Simulation process continues until the circuit stabilizes (no more inputs to simulate and the circuit reaches a steady state).
It is generally observed that at any simulation time step, only a small percentage of all the signal lines changes values, typically 2-10%. Thus, it appears wasteful, as is done with the compiled code approach, to simulate all the circuit elements when only a small portion of the circuit is actively changing. Another fact is that the output of a circuit element will change its value only when at least one of its inputs changes. An event is defined as a change in logic value of a signal line. Based on the facts above, circuit elements need to be evaluated only when an event occurs at one or more of its inputs.

When an event occurs, the outputs of all those elements with inputs connected to that line are potentially active and require evaluation. As one simulates all the potentially active circuit elements, one will find that only some have the output values changed while others do not. A significant reduction in computation can be achieved by simulating only the potentially active elements rather than all the elements. The technique is referred to as event directed or selective trace (tracing after events).

3.5.2 Event Scheduling

When a circuit element is evaluated at $t_o$, and it has been determined that its output node has changed its logic state, then this output event is scheduled at the future time step $t_o + \Delta$, where $\Delta$ is the delay time for this change to take place. The technique used to schedule and reschedule events is called the time-flow mechanism.
One way to organize the scheduling of events is to use a list structure as shown in Figure 3.5, where all the events occurring at the same time are linked together. The headers of these list are stored in increasing order, e.g., $t_i < t_j < t_k$ as illustrated.

*Figure 3.5: List structure scheduler*
Chapter IV
MULTIPLE DELAY SWITCH-LEVEL SIMULATION ALGORITHM

4.1 Introduction

In simulating MOS circuits accurately, one of the problems is the accuracy of delay time. The delay time of a MOS gate depends not only on the output loading but also the input waveform.

The traditional approach to delay calculation is the use of circuit simulation programs such as SPICE [15]. In this case, circuit delays are obtained by solving a set of integral-differential equations. This approach produces highly accurate delays. However, this accuracy does not come without cost: intolerably long computing time. Thus, this approach is impractical for today's state-of-the-art LSI/VLSI circuits, which contain tens or hundreds of thousands of transistors.

Another approach is the use of a switch-level logic simulator such as MOSSIM [5] and others [3] [17]. Unfortunately, this type of simulator predicts accurately the logic levels without regard to the circuit delays due to the unit delay model used.

In this chapter, we shall describe a switch-level logic simulation algorithm that takes circuit delays into account.
It is an extension to the existing unit delay algorithm [5], but with a multiple delay model added. This delay model takes into consideration the effects of not only the output capacitive loading but also the input waveforms.

4.2 Network Model

A switch network [5][6][7] consists of a set of nodes, \( n_1, n_2, \ldots, n_n \), connected by a set of transistors \( t_1, t_2, \ldots, t_m \). Each node \( n_i \) is in a state \( s_i \) as one of 0, 1, and \( X \), where 0 and 1 represent the low and high voltage level, respectively, and \( X \) indicates an indeterminate or unknown voltage level. Each node \( n_i \) is classified as either an input node or a storage node. Input nodes provide strong signals and storage nodes provide weak signals. By signal strength we mean the current supplying or sinking capability of the circuit node. The signal strengths are defined as follows:

1. A node with driver strength (D) provides a strong signal to the circuit being driven, and the signal is not affected by the circuit action. Typical examples of nodes with driver strengths are signal sources such as \( V_{CC}, GND \), and clocks.

2. A node connected to \( V_{SS} \) through a resistor or a depletion pull-up transistor is considered to have the pull-up strength (L). This strength has sufficient driving capability but weaker than nodes with D strength.
3. All other nodes are of the weak strength \( W \). Typically, these nodes are internal circuit nodes connecting enhancement transistors. Such nodes are always connected to GND through stray capacitances due to interconnections or the gate capacitance of a transistor.

A signal is represented by a pair \( (s_i, f_i) \), consisting of its logic state \( s_i \) and its strength \( f_i \). Figure 4.1 illustrates the representation of a network.

![Diagram](image)

**Figure 4.1:** Illustrations of the node states and node strengths of a circuit.

A transistor is a three terminal device with terminals labelled "Drain", "Gate", and "Source" as shown in Figure 4.2. No distinction is made between the drain and the source. Each transistor has a type \( n \), \( p \), or \( d \), corresponding to the n-type enhancement, p-type, or n-type depletion devices, respectively. A transistor acts as a resistive switch connecting its drain and source controlled by its gate. Each transistor \( t_i \) has a state \( z_i \) open, closed, or
unknown, where "open" indicates non-conducting and "closed" conducting. A transistor in the "unknown" state shows an indeterminate conductance, somewhere in-between the conductance of fully open and fully closed. Table 4.1 shows the transistor types and their behaviour.

![MOS transistor diagram](image)

**Figure 4.2:** A MOS transistor.

<table>
<thead>
<tr>
<th>n-type</th>
<th>p-type</th>
<th>d-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>gate node state (effect)</td>
<td>gate node state (effect)</td>
<td>gate node state (effect)</td>
</tr>
<tr>
<td>0 open</td>
<td>0 closed</td>
<td>0 closed</td>
</tr>
<tr>
<td>1 closed</td>
<td>1 open</td>
<td>1 closed</td>
</tr>
<tr>
<td>X unknown</td>
<td>X unknown</td>
<td>X closed</td>
</tr>
</tbody>
</table>

**Table 4.1:** Transistor types and their behaviour

Nodes in a subnetwork are categorized as external or internal. All nodes with connections to circuit elements
outside the subnetwork are external nodes. They are points of interest because events at these nodes cause scheduling of evaluations of other subnetworks.

4.2.1 Network Representation

We assume that we have a logic network in which all the depletion load transistors are replaced by pull-up nodes. Then we can represent the structure of a network, \( \mathcal{N} \), in the following way:

\[
\mathcal{N} = (N, T)
\]

where

\[
N = \{ n_i \}
\]
denotes the set of nodes, and

\[
T = \{ t_i \}
\]
denotes the set of transistors. The state of the network is defined as:

\[
M = (S, Z)
\]

where

\[
S = \{ s_i \}
\]
denotes the node states, and

\[
Z = \{ z_i \}
\]
denotes the transistor states.
Information concerning the nodes and transistors is encoded by the following mappings:

- **NTYPE(N) → (internal, external)** node type
- **TTYPE(T) → (n, p, d)** transistor type
- **DRAIN(T) → N** drain node
- **GATE(T) → N** gate node
- **SOURCE(T) → N** source node

For example, see Figure 4.1, if we are given a node \( n_x \), the mapping **NTYPE\( (n_x) \)** will give the information that node \( n_x \) is an external node.

### 4.2.2 Partitioning of the Network

In order to apply selective trace of activities and time flow mechanism, a network is partitioned into its connected components, referred to as subnetworks. The nodes and transistors belonging to a subnetwork \( S_u \) are identified as follows:

1. The nodes of \( S_u \) are obtained by partitioning the graph, formed by creating a vertex for each node and an edge between each pair of vertices that constitutes the source and drain for some transistor, into its connected components.

2. A transistor \( t_i \) is in the subnetwork \( S_u \) if a node of \( S_u \) is a source or drain node of \( t_i \).

All nodes with connections to circuit elements outside the subnetwork are external nodes. The remaining nodes of
subnetwork are internal nodes. Figure 4.3 shows how a network is being partitioned into subnetworks.

(a) Logic diagram of a static register.

(b) Circuit diagram of network after node partitioning.

Figure 4.3: Network partitioning
4.3 Delay Model

In this section, a delay model is described. This delay model is organized around a subnetwork, instead of logic gates. The use of subnetwork permits both logic gates and pass transistors to be handled in a uniform fashion.

In any given time, the modeler considers a change in value at a particular transistor gate, called the trigger. The delay modeler is given the sizes and types of the transistors, the parasitic capacitances and the capacitance of the nodes of each subnetwork. If the trigger is the last transistor to turn on in a subnetwork, as shown in Figure 4.4(b), it may cause certain other transistor gates or output nodes to change value at later times. The modeler is also given information about the waveform at the input. Its job is to use this information to compute the waveform at the output of the subnetwork.

To compute the delay, first, the modeler starts at the source terminal of the trigger transistor and searches through sources and drains of other transistors to find all paths from the trigger source to $V_{DD}$ or GND. For each such path, it then searches from the drain of the trigger transistor through sources and drains to all outputs. The path from the output through the trigger to $V_{DD}$ or GND is used to compute the delay from trigger to output. Since all transistor sizes between trigger and output are known and the capacitances at each node are given, it can compute the delays
from these values by using the equations derived in Section 2.5.

Figure 4.4 (b) shows the trigger transistor and the output nodes of subnetwork $S_1$. In the case when the trigger transistor is turned off, as shown in Figure 4.4(c), it does not belong to any subnetwork. Then the path from the output node to the $V_{ab}$ is used to compute the delays.
(a) A given circuit for delay calculation

(b) The trigger transistor is turned ON

(c) The trigger transistor is turned OFF

Figure 4.4: Searching of direct paths.
4.4 Simulation Algorithm

The techniques used in this algorithm are: event-driven, time flow mechanism with time queue array for simulation with variable rise and fall delays, selective trace of active path.

Scheduling of events, tracing of subnetworks for evaluation, and subnetwork evaluation proceed as follows:

1. Event scheduling - When an event occurs at a node, the time scheduled for the event is determined from the delay associated with the transition, and the node identifier together with the new node value is appended to the event list for that time slot.

2. Selective trace - For each node on the event list for the present time, all subnetworks which receive signals on the node as an input are selected for evaluation.

3. Subnetwork evaluation - After a subnetwork is evaluated, all resulting events at the output nodes are to be scheduled at appropriate future times.

The evaluation of new node values reflects an instantaneous reaction of the subnetwork due to the changing input states. This new state of the subnetwork is only a projected state which would be reached only after some delay if the input states were frozen for a sufficiently long time. But the inputs can change before this projected state is reached. When this happens the evolution is directed
towards another state and the previously scheduled delays are revised. Thus, a scheduled transition may be rescheduled or cancelled resulting in a spike condition. A spike is propagated as an $X$ pulse, the pulse width of which is determined from the delay between the events that caused it.

There are three types of transitions and their respective delays for scheduling are as follows:

1. $0 \rightarrow 1$ and $X \rightarrow 1$ transitions are scheduled with rise delays.
2. $1 \rightarrow 0$ and $X \rightarrow 0$ transitions are scheduled with fall delays.
3. $0 \rightarrow X$ and $1 \rightarrow X$ transitions are scheduled with unit delay since occurrence of an $X$ during simulation indicates usually an error condition.

Figure 4.5 shows how to keep track of scheduled events at the time queue (TQ). PT is the present time pointer pointing to event list for the present time. The event list contains descriptions for nodes which change states during that time slot. MTEL is the macro time event list or overflow event list, which contains events that are to occur at times which are beyond the range of the current TQ array time frame. Detailed discussion of the lists is presented in the next chapter.
Figure 4.5: Simulation time flow mechanism (Time Queue).

During simulation, errors (e.g. spikes) are indicated by an X. For example, Figure 4.6 shows a circuit with rise delay $t_{\text{PH}} = 5$ units and a fall delay $t_{\text{PL}} = 2$ units. If a negative pulse with a width of 2 units is applied on the input, an abnormal situation occurs. That is, the output change caused by the first input change occurs later than that by the second input change (the first input change causes an output transition from 0 to 1 at time 7 units, the second input change causes an output transition from 1 to 0 at time 6 units). The output would be assigned an error state X with a pulse width of 1 unit.
Figure 4.6: Input and output timing diagrams of an error condition.

A simplified event-driven simulation algorithm is flowcharted in Figure 4.7. This algorithm will be presented in a procedural form augmented by explanatory comments.
Figure 4.7: Event-Driven Simulation Algorithm.
Figure 4.7 (continued)
Procedure SIMULATION is the main body of the algorithm; it shows how the simulation is performed. The procedure is divided into five phases, namely, initialization, node state update, transistor state update and subnetwork repartitioning, subnetwork evaluation, and output scheduling. In the first phase, the given network is partitioned into its connected components called SUBNETWORKS. All nodes in the subnetworks are initialized as follows: Nodes with pullup strength, L, are initialized to 1 and nodes with weak strength, W, are initialized to X. The input list A which contains input events \( \{n_i, y_i, T_j\} \), where \( T_j \) is the time when node \( n_i \) has its new state \( y_i \), are scheduled on the Time Queue.

In the second phase, events for the present time is assessed one by one. For each event, the node value is updated according to the change in the event. In the third phase, the state of transistors for which this node is their gate terminals is updated by using the procedure SET-TRANS. Updating a node value may cause some subnetworks, which having this node as an input, to be added to the evaluation list EVL. All the nodes in a subnetwork on the EVL form an undirected graph which has a vertex for each node and an edge for each transistor which is in the "closed" or "unknown" state. The connected components of this graph becomes SUBNETS.
In the fourth phase, the node states of each subnet is evaluated by procedure NODESTATE. Delays for output nodes which have changed states are computed. The last phase of evaluation is to schedule new events on the Time Queue. Evaluation of state and delay are described in later sections.

Procedure APPEND adds a subnetwork onto the evaluation list only if the given subnework is not already on the list. Then, this subnetwork is deleted from SUBNETWORKS, because it will be repartitioned into new subnetworks during the execution of EVL.
**Input:** A network \( \Omega \), a network configuration \( M \) and an input list \( A \), containing sequence of input events to be simulated \( \langle n_i, y_i, T_j \rangle \).

**Output:** \( M \) is updated by simulating the input events in \( A \).

**procedure** SIMULATION (\( \Omega \), M, A);

begin

(* initialization phase *)

\( G \) --- form an undirected graph with \( V = \{ n_i \} \) and \( E = \{ \langle i, j \rangle \mid \text{for some } \tau^g : \text{SOURCE}(\tau^g) = n_i \text{ and } \text{DRAIN}(\tau^g) = n_j \} \);

\( \text{SUBNETWORKS} \) --- PARTITION(\( G \));

for each \( n_i \) such that \( f_i = 'L' \) do \( s_i = 1 \);

for each \( n_i \) such that \( f_i = 'W' \) do \( s_i = X \);

for each \( \langle n_i, y_i, T_j \rangle \) in \( A \) do

\( TQ \) --- \( \langle n_i, y_i \rangle \) at time \( T_j \);

\( \text{EVL} = 0 \);

for all \( n_i \in PT \) do

if \( s_i \neq y_i \) then

begin (* node state update phase *)

\( s_i = y_i \);

if for some \( \tau^g \) such that \( \text{DRAIN}(\tau^g) = n_i \) and \( n_i \) is input to some \( S_u \) then

APPEND(\( \text{EVL} \), \( S_u \));

if for some \( \tau^g \) such that \( \text{Gate}(\tau^g) = n_i \) then

begin (* transistor state update *)

\( z^g \) --- SET_TRANS(\( TTYPE(\tau^g), s_i \));

end

end

end

- 86 -
if \( n_i \) input to some \( S_u \) then
\[
\text{APPEND(EVL, } S_u)\
\]
else
\[
\text{for some } t_{\overline{q}} \text{ such that } \text{SOURCE}(t_{\overline{q}}) \in S_v \text{ and } \text{DRAIN}(t_{\overline{q}}) \in S_w \text{ do}
\]
\[
\begin{align*}
&\text{begin} \\
&\text{APPEND(EVL, } S_v); \\
&\text{APPEND(EVL, } S_w) \\
&\text{end}
\end{align*}
\]
\end{align*}
end

\begin{align*}
G &\longrightarrow \text{ form an undirected graph with } V = \{ n_i \mid n_i \in \text{EVL} \} \\
\text{and } E = \{ <i, j> \mid \text{for some } t_{\overline{q}} : \text{SOURCE}(t_{\overline{q}}) = n_i, \\
&\text{DRAIN}(t_{\overline{q}}) = n_j \text{ and } Z_{\overline{q}} = \text{'closed' or 'unknown'} \} ;
\end{align*}

\begin{align*}
\text{SUBNETS} &\longrightarrow \text{ PARTITION(G)}; \quad \text{(* subnetwork repartitioning *)}
\end{align*}

\begin{align*}
\text{for all } S_i \in \text{SUBNETS do}
\begin{align*}
&\text{begin} \\
&\text{NODESTATE of } S_i; \quad \text{(* subnetwork evaluation phase *)}
\end{align*}
\end{align*}

\begin{align*}
&\text{Delays of output nodes which have changed} \\
&\text{state in } S_i; \\
&\text{Check for spikes;}
\end{align*}

\begin{align*}
&\text{(* output scheduling phase *)}
\end{align*}

\begin{align*}
&\text{Schedule events with delay delta at } PT + \Delta ; \\
&\text{SUBNETWORKS } \longrightarrow \text{ SUBNETWORKS } U \{S_i\}
\end{align*}

\end{align*}
end

\begin{align*}
\text{while not end of simulation do}
\begin{align*}
&\text{increment } PT
\end{align*}
\end{align*}

\begin{align*}
\text{end}
\end{align*}
procedure APPEND(EVL, S_x);
begin
  if S_x \notin EVL then
    begin
      EVL \leftarrow EVL \cup \{S_x\};
      remove S_x from SUBNETWORKS
    end
  end
Procedure NODESTATE evaluates the node states of the given subnet containing a set of nodes and a set of closed- and unknown-state transistors (transistors with 1's or X's at their gate terminals). At the beginning, all pullup nodes are initialized to 1. Then, we consider the undirected graph containing a vertex for each node in the subnet and an edge for each transistor which is in the "closed" state. The connected components of the graph partition the set of nodes into groups called GROUPS. The state of each of the group is computed by the procedure STATE.

If the subnet contains X-transistors (having X on their gate terminal), then we must look at potential interactions between the groups. The state of an X-transistor is unknown: it may be opened, closed or somewhere in between. Hence, if a node has a unique state regardless of the behaviors of the X-transistors in the subnet, then the node will be set to this state; otherwise, it will be set to X.

Thus far, we have computed the state of each group assuming all X-transistors in the subnet are set "open". The next step is to analyze the groups and set the state of a group to X if some combination of X-transistors could be set to "closed" and result in a different group state. First a supergraph is formed containing a vertex for each group and an edge between a pair of vertices if an X-transistor connects two nodes in the corresponding groups. The connected components of the supergraph is called SUPERGROUP (supergroup contains a set of groups linked by X-transistors).
If a supergroup contains only one element (one group), that means the subnet does not contain X-transistor, and no further analysis is required. Otherwise, the state and strength of the supergroup is computed by the procedure SUPERSTATE. Any group with a state different from the supergroup state must be set to X, because its state would be changed if all X-transistors were set to "closed". All the groups which have an X-state are called "poisoned" groups. Furthermore, a poisoned group can spread the X-state to its neighbor, unless the neighbor is stronger.
Input: A subnet $S_i$.
Output: The new states of subnet $S_i$.

procedure NODESTATE ($S_i$);

begin

for each $n_i$ such that $f_i = "L"$ do $s_i = 1$;

$G \leftarrow$ form an undirected graph with $V = \{n_i\}$ and
$E = \{ <i,j> \mid$ for some $t_q : \text{SOURCE}(t_q) = n_i,$
$\text{DRAIN}(t_q) = n_j$ and $Z_q = 'closed' \};$

GROUPS $\leftarrow$ PARTITION($G$);

for each $G_j \in$ GROUPS do

\begin{align*}
&\text{strength}(G_j), \text{state}(G_j) \leftarrow \text{STATE}(G_j); \\
&SG \leftarrow \text{form an undirected graph with } V = \{G_j\} \text{ and} \\
&E = \{ <i,j> \mid \text{for some } t_q : \text{SOURCE}(t_q) \in G_i \text{ and} \\
&\text{DRAIN}(t_q) \in G_j \text{ and } Z_q = 'unknown' \};
\end{align*}

if $|SG| > 1$ then

begin

\begin{align*}
&Y \leftarrow \text{SUPERSTATE( strength, state, SG )}; \\
&P \leftarrow \{G_j \in SG \mid \text{state}(G_j) \not\in Y \}; \\
&\text{POISON( P, SG, strength )}; \\
&\text{for each } G_j \in P \text{ do state}(G_j) \leftarrow X
\end{align*}

end

for each $G_j \in$ GROUPS do

for each $n_i \in G_j$ do

if $f_i \neq "D"$ and $s_i \neq \text{state}(G_j)$ then

$y_i \leftarrow \text{state}(G_j)$;

for each $n_i \in S_i$ do

if $s_i \neq y_i$ and $\text{NTYPE}(n_i) \not\in 'external'$ then

\end{align*}
\[ s_i \leftarrow y_i \]

end

Procedure STATE computes the state and strength of a given group. In this procedure, the strongest node(s) in the group are inspected, where node strengths are ordered as Driver > Pullup > Weak. The strength of the group is defined as the strength of its strongest node(s). If the states of the strongest nodes are equal, the group state becomes this state; otherwise it becomes X, See Table 4.2.

Procedure SUPERSTATE closely resembles the procedure STATE. The strength of the supergroup is computed as the strength of its strongest group(s). The state of the supergroup is computed as the state of the strongest group(s), if they are equal, and as an X if they are not.
### (a) Group strength

<table>
<thead>
<tr>
<th>node(i)</th>
<th>Driver (D)</th>
<th>Pullup (L)</th>
<th>Weak (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>node(j)</td>
<td>Driver (D)</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>Pullup (L)</td>
<td>D</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Weak (W)</td>
<td>D</td>
<td>L</td>
<td>W</td>
</tr>
</tbody>
</table>

### (b) Group state

Table 4.2
: Group strength and state table.
Input: a group $G_X$.
Output: The strength and state of $G_X$.

procedure STATE($G_X$);
begin
  $k$ <-- null; (* Driver > Pullup > Weak > null *)
  $y$ <-- $x$;
  for each $n_i \in G_X$ do
    if $f_i > k$ then
      begin
      $k$ <-- $f_i$;
      $y$ <-- $s_i$
      end
    else
      if $f_i = k$ and $s_i \neq y$ then
      $y$ <-- $x$
    end
  end
end
Input: A supergroup, strength and state of each group in the supergroup.

Output: State of supergroup.

procedure SUPERSTATE (strength, state, SG);
begin
  \text{K} <-- \text{null}; (* Driver > Pullup > Weak > null *)
  \text{Y} <-- \text{X};
  \text{for each } G_i \in \text{SG do}
  \text{if strength}(G_i) > \text{K then}
    \begin{align*}
      & \text{begin} \\
      & \text{K} <-- \text{strength}(G_i); \\
      & \text{Y} <-- \text{state}(G_i); \\
      & \text{end}
    \end{align*}
  \text{else}
  \text{if strength}(G_i) = \text{K and state}(G_i) \neq \text{Y then}
    \text{Y} <-- \text{X}
  \text{end}
In the procedure POISON, the initial set of poisoned groups $P$, is expanded to the neighboring groups. A neighboring group can be poisoned by these poisoned groups, unless the neighbor is stronger.

Input: An initial set of poisoned groups $P$, a supergroup $SG$, and strength: an array indicating the strength of each group.

Output: $P$ is expanded.

procedure POISON ($P$, $SG$, strength);
     begin
         pstrength <-- strength;
         $B$ <-- $P$;
         while $B 
eq 0$ do
             begin
                 $G_j$ <-- an element of $B$ with maximal strength;
                 $B$ <-- $B - \{G_j\}$;
                 for each $G_k$ connected to $G_j$ by an X-transistor do
                     if pstrength($G_k$) < pstrength($G_j$)
                     or (pstrength($G_k$) = pstrength($G_j$) and $G_k \notin P$) then
                         begin
                             $P$ <-- $P \cup \{G_k\}$;
                             $B$ <-- $B \cup \{G_k\}$;
                             pstrength($G_k$) <-- pstrength($G_j$)
                         end
                 end
             end
         end
     end
Procedure PARTITION is used for partitioning an undirected graph into its connected components. This algorithm is a variation of the depth-first search algorithm of Aho, Hopcroft and Ullman [2].

Input: A graph $G=(V,E)$ where $V$ is a set of vertices, and $E$ is a set of edges.
Output: A set of connected components.

procedure PARTITION(G);
begin
  $j \leftarrow 1$;
  for all $v_i$ in $V$ do mark $v_i$ "new";
  while there exits a vertex $v_i$ in $V$ marked "new" do
    begin
      $S_j \leftarrow \{v_i\}$;
      SEARCH($S_j$, $v_i$);
      $j \leftarrow j + 1$
    end
end

procedure SEARCH($S,v$);
begin
  mark $v$ "old";
  for each vertex $w$ adjacent to $v$ in $G$ do
    if $w$ is marked "new" then
      begin
      \end
end
S <-- S U \{w\};
SEARCH(S,w);
end

The procedure DELAY computes the delay times for each output node, which has changed state, of the subnet \( S_i \) by using the equations which have been derived in previous sections. The procedure first checks whether the future state of the output node is a 1, 0 or X. If the future state is a 1, a direct path is found between the output node and the pullup node or the input node, and the rise delay is computed by using the appropriate rise delay equations. If the future state is a 0, a direct path is found between the output node and the pullup node or input node, an equivalent super-driver-transistor is derived, and the fall delay is computed by using the appropriate fall delay equations. If the future state is an X, a unit delay is assumed because occurrence of an X during simulation usually indicates an error condition for which pessimism dictates the shortest possible delay. In computing delays, capacitances from the side branches are assumed to be connected to the branch nodes (see Figure 4.8). This approach will reduce the tree structure into a chain structure.
Figure 4.8: Side capacitances are assumed to be connected to the branch nodes. In computing delays, for direct path from node 3 to node 7, side capacitance $C_s$ is assumed to be connected to node 4 (total capacitance at node 4 is $C_4 + C_s$).
Input: A subnet $S$
Output: Delays of each output node which has changed state.

procedure DELAY($S$);
begin
for each $n_i \in S$ do
  if $\text{NTYPE}(n_i) = '\text{external}'$ and $s_i \neq y_i$ then
    if $y_i = 1$ then
      begin
        search for a direct path between $n_i$ and
        pullup node or input node;
        compute the rise delay ($\delta$)
      end
    else
      if $y_i = 0$ then
        begin
          substitute the driver transistors with
          an equivalent super-transistor;
          search for a direct path between $n_i$ and
          pullup node or input node;
          compute the fall delay ($\delta$)
        end
      else
        assign a unit delay ($\delta$)
  end
end
Chapter V

IMPLEMENTATION

5.1 Introduction

As we have discussed in earlier chapters, there are a number of basic considerations which dictate the entire functioning of a simulator, as well as the system flexibility. Primary among these considerations is the handling of the logic network data and the basic simulation mechanism. There are two basic methods for digital logic simulation, compiled simulation and table-driven simulation. Compiled simulation could not provide the timing accuracy desired, hence, a table-driven simulation is used in this simulator.

For design verification, detection of timing problems is a major objective; hence, the selection of an appropriate time flow mechanism is of considerable importance. Since the decisions with respect to the table structure and time flow mechanism are considered to be of major importance, the following detailed discussion is provided.

This multiple delay switch-level simulator has been implemented, based on the algorithm described in the previous chapter, on the Amdahl 470/V8 system as a series of Pascal program. Executable version of the program is given in Appendix D.
5.2 **Data Structures**

In a table-driven simulator the circuit is represented by a set of tables indicating the important attributes of each node and transistor, such as:

1. the type of node
2. the state of the node
3. the delay at the node
4. the capacitance at this node
5. the state of the transistor
6. the drain, gate and source of the transistor
7. transistors for which this node is the gate terminal (fanouts)
8. transistors which connect this node to other nodes (interconnects)

Numerous options exist for the implementation of a particular table structure. Each of these table structures has its pros and cons. One particular table structure that is general, effective and efficient is presented here.

The overall table configuration is shown in Figure 5.1. As can be seen from the figure, pointers and linked lists are extensively used. The node table contains all the information about each node and the interconnection pointers for each node. The transistor table contains pointers to the drain, gate and source nodes of each transistor. The fanoutset and interconset tables contain pointers to the respective fanout and interconnection transistors of each
node: fanout transistors are transistors for which this node is the gate node, interconnection transistors are transistors which connect this node to other nodes. The timing table contains the timing information of the node for delay calculation and spike analysis.

**Figure 5.1**: Overall table configuration and interaction
The node table shown in Figure 5.2 contains information about the nodes of a circuit. Each node occupies three words of information. Detailed description of each field of these words is given in Figure 5.3.

<table>
<thead>
<tr>
<th>WORD</th>
<th>INDEX</th>
<th>FLAGS</th>
<th>NODE TYPE</th>
<th>STATE STRING</th>
<th>CAP</th>
<th>VALUE</th>
<th>W/L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>POINTER TO THE TIMING TABLE</td>
<td>NO. OF FANOUTS</td>
<td>NO. OF INTERCONNECTS</td>
<td>node #1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>POINTER TO BEGINNING OF RANLIBSET</td>
<td>POINTER TO BEGINNING OF INTERCONNECT</td>
<td>node #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$n = \text{number of nodes}$

Figure 5.2: Node table
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>indicates the node number</td>
<td>argument in vector arrays to address the piece of information desired</td>
</tr>
<tr>
<td>flags</td>
<td>indicators</td>
<td>checking of certain condition</td>
</tr>
<tr>
<td>node type</td>
<td>type of node</td>
<td>state evaluation</td>
</tr>
<tr>
<td>state</td>
<td>node state</td>
<td>state evaluation</td>
</tr>
<tr>
<td>strength</td>
<td>node strength</td>
<td>strength evaluation</td>
</tr>
<tr>
<td>cap-value</td>
<td>node capacitance</td>
<td>delay calculation</td>
</tr>
<tr>
<td>W/L</td>
<td>width and length of the load transistor; for pullup node only</td>
<td>delay calculation</td>
</tr>
<tr>
<td>pointer to the timing table</td>
<td>pointer points to the timing table</td>
<td>delay calculation and spike detection</td>
</tr>
<tr>
<td>no. of fanouts</td>
<td>number of fanout transistors</td>
<td>event scheduling</td>
</tr>
<tr>
<td>no. of interconnect</td>
<td>number of interconnect transistor</td>
<td>state evaluation</td>
</tr>
<tr>
<td>pointer to beginning of fanoutset</td>
<td>pointer points to the beginning of the fanoutset</td>
<td>event scheduling</td>
</tr>
<tr>
<td>pointer to beginning of interconset</td>
<td>pointer points to the beginning of the interconset</td>
<td>state evaluation</td>
</tr>
</tbody>
</table>

*Figure 5.3: Description of node table*
The transistor table shown in Figure 5.4 contains information about the transistors of a circuit. Each transistor occupies three words of information. Detailed description of each field of these words is given in Figure 5.5.

<table>
<thead>
<tr>
<th>Word</th>
<th>INDEX</th>
<th>FLAGS</th>
<th>STATE</th>
<th>TRANSISTOR TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>Transistor #1</td>
</tr>
<tr>
<td>2</td>
<td>Pointer to Gate Node</td>
<td>Pointer to Drain Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Pointer to Source Node</td>
<td>W/L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>Transistor #2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transistor #3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transistor #m</td>
</tr>
</tbody>
</table>

m = the number of transistors

Figure 5.4: Transistor table
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>indicates the transistor number</td>
<td>argument in vector arrays to address the piece of information desired</td>
</tr>
<tr>
<td>flags</td>
<td>indicators</td>
<td>checking of certain condition</td>
</tr>
<tr>
<td>state</td>
<td>transistor state</td>
<td>state evaluation</td>
</tr>
<tr>
<td>transistor type</td>
<td>type of transistor</td>
<td>state evaluation</td>
</tr>
<tr>
<td>pointer to gate node</td>
<td>pointer points to the gate terminal</td>
<td>state evaluation</td>
</tr>
<tr>
<td>pointer to drain node</td>
<td>pointer points to the drain terminal</td>
<td>state evaluation</td>
</tr>
<tr>
<td>pointer to source node</td>
<td>pointer points to the source terminal</td>
<td>state evaluation</td>
</tr>
<tr>
<td>W/L</td>
<td>width and length of the transistor channel</td>
<td>delay calculation</td>
</tr>
</tbody>
</table>

Figure 5.5: Description of transistor table
The fanoutset table and interconset table, shown in Figure 5.6 and Figure 5.7 respectively, are closely associated with the node table. As we have seen previously, two of the entries in the node table are pointers to the fanoutset and interconset tables, which in turn contains pointers to the transistor table entries of interest.

**Figure 5.6**: Fanoutset table

**Figure 5.7**: Interconset table
The timing table, as shown in Figure 5.8, stores the timing information of each external (input and output) node. These timing information can be used to calculate the delays and checking of spikes. The first word contains the rise time and rise delay of the node. The second word contains the fall time and the fall delay of the node. The third word stores the times and the future states of the node, or zero if no future state pending for this node.

\[
\begin{array}{c|c}
\text{T Imi ng Ta ble} \\
\hline
\hline
\text{T i mes} & \text{F u t u re S t a tes} \\
\hline
\end{array}
\]

*Figure 5.8*: Timing table
5.3 Time Flow Mechanism

Figure 5.9 shows the time flow mechanism used in this multiple delay simulator. The TQ (Time Queue) array is the basic element of storage in the time flow mechanism. Each row represents a discrete point in the simulation time base. Time in the simulator can move forward only by a fixed increment. Each entry in the TQ array contains pointers to a list of events which are to occur at that instant in time.

Figure 5.9: Simulation time flow mechanism
There is one storage area, called the macro time event list (MTEL), used to contain events that are to occur at points in time which are beyond the range of the current TQ array time frame. Consider the TQ array to be indexed from 1 to 2*Z. Let PT be a variable denotes the "present time" in the TQ array being processed and MTC be a variable starting at zero which denotes the number of cycles which have been made through the TQ array. Then, the simulation time is equal to the expression (Z*MTC)+PT-1 and the time frame of the TQ array is from (MTC*Z) to (MTC*Z)+(2*Z)-1 inclusive. The MTEL is ordered on the time of occurrence of the events.

At entry Z+1 of the TQ array, an event known as the MTEL update is always scheduled to occur as the first event at that point in time. The action of this event is as follows:

1. Entries Z+1 through 2*Z replace entries 1 through Z in the TQ array and entries Z+1 through 2*Z are zeroed out.
2. MTC is incremented by one.
3. PT is set to 1.
4. The MTEL update event is scheduled at Z+1.
5. The MTEL is searched to determine if any events scheduled to occur between (MTC*Z)+Z+1 and (MTC*Z)+(2*Z) inclusive. If such events, they are removed from the MTEL and placed in lists pointed to by the pointers on the proper entry in the TQ array.
The scheduling of an event is done by placing the event on the proper list. If the event is to occur within the time frame of the TQ array, then it is put on a list pointed to by a TQ array entry. If the event is to occur beyond the current time frame of the TQ array, then it is inserted in the MTEL in a manner so as to retain the ordering of the MTEL.

5.4 Examples

The algorithms described in the previous chapter have been implemented on the Amdahl 470/V8 system as a series of Pascal program, and a program listing is given in Appendix D. A number of NMOS circuits have been simulated by this program. A few examples are given below.

Example 5

The following circuit, with parallel and series driver transistors, is simulated. Node (2) and (3) are the input nodes. Node (10) is the output node. The output waveform at node (10) produced by this simulator is shown in solid line in Figure 5.10. For comparison purposes the waveform in dotted line is the ones produced by SPICE simulation program.
Figure 5.10: Input and output waveforms for Example 5
Example 6

Example 6 is an inverter pair coupled by a pass transistor, as shown below. Node (1) is the input node and node (6) is the output node. The output waveform produced by this simulator is shown in solid line in Figure 5.11. For comparison purposes the waveform in dotted line is the ones produced by SPICE simulation program. This example is presented to illustrate the ability of this simulator to handle error condition (X-state).
Figure 5.11: Input and output waveforms for Example 6
Chapter VI

CONCLUSIONS

In this thesis, a new multiple delay switch-level simulator for MOS LSI circuits has been presented which includes the following features: table-driven, time queue with macro time event list for simulation with multiple rise and fall delays, selective tracing of active paths, and event scheduling and rescheduling on the time queue including spike detection.

The delay model used in this simulator is organized around a subnetwork, instead of logic gates. The use of subnetworks permits both logic gates and pass transistors to be handled in a uniform fashion. The delay model takes into account the effects of not only output loading capacitance but also the input waveform. The delay model presented here also helps in removing most of the limitations mentioned in [16]: inability to handle "multiple input to output delays", "delay of a gate driving the data input of a pass transistor", "slope of the input waveform", and "overlapping transitions".

An improved algorithm is described for computing the node states and delays of a subnetwork. The algorithm is based on scheduling of subnetworks which are locally evaluated based on local relaxation method.
Some MOS circuit designs depend on the relative sizes of several capacitances for their logical behavior. This is seen frequently with pre-charge buses, in which a charge is first placed on a bus, and then this bus drives its signal through a pass transistor onto a dynamic RAM cell. In this algorithm, the nodes will be set to X (unless they were previously in the same state), because the simulator does not know the relative capacitance values. This shortcoming can be circumvented by allowing different kinds of storage nodes, e.g., "Strong" and "Weak" nodes. The previously described algorithm can be extended by ranking the node strengths as: Driver > Pullup > Strong > Weak. There are other limitations of this multiple delay simulator. First, the use of X state for representing both the unknown or initial state and an error state. For more accurate design verification, the above two cases should be distinguished by having X for initially unknown state and E for an error state. Second, the simulator receives information about transistors and parasitic capacitances of nodes, but not resistances of interconnects. Thus, propagation delay in long wires is ignored.

In this multiple delay simulator, most of the limitations mentioned in [16] were addressed. It will be interesting to study the performance of this proposed simulator. The following areas for future work are prompted from this thesis:
1. The use of more logic values for more accurate logic verification.

2. The provision of more accurate delay time estimation for unknown state.

3. The inclusion of wiring delays in delay time calculation.
REFERENCES


- 119 -


Appendix A

DERIVATION OF RISE TIME AND RISE DELAY OF AN NMOS INVERTER

\[ V_{ds} = \frac{U_n C_o L_1}{2} (V_{gs} - V_{TD})^2 \]

\[ = \frac{U_n C_o L_1}{2} (-V_{TD})^2 \]

\[ I_c = C_L \frac{dV_o}{dt} \]

\[ C_L \frac{dV_o}{dt} = \frac{U_n C_o L_1}{2} (-V_{TD})^2 \]

The time in the saturation region,

\[ t_{sat} = \int_{X_1}^{V_{DD} + V_{TD} - X_1} \frac{2 C_L}{V_{gs} - V_{TD}} \cdot dV_o \]

\[ = \frac{2 C_L}{U_n C_o (L_1/L_1) (-V_{TD})^2} \left( V_{DD} + V_{TD} - X_1 \right) \]

- 122 -
When \( V_{CS} = OV > V_{TD}, V_{DS} = V_{DD} - V_o \leq -V_{TD} \), the load transistor is in linear region,

\[
I_L = \frac{U \cdot C}{2} \cdot \frac{V_L}{L_1} \left[ 2(V_{CS} - V_{TD})V_{DS} - V_{DS}^2 \right]
\]

\[
= \frac{U \cdot C}{2} \cdot \frac{V_L}{L_1} \left[ 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]
\]

\[
I_c = C_L \frac{dV_o}{dt}
\]

\[
\frac{dV_o}{dt} = \frac{U \cdot C}{2} \cdot \frac{V_L}{L_1} \left[ 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]
\]

The time in the linear region,

\[
t_{lin} = \int_{V_{DD} - V_o}^{V_{DD} - (V_{DD} - V_o)} \frac{dV_o}{\frac{U \cdot C}{2} \cdot \frac{V_L}{L_1} \left[ 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]}
\]

\[
= \frac{C_L}{U \cdot C \cdot (-V_{TD}) \cdot \frac{V_L}{L_1}} \int_{V_{DD} - V_o}^{V_{DD} - (V_{DD} - V_o)} \frac{1}{(-V_{TD}) - (V_{DD} - V_o)} \cdot dV_o
\]

\[
= \frac{C_L}{U \cdot C \cdot (-V_{TD}) \cdot \frac{V_L}{L_1}} \left[ \ln \left( \frac{2(-(V_{DD} - V_o))}{(V_{DD} - V_o)} \right) \right]_{V_{DD} - V_o}^{V_{DD} - (V_{DD} - V_o)}
\]

\[
= \frac{C_L}{U \cdot C \cdot (-V_{TD}) \cdot \frac{V_L}{L_1}} \ln \left( \frac{2(V_{DD} - (V_{DD} - V_o))}{(V_{DD} - V_o)} \right)
\]

- 123 -
The total rise time $t_r$ is

$$t_r = t_{sat} + t_{lin}$$

$$= \frac{2 \ C_L}{U_n \ C_o \ V_{DD} \ \frac{V_1}{L_1}} \left[ \frac{V_{DD}}{(-V_{TD})^2} \cdot \frac{(V_{DD} + V_{TD} - X_1)}{(-V_{TD})^2 \cdot \ln(\frac{2(-V_{TD}) - (V_{DD} - X_2)}{(-V_{TD}) - (V_{DD} - X_2)})} \right] + \frac{V_{DD}/2 \cdot \ln(2(-V_{TD}) - (V_{DD} - X_2))}{(-V_{TD}) - (V_{DD} - X_2)}$$

**Rise Delay**

When the load transistor is in saturation region,

$$t_{pl} = \int_{V_2}^{V_D} \frac{2 \ C_L}{U_n \ C_o \ (-V_{TD})^2 \ \frac{V_1}{L_1}} \cdot dV_o$$

$$= \frac{2 \ C_L}{U_n \ C_o \ (-V_{TD})^2 \ \frac{V_1}{L_1}} \cdot (V_{DD} + V_{TD} - \frac{V_1}{L_1})$$
When the load transistor is in linear region,

\[
\begin{align*}
\tau_{p2} &= \int \frac{2C_L}{U_nC_o V_{DD} W/L_1} \frac{dV_o}{2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2} \left( \ln \left( \frac{2(-V_{TD}) - (V_{DD} - V_o)}{(V_{DD} - V_o)} \right) \right)^{1.5} \frac{V_{DD}}{V_{DD} + V_{TD}} \\
&= \frac{C_L}{U_n C_o (-V_{TD}) W/L_1} \ln \left( \frac{2(-V_{TD}) + (V_{DD} - X.5)}{(V_{DD} - X.5)} \right)^{1.5} \frac{V_{DD}}{V_{DD} + V_{TD}}
\end{align*}
\]

The total rise delay \( \tau_{PHH} \) is

\[
\tau_{PHH} = \tau_p + \tau_{p2}
\]

\[
\begin{align*}
\tau_{PHH} &= \frac{2C_L}{U_n C_o V_{DD} W/L_1} \left[ \frac{V_{DD}}{(2(-V_{TD}))^2 (V_{DD} + V_{TD} - V_L)} \\
&+ \frac{V_{DD}}{V_{DD}/2} \ln \left( \frac{2(-V_{TD}) - (V_{DD} - X.5)}{(V_{DD} - X.5)} \right) \right]
\end{align*}
\]
Appendix B

DERIVATION OF FALL TIME AND FALL DELAY OF AN
NMOS INVERTER

Fall Time

When \( V_{DD} > V_o > V_i - V_{TE} \), load transistor is in linear region and driver transistor is in saturation region.

\[
I_L = \frac{U_{n^*o} - N_L}{2} \frac{W_1}{L_1} \left[ 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]
\]

\[
I_D = -\frac{U_{n^*o} - N_L}{2} \frac{W_d}{L_d} (V_i - V_{TE})^2
\]

\[
I_c = C_L \frac{dV_o}{dt}
\]

\[
C_L \frac{dV_o}{dt} = -\frac{U_{n^*o} - N_L}{2} \frac{W_1}{L_1} \left[ (V_{DD} - V_o)^2 - 2(-V_{TD})(V_{DD} - V_o) + k(V_i - V_{TE})^2 \right]
\]

\[
t_{ft} = \frac{2 C_L}{U_{n^*o} - N_L} \int_{V_{TE}}^{V_i-V_{TE}} \frac{dV_o}{(V_{DD} - V_o)^2 - 2(-V_{TD})(V_{DD} - V_o) + k(V_i - V_{TE})^2}
\]
let $V = V_{DD} - V_o$, $dV = -dV_o$

$$t_{r1} = \frac{2 C_L}{u_{n} C_{o} \lambda / L_1} \int_{V_{DD} - X_2}^{V_{DD} - (V_i - V_{TB})} \frac{dV}{V^2 - 2(-V_{TB})V + k(V_i - V_{TE})^2}$$

for $a = 1$, $b = 2V_{TD}$, $c = k(V_i - V_{TE})^2$

$$\int \frac{dx}{ax^2 + bx + c} = \frac{2}{\sqrt{4ac - b^2}} \cdot \tan^{-1}\left(\frac{2ax + b}{\sqrt{4ac - b^2}}\right)$$

$$= \frac{2}{\sqrt{4a(V_i - V_{TB})^2 - 4V_{TD}^2}}$$

$$= \frac{1}{\sqrt{k(V_i - V_{TB})^2 - V_{TD}^2}}$$

$$t_{r1} = \frac{2 C_L}{u_{n} C_{o} \lambda / L_1} \cdot \frac{2}{\sqrt{4ac - b^2}} \cdot \left[\tan^{-1}\left(\frac{2aV + 2V_{TD}}{\sqrt{4ac - b^2}}\right)\right]_{V_{DD} - (V_i - V_{TB})}^{V_{DD} - X_2}$$

$$= \frac{V_{DD}}{\sqrt{k(V_i - V_{TB})^2 - V_{TD}^2}} \cdot \left[\tan^{-1}\left(\frac{V_{DD} - (V_i - V_{TB}) - V_{TD}}{\sqrt{k(V_i - V_{TB})^2 - V_{TD}^2}}\right) - \tan^{-1}\left(\frac{V_{DD} - X_2 + V_{TD}}{\sqrt{k(V_i - V_{TB})^2 - V_{TD}^2}}\right)\right] \cdot \frac{2 C_L}{u_{n} C_{o} \lambda / L_1}$$
When \( V_i - V_{TB} \geq V_o \geq V_{DD} - V_{TD} \), both transistors are in linear region,

\[
I_L = \frac{U_n C_o}{2} \frac{W_1}{L_1} \left[ 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]
\]

\[
I_D = \frac{U_n C_o}{2} \frac{W_d}{L_d} \left[ 2(V_i - V_{TE})V_o - V_o^2 \right]
\]

\[
I_c = C_L \frac{dV_o}{dt}
\]

\[
C_L \frac{dV_o}{dt} = \frac{U_n C_o}{2} \frac{W_1}{L_1} \left[ (k - 1)V_o^2 + (2V_{TD} + 2V_{DD} - 2k(V_i - V_{TE}))V_o^2 \right.
\]

\[
+ \left. (-2V_{TD}V_{DD} - V_o^2) \right]
\]

\[
t_f = \left( \frac{\int \frac{dV_o}{V_i - V_{TE}}}{(k-1)V_o^3 + (2V_{TD} + 2V_{DD} - 2k(V_i - V_{TE}))V_o^2 + (-2V_{TD}V_{DD} - V_o^2)} \right)^{\frac{1}{2}}
\]

\[
= \frac{2 C_L}{U_n C_o W_1/L_1}
\]

For \( a = (k - 1), \ b = 2V_{TD} + 2V_{DD} - 2k(V_i - V_{TE}), \ c = -2V_{TD}V_{DD} - V_o^2 \)

\[
\int \frac{dx}{ax^2 + bx + c} = \frac{1}{\left[ b^2 - 4ac \right]^{1/2}} \ln \left( \frac{2ax + b - \left[ b^2 - 4ac \right]^{1/2}}{2ax + b + \left[ b^2 - 4ac \right]^{1/2}} \right)
\]

\[
\frac{1}{\left[ b^2 - 4ac \right]^{1/2}} = \frac{1}{[4(V_{TD} + V_{DD} - k(V_i - V_{TE}))^2 + 4(k-1)(2V_{TD}V_{DD} + V_o^2)]^{1/2}}
\]

-128-
\[ t_{r2} = \frac{2 C_L}{U_n C_o \frac{W_o}{L_L} \left[ b^2 - 4ac \right]^{1/2}} \left[ \ln \frac{2aV_o + b - b^2 - 4ac}{2aV_o + b + b^2 - 4ac} \frac{1}{2} \right] \]

\[ = \frac{V_{DD} \ln \frac{2a(V_{DD} - V_{TD}) + b - \left[ b^2 - 4ac \right]^{1/2}}{2a(V_{DD} - V_{TD}) + b + \left[ b^2 - 4ac \right]^{1/2}} - \ln \frac{2a(V_i - V_{TH}) + b - \left[ b^2 - 4ac \right]^{1/2}}{2a(V_i - V_{TH}) + b + \left[ b^2 - 4ac \right]^{1/2}}}{U_n C_o V_{DD} \frac{W}{L_L}} \]

When \( V_{DD} + V_{TD} \geq V_o \geq X_1 \), load transistor is in saturation region and driver transistor is in linear region.

\[ I_L = \frac{U_n C_o V}{2 L_L} (-V_{TD})^2 \]

\[ I_D = \frac{U_n C_o V d}{2 L_d} \left[ 2(V_i - V_{TH})V_o - V_o^2 \right] \]

\[ I_c = C_L \frac{dV_o}{dt} \]

\[ \frac{dV_o}{C_L dt} = \frac{U_n C_o V}{2 L_L} \left[ kV_o^2 - 2k(V_i - V_{TH})V_o + (-V_{TD})^2 \right] \]

\[ t_{r3} = \frac{2 C_L}{U_n C_o \frac{W_o}{L_L}} \int_{V_{LD} + V_{TD}}^{X_1} \frac{dV_o}{kV_o^2 - 2k(V_i - V_{TH})V_o + (-V_{TD})^2} - 129 - \]
for \( a = k, \ b = -2k(v_i - V_{TB}), \ c = (-V_{TD})^2 \)

\[
t_{f3} = \frac{V_{DD}}{[b^2 - 4ac]^{1/2}} \cdot \left[ \ln \left| \frac{2ax_1 + b - \left[ b^2 - 4ac \right]^{1/2}}{2ax_1 + b + \left[ b^2 - 4ac \right]^{1/2}} \right| \right] - \ln \left| \frac{2a(v_{DD} + V_{TD}) + b - \left[ b^2 - 4ac \right]^{1/2}}{2a(v_{DD} + V_{TD}) + b + \left[ b^2 - 4ac \right]^{1/2}} \right| \cdot \frac{2c_L}{\frac{V_{DD}}{I_1/L_1}}
\]

The total fall time,

\[
t_f = t_{f1} + t_{f2} + t_{f3}
\]

**Fall Delay.**

When load transistor is in linear region and driver transistor is in saturation region,

\[
t_{pl} = \int_{v_{DD}}^{v_i - V_{TB}} \frac{\frac{2}{2c_L} \frac{dV_o}{\frac{V_{DD} - V_o}{2} - 2(-V_{TD})(v_{DD} - V_o) + k(v_i - V_{TB})^2}}{v_{DD} - V_{DD}}
\]

\[
= \frac{V_{DD}}{[k(v_i - V_{TB})^2 - V_{TD}^2]^{1/2}} \cdot \left[ \tan^{-1} \left( \frac{V_{DD} - (v_i - V_{TB}) + V_{TD}}{\frac{k(v_i - V_{TB})^2 - V_{TD}^2}{1/2}} \right) \right] - \tan^{-1} \left( \frac{V_{TD}}{\frac{k(v_i - V_{TB})^2 - V_{TD}^2}{1/2}} \right) \cdot \frac{2c_L}{\frac{V_{DD}}{I_1/L_1}}
\]

- 130 -
When both transistors are in linear region,

\[ t_{p2} = \left( \int_{V_{i} - V_{TE}}^{X.5} \frac{dV_o}{(k-1)V_o^2 + (2V_{TB} + 2V_{DD} - 2k(V_i - V_{TE}))V_o + (-2V_{TB}V_{DD} - V_{DD}^2)} \right) \cdot \frac{2 \cdot C_L}{U_{n} \lambda \frac{W}{L_1}} \]

for \( a = (k - 1) \), \( b = 2V_{TB} + 2V_{DD} - 2k(V_i - V_{TE}) \), \( c = -2V_{TB}V_{DD} - V_{DD}^2 \)

\[ t_{p2} = - \ln \frac{2a(V_i - V_{TE}) + b - [b^2 - 4ac]^{1/2}}{2ac(V_i - V_{TE}) + b + [b^2 - 4ac]^{1/2}} \cdot \frac{2 \cdot C_L}{U_{n} \lambda \frac{W}{L_1}} \]

The total fall delay,

\[ t_{pfL} = t_{p1} + t_{p2} \]
Appendix C

DERIVATION OF TRANSITION AND DELAY TIMES OF A
PASS TRANSISTOR

\[ V_C \]
\[ = V_L \]
\[ \frac{V_L}{L_{p}} \]
\[ \frac{I_C}{C_L} \]

Rise Time

Transistor \( T_1 \) is in saturation region.

\[ I_{DS} = \frac{U_n C_o V_D}{2} \left( V_{DD} - V_o - V_{TP} \right)^2 \]

\[ I_{DS} = \frac{U_n C_o V_D}{2} \left( V_D - V_o \right)^2 \]

\[ I_C = C_L \frac{dV_o}{dt} \]

\[ C_L \frac{dV_o}{dt} = \frac{U_n C_o V_D}{2} \left( V_D - V_o \right)^2 \]

\[ t_r = \int_{X_1}^{X_2} \frac{2 C_L}{U_n C_o V_D / L_p} \left( \frac{1}{V_D - V_o} \right)^2 \]

\[ = \frac{2 C_L}{U_n C_o V_D / L_p} \left[ \frac{1}{V_D - V_o} \right]^{X_2}_{X_1} \]
\[ t_r = \frac{2 c_L}{V_{n o p} / L_p} \left( \frac{1}{V_f - X_2} - \frac{1}{V_f - X_1} \right) \]

**Rise Delay**

\[ t_{p LL!} = \int_{V_L}^{X_{+}^{5}} \frac{2 c_L}{V_{n o p} / L_p} \frac{dV_o}{(V_f - V_o)^2} \]

\[ = \frac{2 c_L}{V_{n o p} / L_p} \left( \frac{1}{V_f - X_{+}^{5}} - \frac{1}{V_f - V_L} \right) \]

- 135 -
Fall Time

Transistor $T_1$ is in linear region.

$$I_{DS} = \frac{U_{n0} W}{2 L_p} \left[ 2(V_{DD} - V_{TP})V_o - V_o^2 \right]$$

$$= \frac{U_{n0} W}{2 L_p} \cdot (2V_f V_o - V_o^2)$$

$$I_c = C_L \frac{dV_o}{dt}$$

$$L_p \frac{dV_o}{dt} = \frac{U_{n0} W}{2 L_p} \cdot (V_o^2 - 2V_f V_o)$$

$$t_f = \int_{X_2}^{X_1} \frac{2 C_L \cdot \frac{dV_o}{dt}}{U_{n0} W / L_p \cdot (V_o^2 - 2V_f V_o)}$$

$$= \frac{2 C_L}{U_{n0} W / L_p} \cdot \frac{1}{2V_f} \int_{X_2}^{X_1} \left( \frac{1}{V_o} + \frac{1}{2V_f - V_o} \right) \cdot dV_o$$

$$= \frac{C_L}{U_{n0} W / L_p} \left[ \ln \left( \frac{2V_f - V_o}{V_o} \right) \right]_{X_2}^{X_1}$$

$$= 134$$
\[ t_f = \frac{C_L}{U_n C V_f W_p L_{p1}} \left[ \ln \left( \frac{2V_f - X_1}{X_1} \right) - \ln \left( \frac{2V_f - X_2}{X_2} \right) \right] \]

**Fall Delay**

\[ t_{phL} = \int_{V_{II}}^{X} \frac{2C_L}{U_n C V_f W_p L_p} \cdot \frac{dV_o}{(V_o^2 - 2V_f V_o)} \]

\[ = \frac{C_L}{U_n C V_f W_p L_p} \left[ \ln \left( \frac{2V_f - X_{.5}}{X_{.5}} \right) - \ln \left( \frac{2V_f - V_{II}}{V_{II}} \right) \right] \]
PROGRAM simulation (input, output);

CONST
  nodemax = 100;
  transmax = 100;
  Z = 50;
  Z2 = 100;
  UC = 2.0E-5;
  VDD = 5;
  VTE = 1;
  VTD = -4;

TYPE
  nodenumber = 1..nodemax;
  transnumber = 1..transmax;
  statevalue = 0..2;
  strengvalue = 1..3;
  ntype = 1..2;
  ttype = 1..3;

inputlink = @inputnode;
inputnode =
  RECORD
    fptr : inputlink;
    node : integer;
  END;

pathlink = @pathnode;
pathnode =
  RECORD
    fptr : pathlink;
    node : integer;
    no : integer;
  END;

tilink = @timing;
timing =
  RECORD
    fptr, bptr : tilink;
    time : integer;
    newstate : statevalue;
    tsi : real
  END;

trlink = @transint;
transint =
  RECORD
    fptr : trlink;
    transf : integer
  END;
glink = @gate;
gate =
  RECORD
    node : integer;
  END;
slink = @source;
source =
RECORD
  node : integer
END;
dlink = @drain;

RECORD
  node : integer
END;

hlink = @headernode;

RECORD
  fpotr : hlink;
  node : integer
END;

plink = @plist;

RECORD
  fpotr : plink;
  gr : integer
END;
evlink = @evinfo;
evinfo =
RECORD
  fpotr : evlink;
  node : integer;
  newsta : statevalue;
  time : integer;
  tt : real
END;

overlink = @overflow;

RECORD
  fpotr, bptr : overlink;
  node : integer;
  newsta : statevalue;
  time : integer;
  tt : real
END;
pendlink = @pend;

RECORD
  fpotr : pendlink;
  group : integer;
END;

graph = ARRAY[0..nodemax] OF hlink;
gte = ARRAY[0..10] OF statevalue;
gth = ARRAY[0..10] OF strengvalue;
list = ARRAY[1..Z2] OF evlink;
pathline = ARRAY[1..10].OF pathlink;
A SCAL 8000/2.0B

AAEC (17FEB82)

117 06A8 -- nodeinfo =
118 06A8 -- RECORD
119 06A8 -- tiptr : tlink;
120 06A8 -- fcptr,icptr : trlink;
121 06A8 -- trigger,visit,inev : boolean;
122 06A8 -- state,newsta : statevalue;
123 06A8 -- strength : strenvalue;
124 06A8 -- nodetype : ntype;
125 06A8 -- capvalue : real;
126 06A8 -- fc,ic : integer;
127 06A8 -- W,L : real;
128 06A8 -- END;
129 06A8 --
130 06A8 --
131 06A8 -- transinfo =
132 06A8 -- RECORD
133 06A8 -- gpctr : glink;
134 06A8 -- dpctr : dlink;
135 06A8 -- sptr : slink;
136 06A8 -- trigger : boolean;
137 06A8 -- state : statevalue;
138 06A8 -- transtype : ttype;
139 06A8 -- W,L : real
140 06A8 -- END;
141 06A8 --
142 06A8 -- VAR
143 06A8 -- node : ARRAY[nodenumber] OF nodeinfo;
144 25E8 -- trans : ARRAY[transnumber] OF transinfo;
145 35B8 -- network,subnet,sub,group,supergroup,h : graph;
146 3F00 -- n,m : integer;
147 3F08 -- TQ : list;
148 4098 -- inpu,inp : inputlink;
149 40A0 -- tim,pi : tlink;
150 40A8 -- event : evlink;
151 40AC -- MTEL,ev,newev : overlink;
152 40B8 -- evpend : pendlink;
153 40BC -- EVL,point : hlink;
154 40C4 -- fp,l : trlink;
155 40CC -- ST,MTC,evnum : integer;
156 40DB -- t,tm : integer;
157 40E0 -- tt : real;
158 40E8 -- i,j,k,v,w : integer;
159 40FC -- sta : statevalue;
160 4100 -- str : strenvalue;
161 4104 -- unit : boolean;
162 4108 -- pullup : boolean;
163 410C -- q,ii,pn : integer;
164 4118 -- path : pathline;
165 4140 --
166 4140 -- A PROCEDURE search ( VAR first : hlink;
167 0040 -- i : integer;
168 0040 -- X0,X1,XX : statevalue );
169 0054 --
170 0054 -- (* search for an adjacent node *)
171 0054 --
172 0054 -- VAR
173 0054 -- l : trlink ;
base : hlink;
j, k : integer;

node[i].visit := true;
BEGIN
  l := node[i].icptr;
  IF l <> NIL THEN
    WHILE l <> NIL DO
      BEGIN
        IF (trans[j].state = X0) OR (trans[j].state = X1) AND
            (node[trans[j].dptr@.node].visit = false) THEN
          BEGIN
            k := trans[j].dptr@.node;
            new(base);
            base@.fptr := first;
            base@.node := k;
            first := base;
            search(first, k, X0, X1, XX)
          END;
        ELSE IF (trans[j].sptr@.node <> i) AND
            (node[trans[j].sptr@.node].visit = false) THEN
          BEGIN
            k := trans[j].sptr@.node;
            new(base);
            base@.fptr := first;
            base@.node := k;
            first := base;
            search(first, k, X0, X1, XX)
          END;
      END;
    END;
  END;
END;

PROCEDURE partition (VAR header : graph;
    index : integer;
    x0, x1, xx : statevalue);

VAR
  start, delete, hd : hlink;
  i : integer;

BEGIN
  node[header@.node].visit := false;
  start := header@.fptr
END;

- 140 -
start := header[index] ;
m := 0 ;
WHILE start <> NIL DO
  BEGIN
    IF node[start@.node].visit = false THEN
      BEGIN
        m := m+1 ;
        new(hd) ;
        hd@.fptr := NIL ;
        hd@.node := start@.node ;
        i := start@.node ;
        search(hd,i,X0,X1,XX) ;
        header[m] := hd ;
      END ;
    END ;
  start := start@.fptr
END ;

VAR
  event : evlink ;
  ev,newev : overlinc ;
  j : integer ;
  TQ[ ] : array [ ] of event ;

BEGIN
  j := tocurr - ( Z * MTC - 1 ) ;
  IF j <= ZZ THEN
    BEGIN
      new(event) ;
      WITH event@ DO
        BEGIN
          fpotr := TQ[j] ;
          node := x ;
          newsta := nesta ;
          time := tocurr ;
          tt := tsi
        END ;
      END ;
  ELSE
    BEGIN
      new(newev) ;
      WITH newev@ DO
        BEGIN
          node := x ;
          newsta := nesta ;
          time := tocurr ;
          tt := tsi
        END ;
      END ;
END ;
UNTIL newev@.time >= ev@.time;
    newev@.fptr := ev@.fptr;
    newev@.bptr := ev;
    ev@.fptr@.bptr := newev;
    ev@.fptr := newev
END;

PROCEDURE spike (i: integer;
    nesta: statevalue;
    toc: integer;
    ts: real);
(* checking for spike *)
VAR
    pi, tim: tilink;
    pointer: evlink;
    ev: overl ink;
    stop: boolean;
    j: integer;
BEGIN
    new(tim);
    WITH time DO
        BEGIN
            time := toc;
            newstate := nesta;
            tsi := ts;
        END;
    stop := false;
    pi := node[i].tiptr;
    REPEAT
        pi := pi@.bptr;
        IF (tim@.time <= pi@.time) AND (pi <> pi@.fptr) THEN
            BEGIN
                IF tim@.newstate <> pi@.newstate THEN
                    BEGIN
                        pi@.newstate := tim@.newstate;
                        tim@.newstate := 2;
                        j := pi@.time - (Z * MTC - 1);
                        IF j <= Z2 THEN
                            BEGIN
                                pointer := TQ[j];
                                WHILE pointer <> NIL DO
                                    BEGIN
                                        IF pointer@.node = i THEN
                                            BEGIN
                                                pointer@.newsta := pi@.newstate
                                                pointer@ := NIL
                                            END
                                        ELSE
                                            BEGIN
                                                pointer := pointer@.fptr
                                            END
                                    END
                                END
                            END
                        END
                    END
                END
            END
        END
    END
END.
REPEAT
  ev := ev@.bptr
  UNTIL ev@.node = i ;
  ev@.newsta := pi@.newstate
END

END

ELSE
BEGIN
  tim@.fptr := pi@.fptr ;
  tim@.bptr := pi ;
  pi@.fptr@.bptr := tim ;
  pi@.fptr := tim ;
  appendev(i, tim@.newstate, toc, ts) ;
  stop := true
END

UNTIL stop = true

A PROCEDURE readparameter ;

(* read simulation parameters *)

VAR
  pi : tlink ;
  base, p : trlink ;
  q : glink ;
  d : dlink ;
  s : slink ;
  numnode, numtrans : integer ;
  tc, ti, i, t, num : integer ;
  sta : statevalue ;
  str : strengvalue ;
  nty : ntype ;
  tty : ttype ;
  wid, len, cap : real ;

BEGIN
  network[0] := NIL ;
  readln(numnode, numtrans) ;
  FOR num := 1 TO numnode DO
  BEGIN
    readln(i, sta, str, nty, cap, tc, ti, wid, len) ;
    WITH node[i] DO
    BEGIN
      new(h[0]) ;
      h[0]@.node := i ;
      h[0]@.fptr := network[0] ;
      network[0] := h[0] ;
      appendev(i, sta, 0, 0) ;
      state := 2 ;
      newsta := 2 ;
      strength := str ;
      nodedtype := nty ;
      capvalue := cap ;
      fc := tc ;
      ic := ti ;
      W := wid ;
ASCAL 8000/2.0B AAEC (17FEB82)

407 0250 --
408 0262 --
409 0268 --
410 0276 --
411 0292 --
412 02AE --
413 02BE --
414 02D0 --
415 02F0 --
416 0302 --
417 0314 3-
418 0314 --
419 031A --
420 0346 4-
421 0346 --
422 0354 --
423 0370 --
424 038C --
425 038C -4
426 03C0 --
427 03C0 -3
428 03D2 --
429 03DC --
430 03EE 3-
431 03EE --
432 03F4 --
433 0420 4-
434 0420 --
435 042E --
436 044A --
437 0466 --
438 0466 -4
439 049A --
440 049A -3
441 04AC --
442 04B0 -2
443 04B6 -1
444 04D8 --
445 0504 1-
446 0504 --
447 054A --
448 056A 2-
449 056A --
450 0572 --
451 0578 --
452 058A --
453 059C --
454 05AE --
455 05BC --
456 05D8 --
457 05EA --
458 05F8 --
459 0614 --
460 0626 --
461 0634 --
462 0650 --
463 0650 -2
464 0662 -1

L := len;
inev := false;
nwp := pi;
pi%p.fptr := pi;
pi%p.bptr := pi;
pi%p.time := 0;
pi%p.newstate := 2;
pi%p.tsi := 0;
tiptr := pi;
IF fc > 0 THEN
BEGIN
base := NIL;
END
FOR t := 1 TO tc DO
BEGIN
new(p);
read(p%trans);
p%p.fptr := base;
END;
fcptr := p
END
ELSE fcptr := NIL;
IF ic > 0 THEN
BEGIN
base := NIL;
END
FOR t := 1 TO ic DO
BEGIN
new(p);
read(p%trans);
bases := p
END;
icptr := p
END
ELSE icptr := NIL
END
FOR num:=1 TO numtrans DO
BEGIN
readln(i,tty,wid,len);
WITH trans[i] DO
BEGIN
state := 2;
trigger := false;
ttranstype := tty;
W := wid;
L := len;
new(g);
read(g%node);
gptr := g;
new(d);
read(d%node);
dptr := d;
new(s);
read(s%node);
sptr := s
END
END

- T44 -
PROCEDURE state ( front : graph ;
ind : integer ;
VAR grstate : statevalue ;
VAR grstrength : strengvalue ) ;

(* compute the state and strength of the group *)

VAR
k,l : (N,W,P,D) ;
y : statevalue ;
l : integer ;

BEGIN
k := N ; (* null strength *)
y := 2 ; (* X-state *)
WHILE front[ind] <> NIL DO
BEGIN
IF node[i].strength = 1 THEN
  l := W
ELSE
  IF node[i].strength = 2 THEN
    l := P
  ELSE
    IF node[i].strength = 3 THEN
      l := D
    ELSE
      IF l > k THEN
        BEGIN
          k := l ;
          IF k = p THEN
            y := 1
          ELSE
            y := node[i].state
        END
        ELSE
          y := node[i].state
      END
    END
  END
ELSE
  IF ( l = k ) AND ( node[i].state <> y ) THEN
  BEGIN
    front[ind] := front[ind]@.fptr
    END
  END
BEGIN
  IF k = N THEN
    writeln(' error in strength value ')
  ELSE
    IF k = W THEN grstrength := 1
    ELSE
      IF k = P THEN grstrength := 2
      ELSE
        IF k = D THEN grstrength := 3
    END
    END
  END
BEGIN
PROCEDURE poison ( VAR poilist : plink ;
groups : graph ;
gstr : gth ;
pm : integer ) ;

(* poilist is expanded to include other poisoned groups *)

- 145 -
VAR
pvectord: ARRAY[1..3] OF plink;
pl,p2,p3,pp: plink;
l: tlink;
pq: hlink;
i,j,k,q,s,t,v: integer;
found: boolean;

0000 0- A BEGIN
pp := pointlist;
pvectord[1] := NIL;
WHILE pp <> NIL DO
  BEGIN
    IF gstr[pp@.gr] = 1 THEN
      BEGIN
        new(pl);
        pl@.gr := pp@.gr;
        pl@.fptr := pvectord[1];
        pvectord[1] := pl;
      END
    ELSE
      IF gstr[pp@.gr] = 2 THEN
        BEGIN
          new(p2);
          p2@.gr := pp@.gr;
          p2@.fptr := pvectord[2];
        END
      ELSE
        BEGIN
          new(p3);
          p3@.gr := pp@.gr;
          p3@.fptr := pvectord[3];
        END;
    pp := pp@.fptr
  END;
FOR t := 3 DOWNTO s DO
  WHILE pvectord[t] <> NIL DO
    BEGIN
      v := pvectord[t]@.gr;
      pvectord[t] := pvectord[t]@.fptr;
      WHILE groups[v] <> NIL DO
        BEGIN
          j := groups[v]@.node;
          i := node[j].icptr;
          IF l <> NIL THEN
            WHILE l <> NIL DO
              BEGIN
                i := l@.trans;
                IF ( trans[i].state = 2 ) THEN
                  BEGIN
                    IF ( trans[i].dptr@.node = j ) THEN
                      k := trans[i].sptr@.node
                    END
                  END
          END
    END
  END
END
ELSE
  k := trans[i].dptr@.node;
  FOR q := 1 TO pm DO
    BEGIN
      pq := groups[q];
      WHILE pq <> NIL DO
        BEGIN
          IF k = pq@.node THEN
            BEGIN
              IF ( gstr[q] < gstr[v] ) THEN
                BEGIN
                  new(pp);
                  pp@.gr := q;
                  pp@.fptr := poilist;
                  poilist := pp;
                  gstr[q] := gstr[v];
                  new(pp);
                  pp@.gr := q;
                  pp@.fptr := pvector[t];
                  pvector[t] := pp
                END
            ELSE
              BEGIN
                IF ( gstr[q] = gstr[v] ) THEN
                  BEGIN
                    pp := poilist;
                    found := false;
                    WHILE pp <> NIL DO
                      BEGIN
                        IF q = pp@.gr THEN
                          BEGIN
                            found := true;
                            pp := NIL
                          END
                        ELSE
                          BEGIN
                            pp := pp@.fptr
                          END
                        END
                    END
                  END
                ELSE
                  BEGIN
                    new(pp);
                    pp@.gr := q;
                    pp@.fptr := poilist;
                    poilist := pp;
                    gstr[q] := gstr[v];
                    new(pp);
                    pp@.gr := q;
                    pp@.fptr := pvector[t];
                    pvector[t] := pp
                  END
                END
            END
        END
      END
    END
  END
END;

END;
l := l@.fptr
END;
groups[v] := groups[v]@.fptr

- 147 -
PROCEDURE nodestate ( head : graph ;
inx : integer ) ;

VAR
    ptr : hlink ;
    p,pptr : plink ;
    gstate : gte ;
    gstrength : gth ;

BEGIN
    state(head ,inx ,sta,str) ;
    gstate[0] := sta ;
    gstrength[0] := str ;

    IF m > 1 THEN

        BEGIN
            pptr := NIL ;
            r := m ;
            WHILE r > 0 DO
            BEGIN
                state(head ,r,sta,str) ;
                gstate[r] := sta ;
                gstrength[r] := str ;

                IF gstate[r] <> gstate[0] THEN
                    BEGIN
                        new(p) ;
                        p@.gr := r ;
                        p@.fptr := pptr ;
                        pptr := p
                    END ;
            END ;

            r := r - 1

        END ;

    END ;

    poison(pptr,head ,gstrength,m) ;

    WHILE pptr <> NIL DO

        BEGIN
            gstate[pptr@.gr] := -2 ;
            pptr := pptr@.fptr
        END ;

    END ;

    FOR r := 1 TO m DO

        BEGIN
            head[r] <> NIL DO

            BEGIN
                i := head[r]@.node ;
                node[i].trigger := false ;
                IF ( node[i].strength <> 3 ) THEN
                    BEGIN
                        IF ( node[i].state <> gstate[r] ) THEN

                            BEGIN
                                node[i].newsta := gstate[r] ;
                                node[i].trigger := true
                            END

                        ELSE
                            IF ( node[i].newsta <> gstate[r] ) THEN...
BEGIN
node[i].newsta := gstate[r];
node[i].trigger := true
END;
head[r] := head[r]@.fptr
END
ELSE
WHILE head[inx] <> NIL DO
BEGIN
  i := head[inx]@.node;
  node[i].trigger := false;
  IF (node[i].strength <> 3) THEN
    IF (node[i].state <> gstate[0]) THEN
      BEGIN
        node[i].newsta := gstate[0];
        node[i].trigger := true;
      END
    ELSE
      BEGIN
        IF (node[i].newsta <> gstate[0]) THEN
          BEGIN
            node[i].newsta := gstate[0];
            node[i].trigger := true;
          END
        END
      END
  END
END
PROCEDURE append (VAR add1 : hlink);
BEGIN
  VAR list : graph;
  x : integer;
  VAR
    ad : hlink;
    evpen : pendlink;
  BEGIN
    found := false;
    evpen := evpend;
    WHILE evpen <> NIL DO
      IF evpen@.group = x THEN
        BEGIN
          found := true;
          evpen := NIL;
        END
      ELSE
        evpen := evpen@.fptr;
      END
    IF found = false THEN
      BEGIN
        new(evpen);
        evpen@.group := x;
        evpen@.fptr := evpend;
        evpend := evpen;
        ad := list[x];
        WHILE ad@.fptr <> NIL DO
          BEGIN
            evpen := evpen@.fptr;
            found := true;
            evpen := NIL;
          END
        END
      END
END;
PASCAL 8000/2.0B

AAEC (17FEB82)

755 014E -- ad := ad@.fptr ;
756 0156 -- ad@.fptr := addl ;
757 015E -- addl := list[x] ;
758 0165 -- list[x] := NIL
759 0170 -- END
760 0177 -- A END ;
761 0183 --
762 0183 -- A PROCEDURE delete ( VAR del : hlink ) ;
763 0190 -- (* delete those used storage area *)
764 0195 --
765 0195 -- VAR
766 019B -- dele : hlink ;
767 01A3 --
768 01A7 --
769 01AA -- A BEGIN
770 01B2 -- WHILE del <> NIL DO
771 01B9 -- BEGIN
772 01BB -- dele := del ;
773 01C0 -- del := del@.fptr ;
774 01C8 -- dispose(dele)
775 01CF -- END
776 01D0 -- A END ;
777 01DE --
778 01EF -- A PROCEDURE searchpath ( VAR path : pathline ;
779 01F7 -- VAR g : integer ;
780 01F8 -- m : integer ) ;
781 01F9 --
782 0200 -- (* search for direct paths between the input and
783 0205 -- the output nodes *)
784 020A --
785 0210 -- VAR
786 0215 -- l : trlink ;
787 0219 -- tem,base : pathlink ;
788 021E -- pass : boolean ;
789 0223 -- i,j,k : integer ;
790 0227 --
791 022B -- A BEGIN
792 0232 -- node[m].visit := true ;
793 0237 -- l := node[m].icptr ;
794 023C -- IF node[m].ic >= 2 THEN
795 0242 -- BEGIN
796 0247 -- new(tem) ;
797 024D -- tem@ := path[g]@ ;
798 0253 -- pass := false ;
799 0258 -- FOR i := 1 TO ( node[m].ic ) DO
800 025D -- BEGIN
801 0262 -- j := l@.trans ;
802 0267 -- IF (trans[j].trigger = true) AND
803 026C -- (node[trans[j].gptr@.node].tiptr@.fptr@.time = ST) THEN
804 0272 -- BEGIN
805 0277 -- ii := trans[j].gptr@.node ;
806 0282 -- new(inp) ;
807 0287 -- inp@.node := ii ;
808 028C -- inp@.fptr := inp ;
809 0291 -- inp := inp
810 0296 -- END ;
811 029B -- IF (trans[j].state = 2) OR (trans[j].state = 1) THEN
812 02AC -- BEGIN

- 150 -
813 0314 -- IF (trans[j].dptr@.node <> m) AND  
814 035C -- (node[trans[j].dptr@.node].visit = false) THEN
815 03BC 4- BEGIN
816 03BC -- IF pass = true THEN
817 03CE -- q := q + 1
818 03DE -- ELSE
819 03EC -- pass := true;
820 03F4 -- new(path[q]);
821 0420 -- path[q]@ := tem@;
822 0458 -- k := trans[j].dptr@.node;
823 048E -- new(base);
824 049C -- base@.fptr := path[q];
825 04D6 -- base@.node := k;
826 04E6 -- path[q] := base;
827 051E -- searchpath(path,q,k)
828 053C -4 END
829 0540 -- ELSE
830 0544 -- IF (trans[j].sptr@.node <> m) AND  
831 058C -- (node[trans[j].sptr@.node].visit = false) THEN
832 05EC 4- BEGIN
833 05EC -- IF pass = true THEN
834 05FE -- q := q + 1
835 060C -- ELSE
836 061C -- pass := true;
837 0624 -- new(path[q]);
838 0650 -- path[q]@ := tem@;
839 0688 -- k := trans[j].sptr@.node;
840 06BE -- new(base);
841 06CC -- base@.fptr := path[q];
842 0706 -- base@.node := k;
843 071E -- path[q] := base;
844 074E -- searchpath(path,q,k)
845 076C -4 END
846 0770 -3 END;
847 0770 -- l := l@.fptr
848 077A -2 END
849 078C -1 END,
850 07AE -0 A END ;
851 07D8 -- A PROCEDURE paths ( head : hlink ) ;
852 07D8 -- A PROCEDURE paths ( head : hlink ) ;
853 0044 -- (* searching for all the direct paths *)
854 0044 -- (* searching for all the direct paths *)
855 0044 --
856 0044 -4 VAR
857 0044 -- base : pathlink ;
858 0048 -- point : hlink ;
859 004C -- l : trlink ;
860 0050 -- i,j,k : integer ;
861 005C --
862 0000 0- A BEGIN
863 0012 -- point := head ;
864 0024 -- pullup := false ;
865 0030 -- WHILE point <> NIL DO
866 0042 1- BEGIN
867 0042 -- i := point@.node ;
868 005A -- IF node[i].strength = 2 THEN
869 0084 2- BEGIN
870 0084 -- pn := i ;
0098 -- pullup := true;
00A6 -- point := NIL
00A6 -2 END
00AC -- ELSE
00B0 -- point := point@.fptr
00BA -1 END;
00D0 -- IF pullup = false THEN
00E8 1- BEGIN
00E8 -- point := head;
00FA -- WHILE point <> NIL DO
010C 2- BEGIN
010C -- i := point@.node;
0124 -- IF node[i].strength = 3 THEN
014E 3- BEGIN
014E -- pn := i;
0162 -- IF node[i].inew = true THEN
018C 4- BEGIN
018C -- ii := i;
01A0 -- new(inp);
01B4 -- inp@.node := ii;
01B8 -- inp@.fptr := inp;
01DB -- inp := inp;
0200 -- END;
0200 -4 END;
021E -- point := NIL
021E -3 END
0224 -- ELSE
0228 -- point := point@.fptr
0232 -2 END
0244 -1 END;
0248 -- point := head;
025A -- WHILE point <> NIL DO
026C 1- BEGIN
026C -- node[point@.node].visit := false;
0294 -- point := point@.fptr
0294 -- END;
02A4 -- q := 0;
02B4 -- node[pn].visit := true;
02B6 -- IF (node[pn].trigger = true) AND
031C -- (node[pn].nodetype = 2) THEN
035A 1- BEGIN
035A -- q := q + 1;
0378 -- new(path[q]);
0394 -- path[q]@.fptr := NIL;
03D2 -- path[q]@.node := pn;
040E -- path[q]@.no. := 0
043E -1 END;
043C -- l := node[pn].icptr;
046C -- FOR i := 1 TO node[pn].ic. DO
04B6 1- BEGIN
04B6 -- j := 1@.trans;
04CE -- IF (trans[j].trigger = true) AND
0504 -- (node[trans[j].gptr@.node].tiptr@.fptr@.time = ST) THEN
0586 2- BEGIN
0586 -- ii := trans[j].gptr@.node;
05C2 -- new(inp);
05D6 -- inp@.node := ii;
05FA -- inp@.fptr := inp;
0622 --
929 0622 -2  END ;
930 0640 --  IF trans[j].state <> 0 THEN
931 0670 2-  BEGIN
932 0670  new(path[q]);
933 068E --  path[q].fptr := NIL;
934 06BA --  path[q].node := pn;
935 06F8 --  IF (trans[j].dptr@.node <> pn) AND
936 0724 --  (node[trans[j].dptr@.node].visit = false) THEN
937 0772 --  BEGIN
938 0822 --  k := trans[j].dptr@.node;
939 0850 --  new(base);
940 0888 --  base@.fptr := path[q];
941 08C0 --  base@.node := k;
942 08F8 --  path[q] := base;
943 09C0 --  searchpath(path,q,k)
944 09C8 --  END
945 0BAC 3-  ELSE
946 0BB2 --  IF (trans[j].sptr@.node <> pn) AND
947 0C2C --  (node[trans[j].sptr@.node].visit = false) THEN
948 0D70 3-  BEGIN
949 0D98 --  k := trans[j].sptr@.node;
950 0E46 --  new(base);
951 0EB8 --  base@.fptr := path[q];
952 0F20 --  base@.node := k;
953 0F98 --  path[q] := base;
954 10C0 --  searchpath(path,q,k)
955 10C8 --  END
956 10D4 3-  END
957 10E0 2-  1 := 1@.fptr
958 10F8 0  A END ;
961 086C A PROCEDURE delays ( head : hlink );
962 0A24 A (* computing the delay for each output node which has
963 0A4C A changed state *)
964 0A64 0  VAR
965 0A84 A pi : tilink';
966 0AA4 A storage,base,pa,pp : pathlink /  
967 0ACC A first,point : hlink;
968 0C44 A l : trlink;
969 0D04 A i,j,k,r,di,beta,delay : integer;
970 0D84 A totcap,tot,tstp,tsis,tin,ratio,dratio,tdelel : real;
971 0DE4 A occur : boolean;
972 0F14 A
973 0F34 A BEGIN
974 0F54 A first := head;
975 0F74 A WHILE first <> NIL DO
976 0F94 A BEGIN
977 0FB4 A IF (node[first@.node].trigger = true) AND
978 0FD4 A (node[first@.node].nodetype = 2) THEN
979 0038 1-  BEGIN
980 0058 --  IF (node[first@.node].newsta = 2) THEN
981 0078 2-  BEGIN
982 0098 A (* if the new state is X *)
983 00B2 A delay := 1;
984 00D4 A tm := ST + delay;
985 00F4 A
tt := 0;
spike(first@.node,node[first@.node].newsta,tm,tt)
END
ELSE
IF (node[first@.node].newsta = 1) THEN
BEGIN
(* if the new state is 1 *)
paths(head);
r := 0;
FOR i := 1 TO q DO
IF (node[path[i]@.node].strength = 3) OR
((node[path[i]@.node].strength = 1) AND
(node[path[i]@.node].nodetype <> 2)) THEN
path[i] := NIL
ELSE
BEGIN
r := r + 1;
path[r] := path[i];
IF path[r]@.node = first@.node THEN
di := r
END;
FOR i := 1 TO r DO
BEGIN
j := 0;
pa := path[i];
WHILE pa@.fptr <> NIL DO
BEGIN
j := j + 1;
pa := pa@.fptr
END;
pa := path[i];
WHILE pa <> NIL DO
BEGIN
pa@.no := j;
j := j - 1;
pa := pa@.fptr
END
END;
IF pullup = true THEN
BEGIN
totcap := 0;
storage := NIL;
FOR i := 1 TO r DO
BEGIN
pa := path[i];
WHILE pa <> NIL DO
BEGIN
pp := storage;
occur := false;
WHILE pp <> NIL DO
IF pa@.node <> pp@.node THEN
pp := pp@.fptr
ELSE
BEGIN
occur := true;
pp := NIL
END;
IF occur = false THEN
BEGIN
new(base);
base.@node := pa.@node;
base.@no := pa.@no;
base.@fptr := storage;
storage := base;
totcap := (1 / sqrt(pa.@no + 1)) * node[p*a.@node].capvalue + totcap
END;

pa := pa.@fptr
END

END;

ratio := node[pi].W / node[pi].L;
tstp := 3.6976 * (1 / (UC * VDD)) *
(1 / ratio) * totcap;
pi := node[di].tiptr;
REPEAT
    pi := pi.@bptr
UNTIL pi.@time = ST;
IF pi.@tsi <> 0 THEN
    tsis := (1 + 0.3 * (pi.@tsi / tstp) ** 1.5)
    tstp
ELSE
    tsis := tstp
    tdel := tsis / sqrt(sqr(2.9) + 0.15 *
    sqrt(node[pi].capvalue / totcap))
END
ELSE
    BEGIN
        pi := node[di].tiptr;
        REPEAT
            pi := pi.@bptr
        UNTIL pi.@time = ST;
        tsis := pi.@tsi;
        tdel := 0
    END;
    IF path[di].@no = 0 THEN
        BEGIN
            tm := ST + round(tdel * 1.0E9);
            tt := tsis;
            spike(first.@node, node[first.@node].newsta, tm, tt)
        END;
    ELSE
        BEGIN
            totcap := 0;
tin := tsis;
    pa := path[di];
    WHILE pa <> NIL DO
        BEGIN
            IF pa.@no <> 0 THEN
                totcap := totcap + sqrt(pa.@no)
                node[pa.@node].capvalue
            pa := pa.@fptr
        END;
    FOR i := 1 TO r DO
        IF i <> di THEN
            BEGIN
                tot := 0;
                pa := path[i];
WHILE pa <> NIL DO
  BEGIN
    pp := path[di] ;
    occur := false ;
    WHILE pp <> NIL DO
      IF pa@.node <> pp@.node THEN
        pp := pp@.fptr
      ELSE
        BEGIN
          occur := true ;
          pp := NIL
          END ;
      END
      IF occur = false THEN
        BEGIN
          tot := node[pa@.node].capvalue + tot ;
          pa := pa@.fptr
          END
        ELSE
          BEGIN
            totcap := sqrt(pa@.no) * tot + totcap ;
            pa := NIL
            END
      END
    END ;
  pa := path[di] ;
  l := node[pa@.node].icptr ;
  ratio := trans[1@.trans],W / trans[1@.trans],L ;
  tscp := 17.778 * ( 1 / ( UC * ( VDD - VTE ) ) ) * 
         sqrt(path[di]@.no) * ( 1 / ratio ) * 
         totcap ;
  IF tin <> 0 THEN
    tsis := ( 1 + 0.3 * ( tin / tscp ) ** 1.5 ) * 
    tscp
  ELSE
    tsis := tscp ;
    tdel := tsis / 8.9 ;
    tm := ST + round(tdel * 1.0E9) ;
    tt := tsis ;
    i := first@.node ;
    spike(i,node[i].newsta,tm,tt)
  END
  END
ELSE
  BEGIN
    (* if the new state is 0 *)
    paths(head) ;
    r := 0 ;
    dratio := 0 ;
    FOR i := 1 TO q DO
      BEGIN
        ratio := 0 ;
        IF (node[path[i]@.node].strength = 3) THEN
          BEGIN
            pa := path[i] ;
            WHILE pa <> NIL DO
              BEGIN
                IF pa@.node <> pn THEN
                  IF pa@.fptr = pn THEN
                    pa := pa@.next
                  ELSE
                    pa := pa@.fptr
                  END
                END
              pa := pa@.fptr
            END
          END
        END
      END
BEGIN
  l := node[pa.@.node].icptr;
  WHILE l <> NIL DO
    BEGIN
      IF (trans[l@.trans].dpotr.@.node = pa.@.fpotr.@.node) OR
          (trans[l@.trans].spotr.@.node = pa.@.fpotr.@.node) THEN
        ratio := (trans[l@.trans].L / trans[l@.trans].W) + ratio;
        l := l@.fpotr
      END;
      pa := pa.@.fpotr
      END;
      dratio := (dratio + (1 / ratio));
      path[i] := NIL
    END;
    ELSE
      BEGIN
        IF (node[path[i]@.node].strength = 1) AND
            (node[path[i]@.node].nodetype <> 2) THEN
          path[i] := NIL
        ELSE
          BEGIN
            r := r + 1;
            path[r] := path[i];
            IF path[r]@.node = first@.node THEN
              di := r
            END
          END
      END
    END;
FOR i := 1 TO r DO
  BEGIN
    j := 0;
    pa := path[i];
    WHILE pa@.fpotr <> NIL DO
      BEGIN
        j := j + 1;
        pa := pa@.fpotr
      END;
    pa := path[i];
    WHILE pa <> NIL DO
      BEGIN
        pa@.no := j;
        j := j - 1;
        pa := pa@.fpotr
      END
  END;
END;
If pullup = true THEN
BEGIN
  totcap := 0;
  storage := NIL;
FOR i := 1 TO r DO
  BEGIN
    pa := path[i];
    WHILE pa <> NIL DO
      BEGIN
        END;
    END;
BEGIN
pp := storage;
occur := false;
WHILE pp <> NIL DO
  IF pa@.node <> pp@.node THEN
    pp := pp@.fptr
  ELSE
    BEGIN
      occur := true;
      pp := NIL
    END;
  IF occur = false THEN
    BEGIN
      new(base);
      base@.node := pa@.node;
      base@.no := pa@.no;
      base@.fptr := storage;
      storage := base;
      totcap := (1 / sqrt(pa@.no + 1)) * node[pa@.node].capvalue + totcap
    END;
  pa := pa@.fptr
END
ratio := node[pi].W / node[pi].L;
beta := round(dratio / ratio);
IF ((beta > 2) AND (beta < 6)) THEN
  tsp := 4.0624 * (1 / (UC * VDD)) *
        (1 / dratio) * totcap
ELSE
  IF ((beta > 6) AND (beta < 10)) THEN
    tsp := 3.8758 * (1 / (UC * VDD)) *
           (1 / dratio) * totcap
  ELSE
    writeln('beta = ', beta, ' error');
    pi := node[pi].tiptr;
    REPEAT
      pi := pi@.bptr
    UNTIL pi@.time = ST;
    IF pi@.tsi <> 0 THEN
      tsis := (1 + 0.25 * (pi@.tsi / tsp)) ** 1.1
        * tsp
    ELSE
      tsis := tsp;
      tdel := tsis / sqrt(node[pi].capvalue / totcap)
    END
  ELSE
    BEGIN
      pi := node[pi].tiptr;
      REPEAT
        pi := pi@.bptr
      UNTIL pi@.time = ST;
      tsis := pi@.tsi;
      tdel := 0
    END;
  IF path[di]@.no = 0 THEN
  -158-
BEGIN
  tm := ST + round(tdel * 1.0E9);
  tt := tsis;
  spike(first@.node, node[first@.node].newsta, tm, tt)
END
ELSE
  BEGIN
    totcap := 0;
    tin := tsis;
    pa := path[di];
    WHILE pa <> NIL DO
      BEGIN
        IF pa@.no <> 0 THEN
          totcap := totcap + sqrt(pa@.no) * node[pa@.node].capvalue;
          pa := pa@.fptr;
        END;
      END;
    FOR i := 1 TO r DO
      IF i <> di THEN
        BEGIN
          tot := 0;
          pa := path[i];
          WHILE pa <> NIL DO
            BEGIN
              IF pa@.node <> pp@.node THEN
                pp := pp@.fptr;
              ELSE
                BEGIN
                  occur := true;
                  pp := NIL;
                END;
              IF occur = false THEN
                BEGIN
                  tot := node[pa@.node].capvalue + tot;
                  pa := pa@.fptr;
                END;
              ELSE
                BEGIN
                  totcap := sqrt(pa@.no) * tot + totcap;
                  pa := NIL;
                END;
            END;
        END;
      END;
    END;
  END;
END;
tsis := tstp;
tdel := tsis / 2.5;
tm := ST + round(tdel * 1.0E9);
tt := tsis;
i := first@.node;
spike(i,node[i].newsta,tm,tt)

END

end;

first := first@.fptr

END

A END ;

PROCEDURE simulate ;

(* perform the simulation process *)

VAR
Tempty,stop,outev,share : boolean ;
i,k,delay : integer ;

BEGIN

REPEAT

evpend := NIL ;
EVL := NIL ;
ST := Z * MTC + t - 1 ;
IF TQ[t] <> NIL THEN
  BEGIN
    WHILE TQ[t] <> NIL DO
      BEGIN
        i := TQ[t]@.node ;
        IF ( node[i].state <> TQ[t]@.newsta ) THEN
          BEGIN
            node[i].state := TQ[t]@.newsta ;
            IF ( node[i].strength = 3 ) AND ( node[i].ic > 0 )
              THEN
                BEGIN
                  node[i].inv := true ;
                  j := node[i].group ;
                  append(EVL,subnet,j)
                END ;
          END ;

          fp := node[i].fcptr ;
          WHILE fp <> NIL DO
            BEGIN
              IF ( trans[fp@.trans].transtype = 1 ) THEN
                BEGIN
                  trans[fp@.trans].trigger := true ;
                  trans[fp@.trans].state := node[i].state
                END
              ELSE
                IF ( trans[fp@.trans].transtype = 2 ) THEN
                  BEGIN
                    trans[fp@.trans].trigger := true ;
                    IF ( node[i].state = 0 ) THEN
                      trans[fp@.trans].state := 1
                    ELSE
...
IF ( node[i].state = 1 ) THEN
  trans[fp@.trans].state := 0
ELSE
  trans[fp@.trans].state := 2
END
ELSE
BEGIN
  trans[fp@.trans].trigger := true;
  trans[fp@.trans].state := 1
END;
ν := trans[fp@.trans].dptr@.node;
w := trans[fp@.trans].spt@.node;
IF ( node[ν].group = node[w].group ) THEN
  append(EVL, subnet, node[ν].group)
ELSE
  append(EVL, subnet, node[ν].group);
  append(EVL, subnet, node[w].group)
END;
fp := fp@.fptr
END;
TQ[t] := TQ[t]@.fptr
END;
sub[0] := EVL;
partition(sub[0], 0, 1, 1, 2);
delete(sub[0]);
inpu := NIL;
FOR k := 1 TO m DO
BEGIN
  n := n + 1;
  subnet[n] := sub[k];
  point := subnet[n];
  WHILE point <= NIL DO
  BEGIN
    i := point@.node;
    node[i].group := n;
    point := point@.fptr
  END;
  nodestate(subnet, n);
  point := subnet[n];
  outev := false;
  WHILE point <= NIL DO
  BEGIN
    i := point@.node;
    IF ( node[i].trigger = true ) THEN
      IF ( node[i].nodetype = 1 ) THEN
        node[i].state := node[i].newsta
      ELSE
        IF unit = true THEN
          BEGIN
            delay := 1;
            tm := ST + delay;
            tt := 0;
            appenddev(i, node[i].newsta, tm, tt);
          END
        ELSE
          outev := true;
      END
  END
- 161 -
point := point@.fptr
END;

share := true;
IF (outev = true) THEN
BEGIN
point := subnet[n];
WHILE point <> NIL DO
BEGIN
IF (node[point@.node].strength = 2) OR
(node[point@.node].strength = 3) THEN
BEGIN
share := false;
point := NIL
END
ELSE
BEGIN
point := point@.fptr
END;
IF share = true THEN
BEGIN
point := subnet[n];
WHILE point <> NIL DO
BEGIN
IF node[point@.node].nodetype = 2 THEN
BEGIN
i := point@.node;
tm := ST + 1;
spike(i,node[i].newsta,tm,0)
END;
point := point@.fptr
END
ELSE
END
END
END
END;
END
END
END;
END
END
END;
END
END
BEGIN
i := input@.node;
IF node[i].inev = true THEN
node[i].inev := false;
pi := node[i].tptr@.fptr;
IF pi@.time = ST THEN
BEGIN
node[i].tptr@.fptr := pi@.fptr;
pi@.fptr@.bptr := node[i].tptr
END;
inpu := input@.fptr
END;
END
BEGIN
FOR k := 1 TO m DO
BEGIN
point := sub[k];
WHILE point <> NIL DO
BEGIN
l := node[point@.node].icptr;
WHILE l <> NIL DO
BEGIN
IF trans[l@.trans].trigger = true THEN
trans[l@.trans].trigger := false;
END
END
END
END
PASCAL 8000/2.0B

AAEC (17FEB82)

1509 1198 --  l := l@.fptr
1510 11A8 -5  END ;
1511 11C4 --  point := point@.fptr
1512 11D4 -4  END ;
1513 11EC -3  END ;
1514 11F0 -2  END ;
1515 1212 t := IF unit = false THEN
1516 122A 2-  BEGIN
1517 122A --  write(' time ',ST', node 1 ',node[1].state);
1518 128A --  writeln(' node 5 ',node[5].state,' node 6 ',node[6].state)
1519 12B4 -2  END ;
1520 12BA --  t := t + 1 ;
1521 1308 --  IF t = Z + 1 THEN
1522 1326 2-  BEGIN
1523 1326 --  FOR i := 1 TO Z DO
1524 1344 3-  BEGIN
1525 1344 --  TQ[i] := TQ[Z+i] ;
1526 138A --  TQ[Z+i] := NIL
1527 13A0 -3  END ;
1528 13CE --  MTC := MTC + 1 ;
1529 13EC --  t := 1 ;
1530 13FA --  WHILE ( MTEL@.fptr@.time <= ( Z * MTC + Z2 - 1 ) ) AND
1531 144C --  ( MTEL <> MTEL@.fptr ) DO
1532 148A 3-  BEGIN
1533 148A --  j := MTEL@.fptr@.time - ( Z * MTC - 1 ) ;
1534 14BA --  new(event);
1535 14BC --  WITH event@ DO
1536 14EC --  BEGIN
1537 14FE 4-  fptr := TQ[j] ;
1538 152E --  node := MTEL@.fptr@.node ;
1539 1556 --  newsta := MTEL@.fptr@.newsta ;
1540 1582 --  time := MTEL@.fptr@.time ;
1541 15AA --  tt := MTEL@.fptr@.tt
1542 15C4 -4  END ;
1543 15D6 --  TQ[j] := event ;
1544 160C --  ev := MTEL@.fptr ;
1545 1634 --  MTEL@.fptr := ev@.fptr ;
1546 1666 --  ev@.fptr@.bptr := MTEL
1547 1680 -3  END ;
1548 169B -2  END ;
1549 169C --  i := 1 ;
1550 16A4 --  TQempty := true ;
1551 16AC --  WHILE i <= Z2 DO
1552 16BE --  IF TQ[.i.] = NIL THEN
1553 16B8 --  i := i + 1
1554 16F2 --  ELSE
1555 16FE 2-  BEGIN
1556 16FE --  i := Z2 + 1 ;
1557 170A --  TQempty := false
1558 170A -2  END ;
1559 1714 --  IF ( TQempty = true ) AND ( MTEL = MTEL@.fptr ) THEN
1560 176A --  stop := true
1561 176A --  ELSE
1562 1776 --  stop := false
1563 1776 -1  UNTIL stop = true
1564 1786 -0 A END ;
BEGIN
  n := 0;
  MTC := 0;
  t := 1;
  new(MTEL); /* create an empty overflow event list */
  WITH MTEL DO
    BEGIN
      fpotr := MTEL;
      bptr := MTEL;
      node := 0;
      newsta := 0;
      time := 0;
      END;
    FOR i := 1 TO Z2 DO /* create an empty TQ array */
      TQ[i] := NIL;
      readparameter;
      sub[0] := network[0];
      partition(sub,0,1,1,2);
      delete(sub[0]);
      FOR i := 1 TO m DO
        BEGIN
          n := n + 1;
          subnet[n] := sub[i];
          point := subnet[n];
          WHILE point <> NIL DO
            BEGIN
              j := point@.node;
              node[j].group := n;
              point := point@.fptr
            END
        END
    FOR i := 1 TO evnum DO
      BEGIN
        read(i,sta,tm,tt);
        new(tm); /* simulate */
        read(evnum);
        FOR j := 1 TO evnum DO
          BEGIN
            read(i,sta,tm,tt);
            new(tm); /* simulate */
            read(evnum);
            FOR j := 1 TO evnum DO
              BEGIN
                read(i,sta,tm,tt);
                new(tm); /* simulate */
                read(evnum);
                FOR j := 1 TO evnum DO
                  BEGIN
                    read(i,sta,tm,tt);
                    new(tm); /* simulate */
                    read(evnum);
                    FOR j := 1 TO evnum DO
                      BEGIN
                        read(i,sta,tm,tt);
                        new(tm); /* simulate */
                        read(evnum);
                        FOR j := 1 TO evnum DO
                          BEGIN
                            read(i,sta,tm,tt);
                            new(tm); /* simulate */
                            read(evnum);
                            FOR j := 1 TO evnum DO
                              BEGIN
                                read(i,sta,tm,tt);
                                new(tm); /* simulate */
                                read(evnum);
                                FOR j := 1 TO evnum DO
                                  BEGIN
                                    read(i,sta,tm,tt);
                                    new(tm); /* simulate */
                                    read(evnum);
                                    FOR j := 1 TO evnum DO
                                      BEGIN
                                        read(i,sta,tm,tt);
                                        new(tm); /* simulate */
                                        read(evnum);
                                        FOR j := 1 TO evnum DO
                                          BEGIN
                                            read(i,sta,tm,tt);
                                            new(tm); /* simulate */
                                            read(evnum);
                                            FOR j := 1 TO evnum DO
                                              BEGIN
                                                read(i,sta,tm,tt);
                                                new(tm); /* simulate */
                                                read(evnum);
                                                FOR j := 1 TO evnum DO
                                                  BEGIN
                                                    read(i,sta,tm,tt);
                                                    new(tm); /* simulate */
                                                    read(evnum);
                                                    FOR j := 1 TO evnum DO
                                                      BEGIN
                                                        read(i,sta,tm,tt);
                                                        new(tm); /* simulate */
                                                        read(evnum);
                                                        FOR j := 1 TO evnum DO
                                                          BEGIN
                                                            read(i,sta,tm,tt);
                                                            new(tm); /* simulate */
                                                            read(evnum);
                                                            FOR j := 1 TO evnum DO
                                                              BEGIN
                                                                read(i,sta,tm,tt);
                                                                new(tm); /* simulate */
                                                                read(evnum);
                                                                FOR j := 1 TO evnum DO
                                                                  BEGIN
                                                                    read(i,sta,tm,tt);
                                                                    new(tm); /* simulate */
                                                                    read(evnum);
                                                                    FOR j := 1 TO evnum DO
                                                                      BEGIN
                                                                        read(i,sta,tm,tt);
                                                                        new(tm); /* simulate */
                                                                        read(evnum);
                                                                        FOR j := 1 TO evnum DO
                                                                          BEGIN
                                                                            read(i,sta,tm,tt);
                                                                            new(tm); /* simulate */
                                                                            read(evnum);
                                                                            FOR j := 1 TO evnum DO
                                                                              BEGIN
                                                                                read(i,sta,tm,tt);
                                                                                new(tm); /* simulate */
                                                                                read(evnum);
                                                                                FOR j := 1 TO evnum DO
                                                                                  BEGIN
                                                                                    read(i,sta,tm,tt);
                                                                                    new(tm); /* simulate */
                                                                                    read(evnum);
                                                                                    FOR j := 1 TO evnum DO
                                                                                      BEGIN
                                                                                          read(i,sta,tm,tt);
                                                                                          new(tm); /* simulate */
                                                                                          read(evnum);
                                                                                          FOR j := 1 TO evnum DO
                                                                                              BEGIN
                                                                                                  read(i,sta,tm,tt);
                                                                                                  new(tm); /* simulate */
                                                                                                  read(evnum);
                                                                                                  FOR j := 1 TO evnum DO
                                                                                                    BEGIN
                                                                                                        read(i,sta,tm,tt);
                                                                                                        new(tm); /* simulate */
                                                                                                        read(evnum);
                                                                                                        FOR j := 1 TO evnum DO
                                                                                                          BEGIN
                                                                                                              read(i,sta,tm,tt);
                                                                                                              new(tm); /* simulate */
                                                                                                              read(evnum);
                                                                                                              FOR j := 1 TO evnum DO
                                                                                                                BEGIN
                                                                                                                    read(i,sta,tm,tt);
                                                                                                                    new(tm); /* simulate */
                                                                                                                    read(evnum);
                                                                                                                    FOR j := 1 TO evnum DO
                                                                                                                      BEGIN
                                                                                                                          read(i,sta,tm,tt);
                                                                                                                          new(tm); /* simulate */
                                                                                                                          read(evnum);
                                                                                                                          FOR j := 1 TO evnum DO
                                                                                                                            BEGIN
                                                                                                                                read(i,sta,tm,tt);
                                                                                                                                new(tm); /* simulate */
                                                                                                                                read(evnum);
                                                                                                                                FOR j := 1 TO evnum DO
                                                                                                                                  BEGIN
                                                                                                                                      read(i,sta,tm,tt);
                                                                                                                                      new(tm); /* simulate */
                                                                                                                                      read(evnum);
                                                                                                                                      FOR j := 1 TO evnum DO
                                                                                                                                        BEGIN
                                                                                                                                            read(i,sta,tm,tt);
                                                                                                                                            new(tm); /* simulate */
                                                                                                                                            read(evnum);
                                                                                                                                            FOR j := 1 TO evnum DO
                                                                                                                                              BEGIN
                                                                                                                                                  read(i,sta,tm,tt);
                                                                                                                                                  new(tm); /* simulate */
                                                                                                                                                  read(evnum);
                                                                                                                                                  FOR j := 1 TO evnum DO
                                                                                                                                                    BEGIN
                                                                                                                                                      read(i,sta,tm,tt);
                                                                                                                                                      new(tm); /* simulate */
                                                                                                                                                      read(evnum);
                                                                                                                                                      FOR j := 1 TO evnum DO
                                                                                                                                                        BEGIN
                                                                                                                                                          read(i,sta,tm,tt);
                                                                                                                                                          new(tm); /* simulate */
                                                                                                                                                          read(evnum);
                                                                                                                                                          FOR j := 1 TO evnum DO
                                                                                                                                                                        BEGIN
                                                                                                                                                                            read(i,sta,tm,tt);
                                                                                                                                                                            new(tm); /* simulate */
                                                                                                                                                                            read(evnum);
                                                                                                                                                                            FOR j := 1 TO evnum DO
                                                                                                                                                                                BEGIN
                                                                                                                                                                                    read(i,sta,tm,tt);
                                                                                                                                                                                    new(tm); /* simulate */
                                                                                                                                                                                    read(evnum);
                                                                                                                                                                                    FOR j := 1 TO evnum DO
                                                                                                                                                                                        BEGIN
                                                                                                                                                                                            read(i,sta,tm,tt);
                                                                                                                            /* simulate */
                                                                                                                        FOR j := 1 TO evnum DO
                                                                                        BEGIN
                                                                                                                read(i,sta,tm,tt);
                                                                                                                new(tm); /* simulate */
                                                                                                                read(evnum);
                                                                                                                FOR j := 1 TO evnum DO
                                                                                                                  BEGIN
                                                                                                                      read(i,sta,tm,tt);
                                                                                                                      new(tm); /* simulate */
                                                                                                                      read(evnum);
                                                                                                                      FOR j := 1 TO evnum DO
                                                                                                                        BEGIN
                                                                                                                            read(i,sta,tm,tt);
                                                                                                                            new(tm); /* simulate */
                                                                                                                            read(evnum);
                                                                                                                            FOR j := 1 TO evnum DO
                                                                                                                                BEGIN
                                                                                                                                    read(i,sta,tm,tt);
                                                                                                                                    new(tm); /* simulate */
                                                                                                                                    read(evnum);
                                                                                                                                    FOR j := 1 TO evnum DO
                                                                                                                                        BEGIN
                                                                                                                                            read(i,sta,tm,tt);
                                                                                                                                            new(tm); /* simulate */
                                                                                                                                            read(evnum);
                                                                                                                                            FOR j := 1 TO evnum DO
                                                                                                                                                BEGIN
                                                                                                                                                    read(i,sta,tm,tt);
                                                                                                                                                    new(tm); /* simulate */
                                                                                                                                                    read(evnum);
                                                                                                                                                    FOR j := 1 TO evnum DO
                                                                                                                                                                                                                    BEGIN
                                                                                                                                                                                                                    read(i,sta,tm,tt);
                                                                                                                                                                                                                    new(tm); /* simulate */
                                                                                                                                                                                                                    read(evnum);
                                                                                                                                                                                                                    FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
                                                                                                                                                                                                                                                                                        FOR j := 1 TO evnum DO
                                                                                                                                                                                                                        BEGIN
                                                                                                                                                                                                                                                                                        read(i,sta,tm,tt);
                                                                                                                                                                                                                                                                                        new(tm); /* simulate */
                                                                                                                                                                                                                                                                                        read(evnum);
PASCAL 8000/2.0B       AAEC (17FEB82)

1625  06C0 --       unit := false;
1626  06CC --       simulate
1627  06CC -0       END. (* end of main program *)

*AAEC PASCAL 2.0B COMPILATION CONCLUDED*

*NO ERRORS DETECTED IN PASCAL PROGRAM*