CANADIAN THESES ON MICROFICHE

THÈSES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

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.

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.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL 339 (r. 86/09)

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

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

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

FIGURES

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]

[17]. 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 (SiO_2), 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.

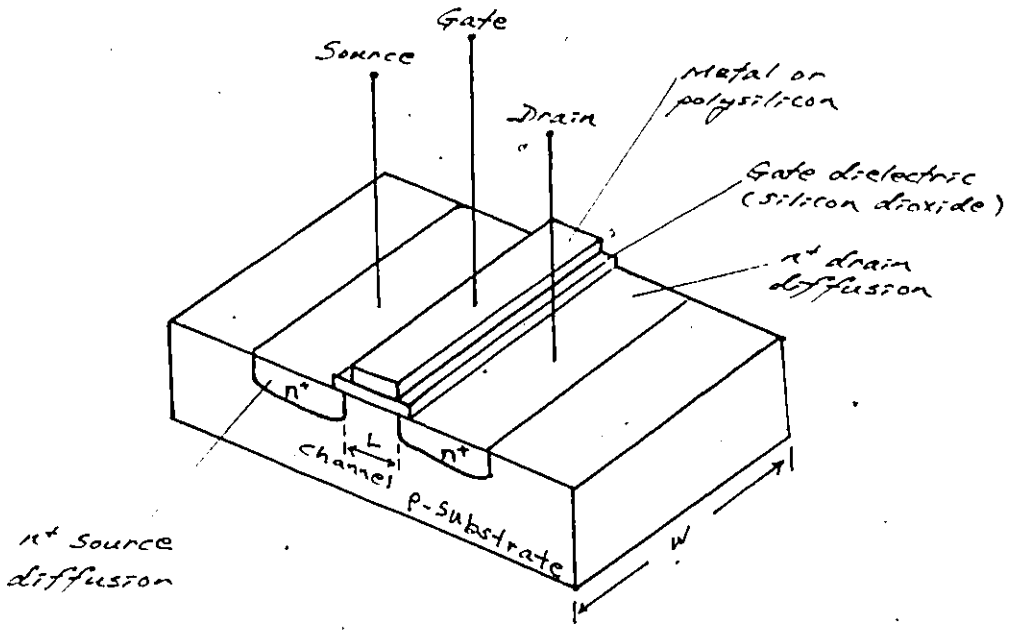


Figure 2.1: NMOS Transistor.

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 electic 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 text-books.

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_{TE} 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 devices 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} = 0V and the threshold voltage V_T is negative.

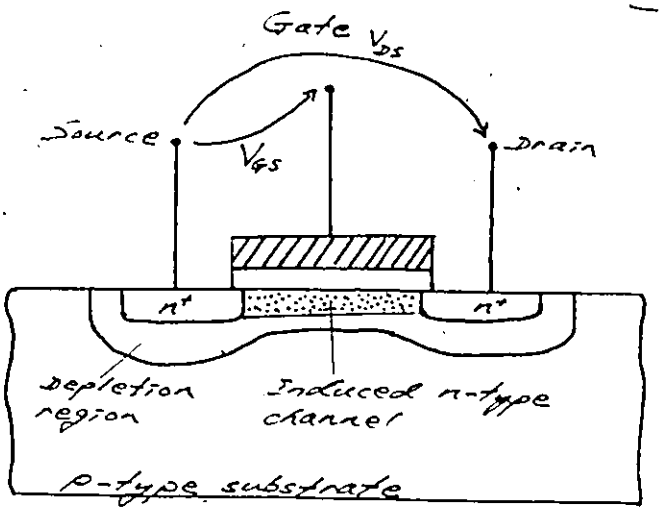
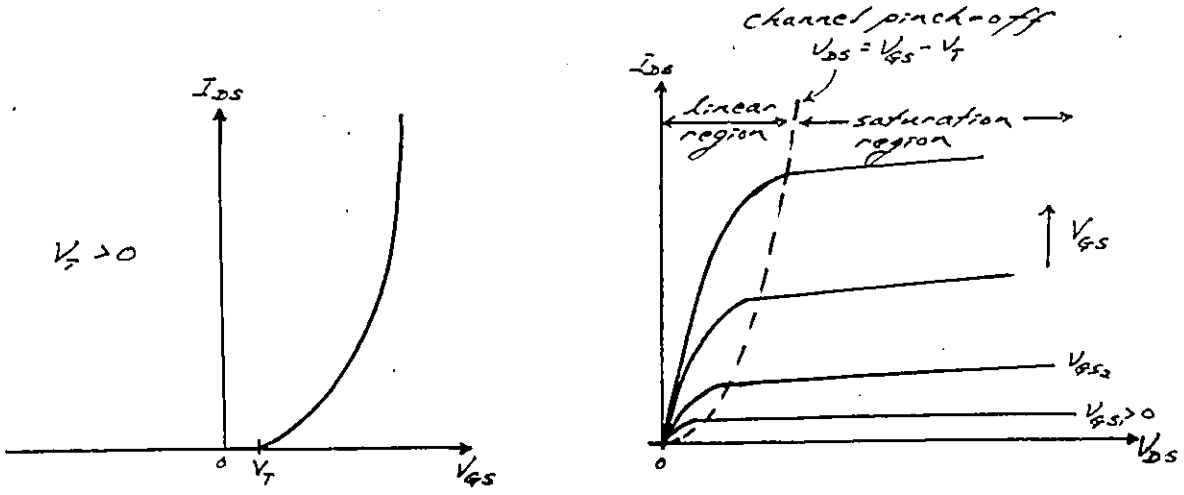


Figure 2.2: NMOS device cross-section showing depletion region and induced channel.

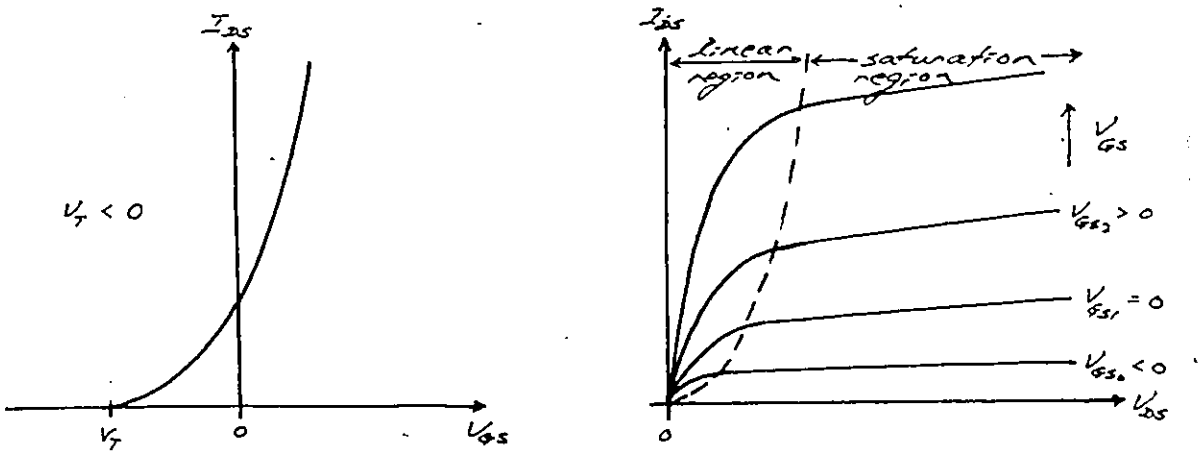
Either transistor type may operate in the linear region or in the saturation region. In the linear region there is a continuous channel between source and drain, and I_{DS} varies almost linearly with V_{DS} for a fixed V_{GS}. In the saturation region, the channel is pinched off at the drain when $V_{GS}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

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

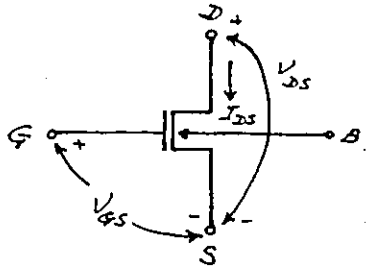
$$I_{DS} = \beta [2(V_{GS} - V_T) V_{DS} - V_{DS}^2] \quad \text{for} \quad 0 < V_{DS} \leq V_{GS} - V_T$$



(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

$$I_{DS} = \beta (V_{GS} - V_T)^2 \qquad \text{for} \quad 0 < V_{GS} - V_T \leq V_{DS}$$

The constant β is often referred to as the transistor gain factor, which is defined as

$$\beta = \frac{\mu_n C_{ox}}{2} \cdot \frac{W}{L}$$

where μ_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

$$I_{SD} = \beta [2(V_{SG} - |V_T|) V_{SD} - V_{SD}^2] \qquad \text{for} \quad 0 < V_{SD} \leq V_{SG} - |V_T|$$

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

$$I_{SD} = \beta (V_{SG} - |V_T|)^2 \qquad \text{for} \quad 0 < V_{SG} - |V_T| \leq V_{SD}$$

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_{db} 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

$$C_{j0} = \sqrt{\frac{q\,\varepsilon_{Si}\,N_A}{2\phi_0}}$$

and the built-in junction potential ϕ_0 is given by

$$\phi_0 = (kT/q) \cdot \ln(N_A N_D / n_i^2)$$

where k - Boltzmann's constant

T - absolute temperature

q - electronic charge

N_A, N_D - doping concentration

n_i - intrinsic carrier concentration of the silicon

ε_{si} - permittivity of the silicon

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

$$C_g = W \cdot L \cdot C_{ox} = W \cdot L \cdot \frac{\varepsilon_{ox}}{t_{ox}}$$

where W - channel width

L - channel length

ε_{ox} - permittivity of the gate dielectric

t_{ox} - thickness of the gate dielectric

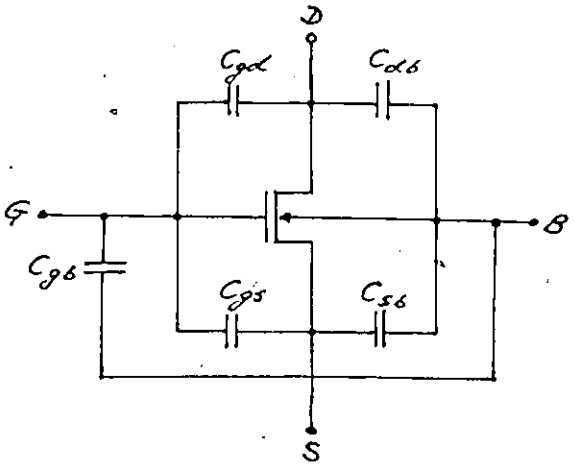


Figure 2.4: MOS transistor capacitances.

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.

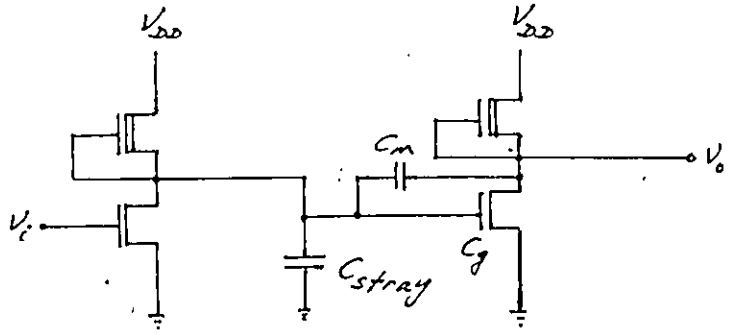
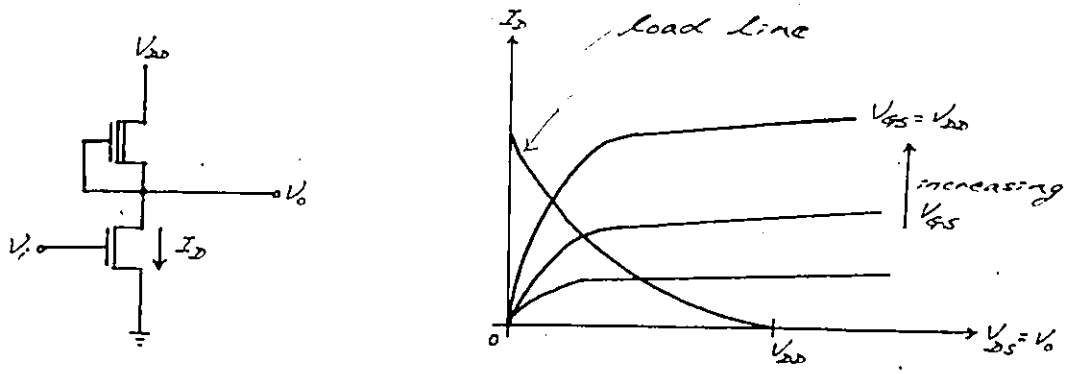


Figure 2.5: The Miller capacitance.

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 digitial 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 the ground

potential and thus has a logic value 0. When V_i 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}.

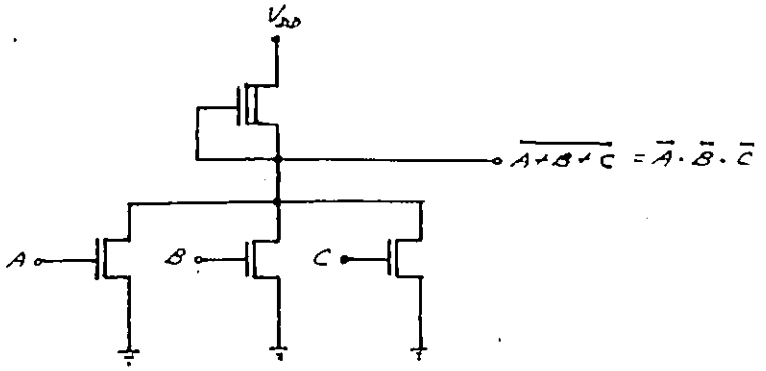


Figure 2.7: 3-input NOR gate.

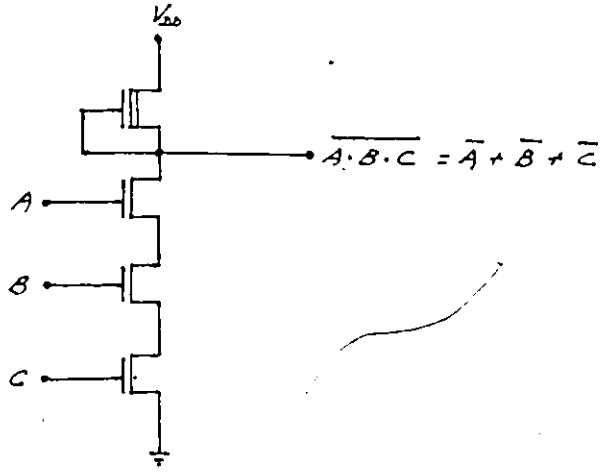


Figure 2.8: 3-input NAND gate.

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.

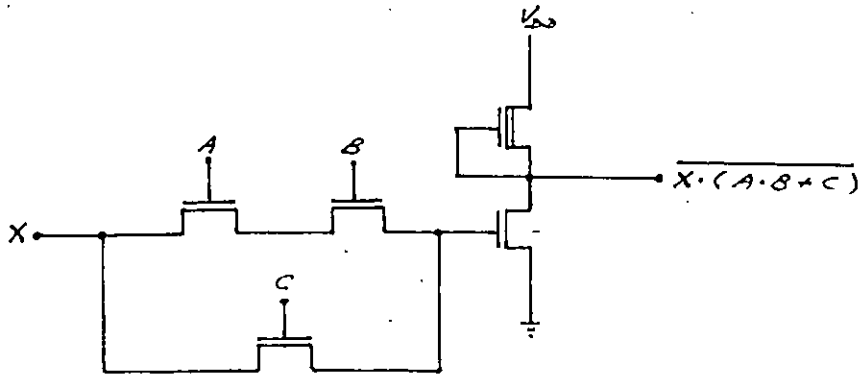


Figure 2.9: Steering logic circuit.

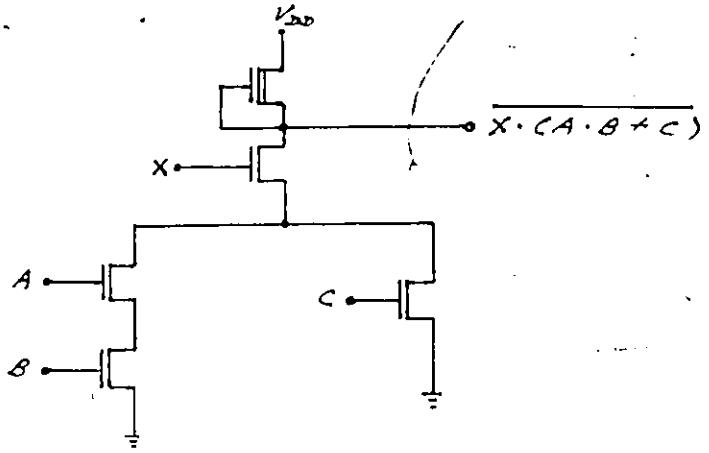


Figure 2.10: An equivalent circuit of Figure 2.9.

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 capcacitances.

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.

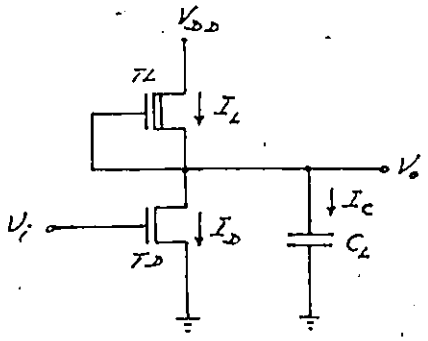


Figure 2.11: Transient model for an NMOS inverter.

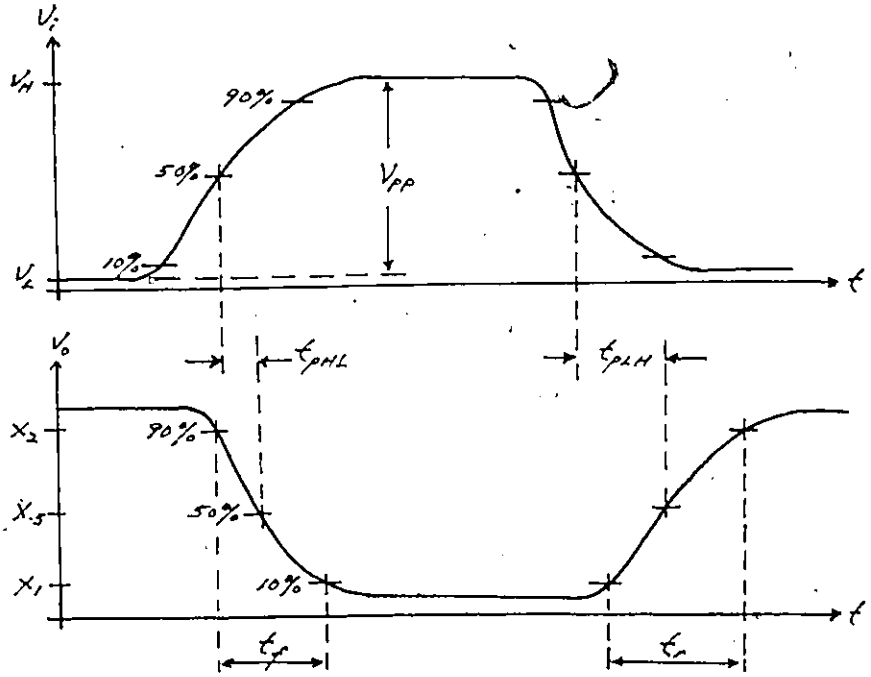
2.5.1 Definition of Transient Characteristics

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

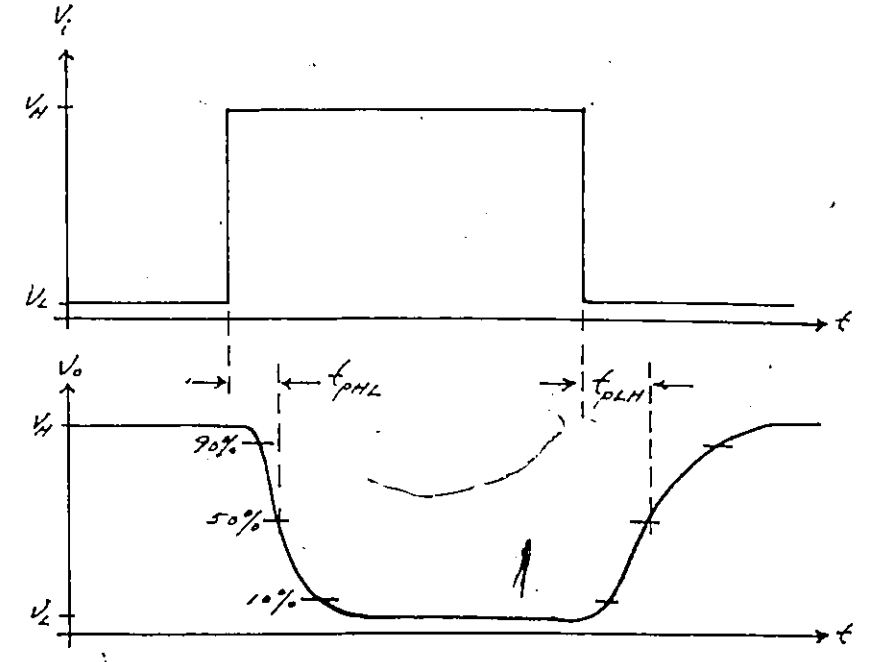
The waveforms at the input and output nodes are shown in Figure 2.12(a), where V_i is the input node voltage, V_o is the output node voltage, V_H, V_L and V_{pp} are the output high, output low levels and the magnitude of the voltage swing, respectively. Rise and fall times, t_r and t_f, are defined 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_{gb}, 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_{db} and C_{sb}, 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_{BB} of 0V.

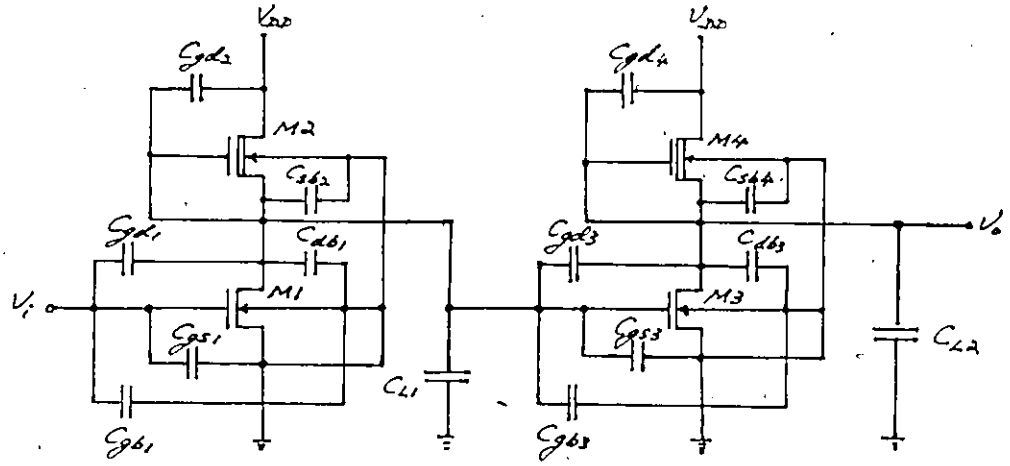
Figure 2.13 shows how the device capacitances in a circuit comprising two cascaded inverters can be lumped at the inverter output nodes. First, all capacitances across which there is no voltage change are ignored, since they have no effect on circuit performance. The device capacitances which must be 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:

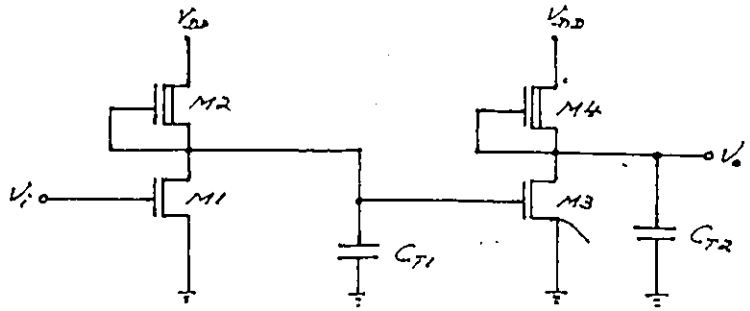
$$C_{T1} = K_{eq} (C_{db1} + C_{sb2}) + C_{L1} + C_{g3}$$

$$C_{T2} = K_{eq} (C_{db3} + C_{sb4}) + C_{L2}$$

The result of the linearization of junction capacitances using K_{eq} 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_D is assumed to be equal to zero and the currents I_L and I_c are given by:

a) When $V_{DD} - V_o = V_{DS} \geq V_{GS} - V_{TD}$ and $V_{GS} > V_{TD}$, the load transistor TL is in saturation region, and the currents are given by

$$I_L = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_L}{L_L} \cdot (V_{GS} - V_{TD})^2$$

$$I_C = C_L \cdot \frac{dV_o}{dt}$$

b) when $V_{DD} - V_o = V_{DS} \leq V_{GS} - V_{TD}$ and $V_{GS} > V_{TD}$, the load transistor TL is in linear region, the currents are given by

$$I_L = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_L}{L_L} \cdot [\, 2(V_{GS} - V_{TD})V_{DS} - V_{DS}^2 \,]$$

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

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:

$$t_r = \frac{2 C_L}{\mu_n C_{ox} W_L / L_L V_{DD}} \cdot \left[\frac{V_{DD}}{(-V_{TD})^2} (V_{DD} + V_{TD} - X_1) + \frac{V_{DD}/2}{(-V_{TD})} \cdot \ln \left(\frac{2(-V_{TD}) - (V_{DD} - X_2)}{(V_{DD} - X_2)} \right) \right] \quad (2.1)$$

$$t_{pLH} = \frac{2 C_L}{\mu_n C_{ox} W_L / L_L V_{DD}} \cdot \left[\frac{V_{DD}}{(-V_{TD})^2} (V_{DD} + V_{TD} - V_L) + \frac{V_{DD}/2}{(-V_{TD})} \cdot \ln \left(\frac{2(-V_{TD}) - (V_{DD} - X_{.5})}{(V_{DD} - X_{.5})} \right) \right] \quad (2.2)$$

where W_L - width of the load channel

L_L - length of the load channel

X_1, X_2 and $X_{.5}$ - 10%, 90% and 50% points of the voltage swing

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

$$t_r = 37.0 \times 10^3 \cdot (L_L/W_L) \cdot C_L \qquad (2.3)$$

$$t_{pLH} = 16.1 \times 10^3 \cdot (L_L/W_L) \cdot C_L \qquad (2.4)$$

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 $V_{DD} \geq V_o \geq V_{DD} - V_{TE}$, load transistor TL is in linear region and driver transistor TD is in saturation region, and the currents are given by:

$$I_L = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_L}{L_L} \cdot [\, 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \,]$$

$$I_D = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_D}{L_D} \cdot (V_{DD} - V_{TE})^2$$

$$I_C = C_L \cdot \frac{dV_o}{dt}$$

b)When $V_{DD} - V_{TE} \geq V_o \geq V_{DD} + V_{TD}$, both transistors are in linear region, and the currents are given by:

$$I_L = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_L}{L_L} \cdot [\, 2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \,]$$

$$I_D = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_d}{L_d} \cdot [\, 2(V_{DD} - V_{TE})V_o - V_o^2 \,]$$

$$I_C = C_L \cdot \frac{dV_o}{dt}$$

c)When $V_o \leq V_{DD} + V_{TD}$, load transistor TL is in saturation region and driver transistor TD is in linear region, and the currents are given by:

$$I_L = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_L}{L_L} \cdot (-V_{TD})^2$$

$$I_D = \frac{\mu_n C_{ox}}{2} \cdot \frac{W_d}{L_d} \cdot [\, 2(V_{DD} - V_{TE})V_o - V_o^2 \,]$$

$$I_C = C_L \cdot \frac{dV_o}{dt}$$

From the current equations given above, we can derive the fall time t_f and the fall delay t_{PHL}. Details are shown in Appendix B. The resultant equations are as follows:

$$t_f = \frac{2 C_L}{\mu_n C_{ox} W_L / L_L V_{DD}} \cdot \left[\frac{V_{DD}}{[k(V_{DD} - V_{TD})^2 - V_{TD}^2]^{1/2}} \cdot \left(\tan^{-1} \frac{V_{TE} + V_{TD}}{[k(V_{DD} - V_{TD})^2 - V_{TD}^2]^{1/2}} - \tan^{-1} \frac{V_{DD} - X_3 + V_{TD}}{[k(V_{DD} - V_{TD})^2 - V_{TD}^2]^{1/2}} \right) \right.$$

$$+ \frac{V_{DD}}{[b_2^2 - 4a_2c_2]^{1/2}} \cdot \left(\ln \left| \frac{2a_2(V_{DD} + V_{TD}) + b_2 - (b_2^2 - 4a_2c_2)^{1/2}}{2a_2(V_{DD} + V_{TD}) + b_2 + (b_2^2 - 4a_2c_2)^{1/2}} \right| \right.$$

$$\left. - \ln \left| \frac{2a_2(V_{DD} - V_{TE}) + b_2 - (b_2^2 - 4a_2c_2)^{1/2}}{2a_2(V_{DD} - V_{TE}) + b_2 + (b_2^2 - 4a_2c_2)^{1/2}} \right| \right)$$

$$+ \frac{V_{DD}}{[b_3^2 - 4a_3c_3]^{1/2}} \cdot \left(\ln \left| \frac{2a_3X_1 + b_3 - (b_3^2 - 4a_3c_3)^{1/2}}{2a_3X_1 + b_3 + (b_3^2 - 4a_3c_3)^{1/2}} \right| \right.$$

$$\left. \left. - \ln \left| \frac{2a_3(V_{DD} + V_{TD}) + b_3 - (b_3^2 - 4a_3c_3)^{1/2}}{2a_3(V_{DD} + V_{TD}) + b_3 + (b_3^2 - 4a_3c_3)^{1/2}} \right| \right) \right] \quad (2.5)$$

$$t_{PHL} = \frac{2C_L}{\mu_n C_{ox} W_d/L_d V_{DD}} \cdot \left[\frac{V_{DD}}{\left[k(V_{DD}-V_{TD})^2 - V_{TD}^2\right]^{1/2}} \cdot \left(\tan^{-1} \frac{V_{TE}+V_{TD}}{\left[k(V_{DD}-V_{TD})^2 - V_{TD}^2\right]^{1/2}} \right.\right.$$

$$\left. - \tan^{-1} \frac{V_{DD}-V_M+V_{TD}}{\left[k(V_{DD}-V_{TD})^2 - V_{TD}^2\right]^{1/2}} \right)$$

$$+ \frac{V_{DD}}{\left[b_2^2 - 4a_2c_2\right]^{1/2}} \cdot \left(\ln \left| \frac{2a_2 X.5 + b_2 - (b_2^2 - 4a_2c_2)^{1/2}}{2a_2 X.5 + b_2 + (b_2^2 - 4a_2c_2)^{1/2}} \right| \right.$$

$$\left.\left. - \ln \left| \frac{2a_2(V_{DD}-V_{TE}) + b_2 - (b_2^2 - 4a_2c_2)^{1/2}}{2a_2(V_{DD}-V_{TE}) + b_2 + (b_2^2 - 4a_2c_2)^{1/2}} \right| \right) \right] \qquad (2.6)$$

and

$$a_2 = (k-1)$$

$$b_2 = 2V_{TD} + 2V_{DD} - 2k(V_{DD} - V_{TE})$$

$$c_2 = -2V_{TD}\,V_{DD} - V_{DD}^2$$

$$a_3 = k$$

$$b_3 = -2k(V_{DD} - V_{TE})$$

$$c_3 = (-V_{TD})^2$$

where

$$k = \frac{W_d/L_d}{W_l/L_l}$$

In a typical process, where $V_{TE} = 0.2V_{DD}$ and $V_{TD} = -0.8V_{DD}$, the above equations become as follows:

i) for k = 4,

$$t_f = 4.0624 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \cdot \left(\frac{L_d}{W_d}\right) \cdot C_L \qquad (2.7)$$

$$t_{PHL} = 1.6197 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \cdot \left(\frac{L_d}{W_d}\right) \cdot C_L \qquad (2.8)$$

ii) for k = 8,

$$t_f = 3.8758 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \cdot \left(\frac{L_d}{W_d}\right) \cdot C_L \qquad (2.9)$$

$$t_{PHL} = 1.6209 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \cdot \left(\frac{L_d}{W_d}\right) \cdot C_L \qquad (2.10)$$

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

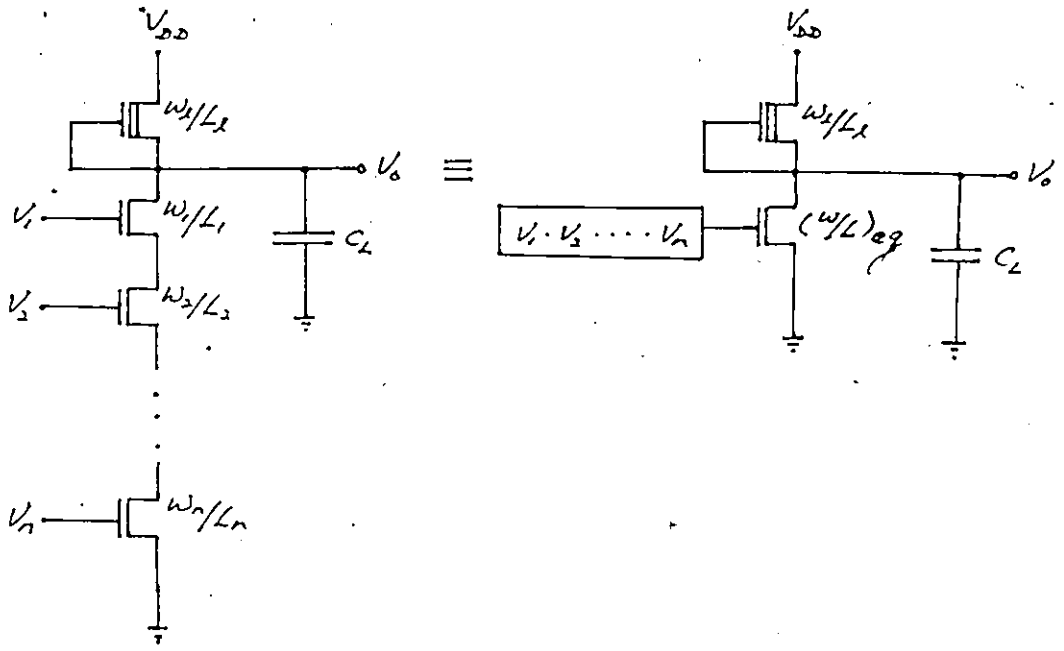
In the previous section, we have derived all the delay time equations for an NMOS inverter. Here, we are going to extend the concepts to any multiple inputs gate such as NAND, NOR and any combination of series and parallel driver transistors. This is done by using the following steps.

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.

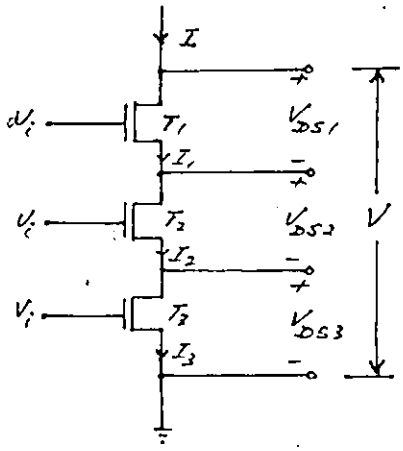
For several devices connected in series whose geometric aspect ratios are W_1/L_1, W_2/L_2, ..., W_n/L_n,

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



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

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



$$V = V_{DS1} + V_{DS2} + V_{DS3}$$

$$I_1 = \mu_n C_{ox} \cdot \left(\frac{W_1}{L_1}\right) \cdot \left[(V_i - V_{DS2} - V_{DS3} - V_{TE}) V_{DS1} - \frac{V_{DS1}^2}{2} \right]$$

$$I_2 = \mu_n \cdot C_{ox} \left(\frac{W_2}{L_2}\right) \cdot \left[(V_i - V_{DS3} - V_{TE}) V_{DS2} - \frac{V_{DS2}^2}{2} \right]$$

$$I_3 = \mu_n C_{ox} \left(\frac{W_3}{L_3}\right) \cdot \left[(V_i - V_{TE}) V_{DS3} - \frac{V_{DS3}^2}{2} \right]$$

Since $I = I_1 = I_2 = I_3$,

$$\left(\frac{L_1}{W_1}\right) I = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V_{DS1} - V_{DS1} \cdot V_{DS2} - V_{DS1} \cdot V_{DS3} - \frac{V_{DS1}^2}{2} \right]$$

$$\left(\frac{L_2}{W_2}\right) I = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V_{DS2} - V_{DS2} \cdot V_{DS3} - \frac{V_{DS2}^2}{2} \right]$$

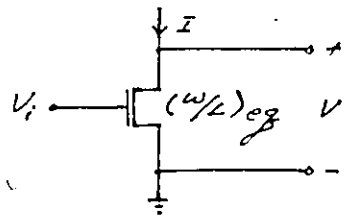
$$\left(\frac{L_3}{W_3}\right) I = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V_{DS3} - \frac{V_{DS3}^2}{2} \right]$$

Adding up the three equations, we have

$$\left(\frac{L_1}{W_1} + \frac{L_2}{W_2} + \frac{L_3}{W_3}\right) I = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) (V_{DS1} + V_{DS2} + V_{DS3}) - \frac{1}{2} \cdot (V_{DS1}^2 + V_{DS2}^2 + V_{DS3}^2 + 2V_{DS1} \cdot V_{DS2} + 2V_{DS1} V_{DS3} + 2V_{DS2} V_{DS3}) \right]$$

$$= \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V - \frac{V^2}{2} \right]$$

$$I = \mu_n C_{ox} \cdot \left(\frac{1}{L_1/W_1 + L_2/W_2 + L_3/W_3} \right) \cdot \left[(V_i - V_{TE}) V - \frac{V^2}{2} \right]$$



$$I = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right)_{eq} \cdot \left[(V_i - V_{TE}) V - \frac{V^2}{2} \right]$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{L_1/W_1 + L_2/W_2 + L_3/W_3}$$

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

$$I_1 = \mu_n C_{ox} \cdot \left(\frac{W_1}{L_1}\right) \frac{1}{2} (V_i - V_{TE} - V_{DS2} - V_{DS3})^2$$

$$I_2 = \mu_n C_{ox} \cdot \left(\frac{W_2}{L_2}\right) \cdot \left[(V_i - V_{TE} - V_{DS3}) V_{DS2} - \frac{V_{DS2}^2}{2}\right]$$

$$I_3 = \mu_n C_{ox} \cdot \left(\frac{W_3}{L_3}\right) \cdot \left[(V_i - V_{TE}) V_{DS3} - \frac{V_{DS3}^2}{2}\right]$$

$$\left(\frac{L_1}{W_1}\right) I_1 = \mu_n C_{ox} \cdot \left[\frac{1}{2}(V_i - V_{TE})^2 + \frac{1}{2} V_{DS2}^2 + \frac{1}{2} V_{DS3}^2 - (V_i - V_{TE}) V_{DS2} - (V_i - V_{TE}) V_{DS3} + V_{DS2} V_{DS3}\right]$$

$$\left(\frac{L_2}{W_2}\right) I_2 = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V_{DS2} - V_{DS2} V_{DS3} - \frac{V_{DS2}^2}{2}\right]$$

$$\left(\frac{L_3}{W_3}\right) I_3 = \mu_n C_{ox} \cdot \left[(V_i - V_{TE}) V_{DS3} - \frac{V_{DS3}^2}{2}\right]$$

Adding up the three equations, we have

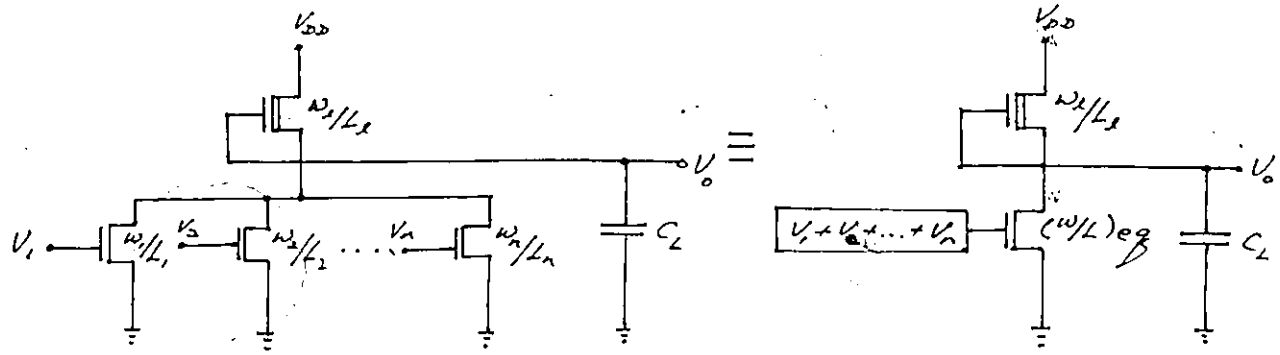
$$\left(\frac{L_1}{W_1} + \frac{L_2}{W_2} + \frac{L_3}{W_3}\right) I = \mu_n C_{ox} \frac{1}{2} (V_i - V_{TE})^2$$

$$I = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right)_{eq} \cdot \frac{1}{2} (V_i - V_{TE})^2$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{L_1/W_1 + L_2/W_2 + L_3/W_3}$$

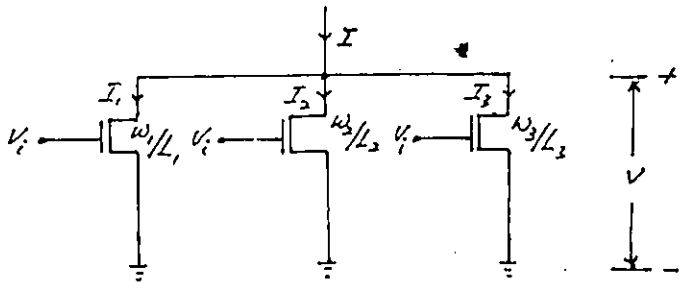
For several devices connected in parallel whose geometric aspect ratios are W/L , W/L , ... , W/L ,

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



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

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



$$I_1 = \mu_n C_{ox} \cdot \left(\frac{W_1}{L_1}\right) \cdot \left[(V_i - V_{TE})V - \frac{V^2}{2} \right]$$

$$I_2 = \mu_n C_{ox} \cdot \left(\frac{W_2}{L_2}\right) \cdot \left[(V_i - V_{TE})V - \frac{V^2}{2} \right]$$

$$I_3 = \mu_n C_{ox} \cdot \left(\frac{W_3}{L_3}\right) \cdot \left[(V_i - V_{TE})V - \frac{V^2}{2} \right]$$

Since $I = I_1 + I_2 + I_3$, adding up the equations, we get

$$I = I_1 + I_2 + I_3$$

$$= \mu_n C_{ox} \cdot \left(\frac{W_1}{L_1} + \frac{W_2}{L_2} + \frac{W_3}{L_3}\right) \cdot \left[(V_i - V_{TE})V - \frac{V^2}{2} \right]$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{W_1}{L_1} + \frac{W_2}{L_2} + \frac{W_3}{L_3}$$

Case 2: When $V \geq V_i - V_{TE} > 0$, all transistors operate in the saturation region.

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

$$I_2 = \mu_n C_{ox} \cdot \left(\frac{W_2}{L_2}\right) \frac{1}{2} (V_i - V_{TE})^2$$

$$I_3 = \mu_n C_{ox} \cdot \left(\frac{W_3}{L_3}\right) \frac{1}{2} (V_i - V_{TE})^2$$

Adding up the three equations, we get

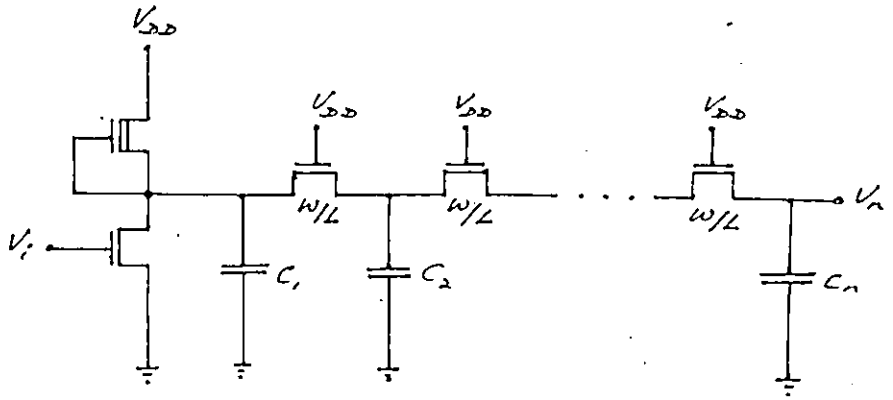
$$I = I_1 + I_2 + I_3$$

$$= \mu_n C_{ox} \cdot \left(\frac{W_1}{L_1} + \frac{W_2}{L_2} + \frac{W_3}{L_3}\right) \frac{1}{2} (V_i - V_{TE})^2$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{W_1}{L_1} + \frac{W_2}{L_2} + \frac{W_3}{L_3}$$

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:

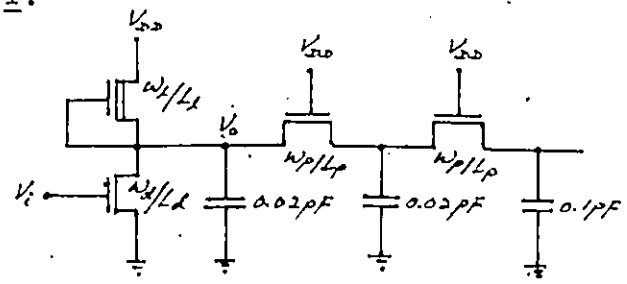
$$C_{eq} = \left(C_1 + \frac{1}{\sqrt{2}} C_2 + \frac{1}{\sqrt{3}} C_3 + \ldots + \frac{1}{\sqrt{n}} C_n \right)$$



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:

$W_1/L_1 = 5/10$, $W_2/L_2 = 10/5$, $W_p/L_p = 5/5$ (all in μm)

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

The equivalent load capacitance,

$$C_{eq} = (0.02 + \frac{1}{\sqrt{2}} \times 0.02 + \frac{1}{\sqrt{3}} \times 0.1) = 0.09 \text{ pF}$$

From eqs. (2.3) and (2.4),

$$t_r = 37 \times 10^3 \cdot (\frac{10}{5}) \times 0.09 \times 10^{-12} = 6.7 \text{ ns}$$

$$t_{pLH} = 16.1 \times 10^3 \cdot (\frac{10}{5}) \times 0.09 \times 10^{-12} = 2.9 \text{ ns}$$

From eqs. (2.7) and (2.8),

$$t_f = 40.624 \times 10^3 \cdot (\frac{5}{10}) \times 0.09 \times 10^{-12} = 1.8 \text{ ns}$$

$$t_{pHL} = 16.197 \times 10^3 \cdot (\frac{5}{10}) \times 0.09 \times 10^{-12} = 0.7 \text{ ns}$$

(ns)	Approximate Results	SPICE Results
t_r	6.7	7.0
t_f	1.8	1.2
t_{pLH}	2.9	1.3
t_{pHL}	0.7	0.3

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 parasite 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:

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

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

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

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

where V_f - final value at output node

L_p - channel length of pass transistor

W_p - channel width of pass transistor

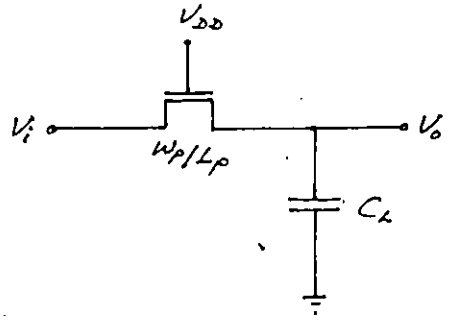


Figure 2.14: A pass transistor.

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

$$C_{eq} = \frac{1}{N\sqrt{n}} \cdot (C_1 + \sqrt{2}\, C_2 + \cdots + \sqrt{n}\, C_n)$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{n} \cdot \left(\frac{W_p}{L_p}\right)$$

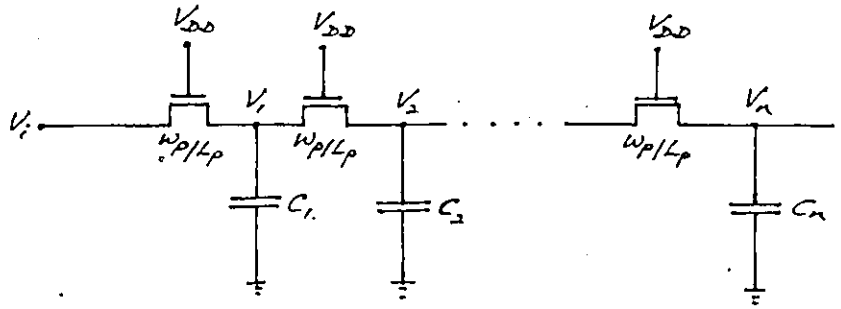
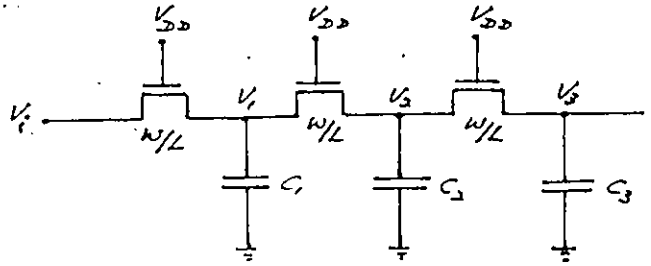
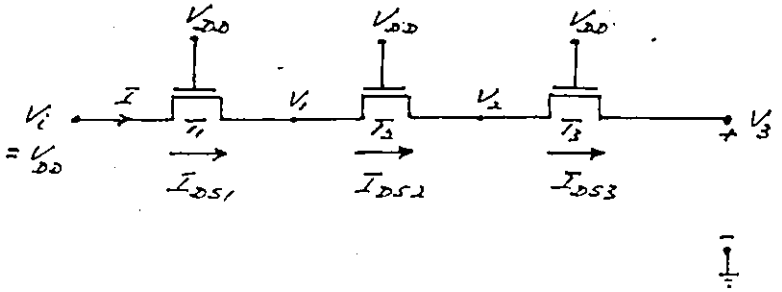


Figure 2.15: Pass transistor chain.

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



Static analysis



T_1 saturates, T_2 and T_3 operate in linear region.

$$I_{DS1} = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right) \frac{1}{2} (V_{DD} - V_1 - V_{TE})^2$$

$$I_{DS2} = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right) \frac{1}{2} [(V_{DD} - V_2 - V_{TE})^2 - (V_{DD} - V_1 - V_{TE})^2]$$

$$I_{DS3} = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right) \frac{1}{2} [(V_{DD} - V_3 - V_{TE})^2 - (V_{DD} - V_2 - V_{TE})^2]$$

Adding up the three equations, we have

$$3I = \mu_n C_{ox} \cdot \left(\frac{W}{L}\right) \frac{1}{2} (V_{DD} - V_3 - V_{TE})^2$$

Thus,

$$I = \frac{1}{3} \left[\mu_n C_{ox} \cdot \left(\frac{W}{L}\right) \frac{1}{2} (V_{DD} - V_3 - V_{TE})^2\right]$$

Then,

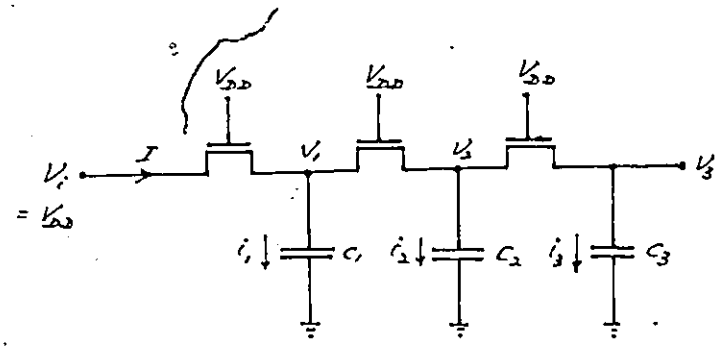
$$(V_{DD} - V_1 - V_{TE})^2 = \frac{1}{3} \cdot (V_{DD} - V_3 - V_{TE})^2$$

$$(V_{DD} - V_2 - V_{TE})^2 = \frac{2}{3} \cdot (V_{DD} - V_3 - V_{TE})^2$$

$$V_1 = \frac{1}{\sqrt{3}} \cdot V_3 + \left(1 - \frac{1}{\sqrt{3}}\right) \cdot (V_{DD} - V_{TE})$$

$$V_2 = \frac{\sqrt{2}}{\sqrt{3}} \cdot V_3 + \left(1 - \frac{\sqrt{2}}{\sqrt{3}}\right) \cdot (V_{DD} - V_{TE})$$

Adding capacitors to the circuit used for static analysis



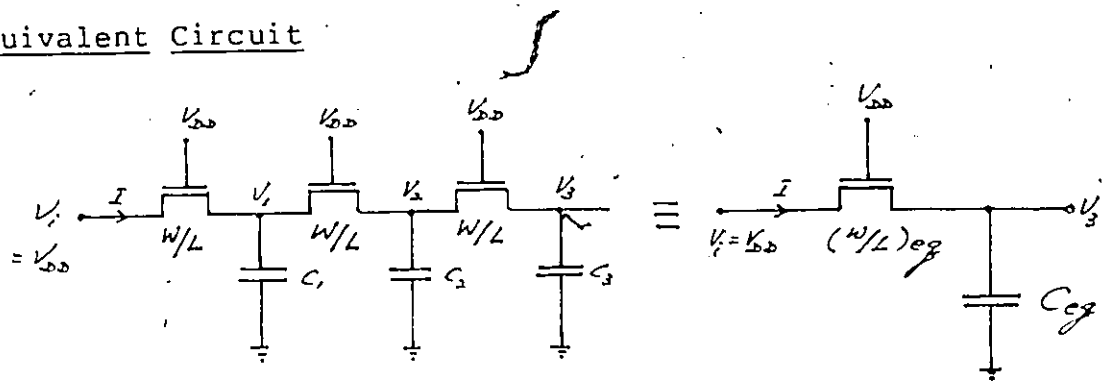
$$i_1 = C_1 \frac{dV_1}{dt} = C_1 \frac{1}{\sqrt{3}} \frac{dV_3}{dt}$$

$$i_2 = C_2 \frac{dV_2}{dt} = C_2 \frac{\sqrt{2}}{\sqrt{3}} \cdot \frac{dV_3}{dt}$$

$$i_3 = C_3 \frac{dV_3}{dt} = C_3 \frac{\sqrt{3}}{\sqrt{3}} \cdot \frac{dV_3}{dt}$$

$$I = i_1 + i_2 + i_3 = \frac{1}{\sqrt{3}} \cdot (C_1 + \sqrt{2} C_2 + \sqrt{3} C_3) \cdot \frac{dV_3}{dt}$$

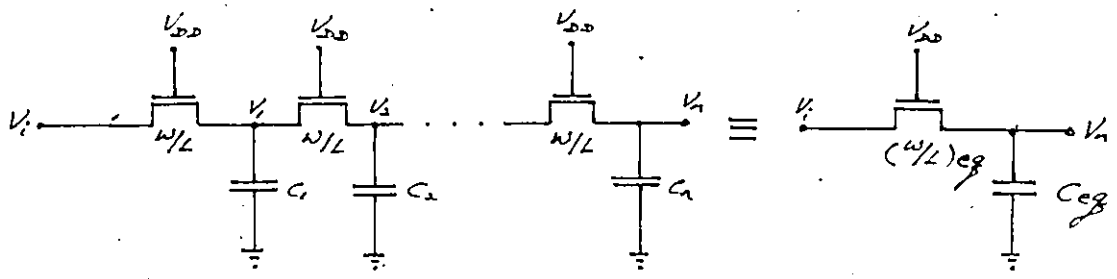
Equivalent Circuit



$$C_{eq} = \frac{1}{\sqrt{3}} \cdot (C_1 + \sqrt{2} C_2 + \sqrt{3} C_3)$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{3} \left(\frac{W}{L}\right)$$

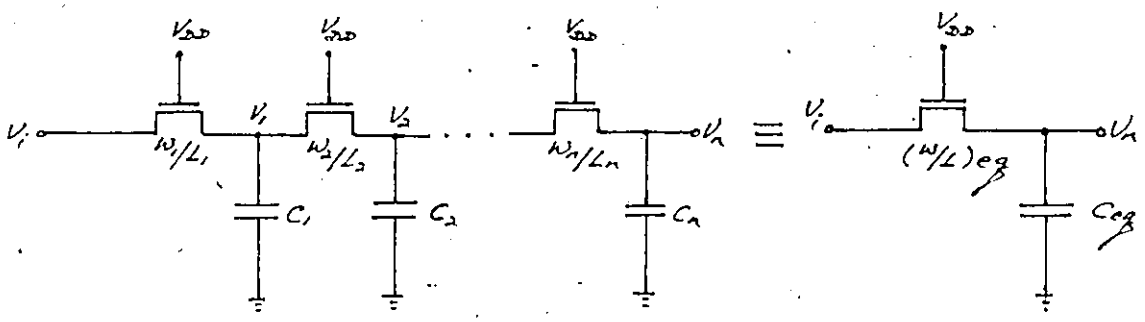
Generalization



$$C_{eq} = \frac{1}{\sqrt{n}} \cdot (C_1 + \sqrt{2}\, C_2 + \sqrt{3}\, C_3 + \cdots + \sqrt{n}\, C_n)$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{n} \cdot \left(\frac{W}{L}\right)$$

More on Generalization

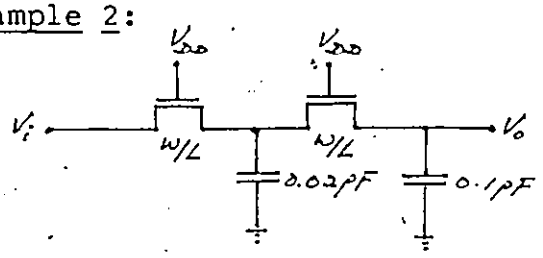


$$C_{eq} = \frac{1}{\sqrt{\sum_{i=1}^{n} \frac{L_i}{W_i}}} \cdot \left(\sum_{k=1}^{n} \sqrt{\sum_{j=1}^{k} \frac{L_j}{W_j}} \cdot C_k \right)$$

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{\sum_{i=1}^{n} \frac{L_i}{W_i}}$$

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:

$V_{DD} = 5V$, $\mu_n C_{ox} = 2 \times 10^{-5}$ A/V^2, W/L = 5/5 (in μm)

$V_{TE} = 0.2V_{DD}$

So, $V_f = V_{DD} - V_{TE} = 4V$.

$$\left(\frac{W}{L}\right)_{eq} = \frac{1}{2} \cdot \left(\frac{5}{5}\right) = \frac{1}{2}$$

$$C_{eq} = \frac{1}{\sqrt{2}} \cdot (0.02 + \sqrt{2} \times 0.1) = 0.1141 \text{ pF}$$

$$t_r = 17.778 \left(\frac{1}{2 \times 10^{-5} \times 4}\right)(2)(0.1141 \times 10^{-12}) = 50.7 \text{ ns}$$

$$t_{pLH} = 2 \left(\frac{1}{2 \times 10^{-5} \times 4}\right)(2)(0.1141 \times 10^{-12}) = 5.7 \text{ ns}$$

$$t_f = 2.7438 \left(\frac{1}{2 \times 10^{-5} \times 4}\right)(2)(0.1141 \times 10^{-12}) = 7.8 \text{ ns}$$

$$t_{pHL} = 1.0986 \left(\frac{1}{2 \times 10^{-5} \times 4}\right)(2)(0.1141 \times 10^{-12}) = 3.1 \text{ ns}$$

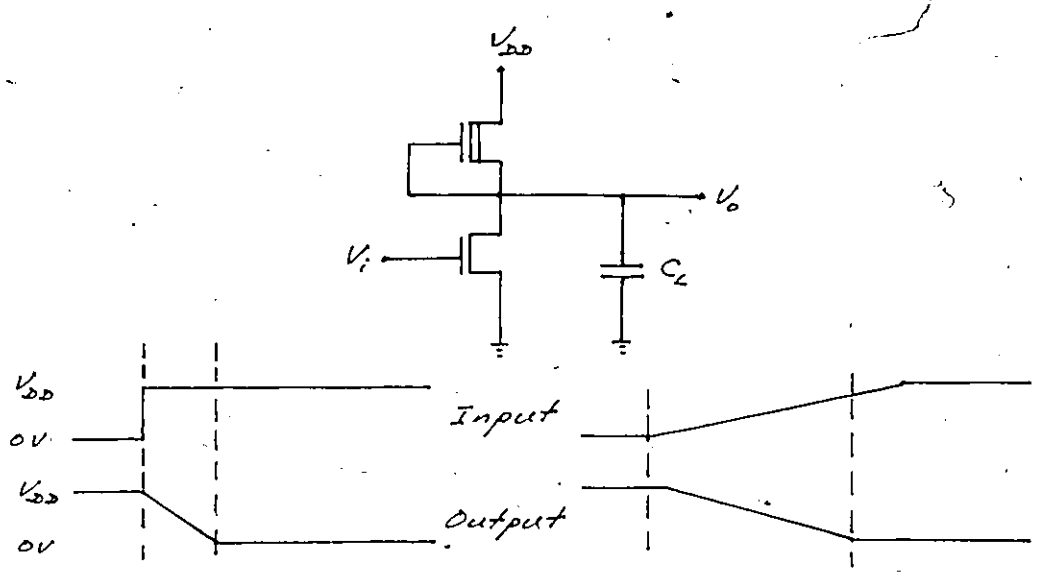
(ns)	Approximate Results	SPICE Results	% error
t_r	50.7	47.5	7
t_f	7.8	7.1	10
t_{pLH}	5.7	5.8	-2
t_{pHL}	3.1	3.3	-6

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_{DD} 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/



(a) Step function input (b) Linear ramp input

Figure 2.16: An inverter driven by different input waveforms.

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

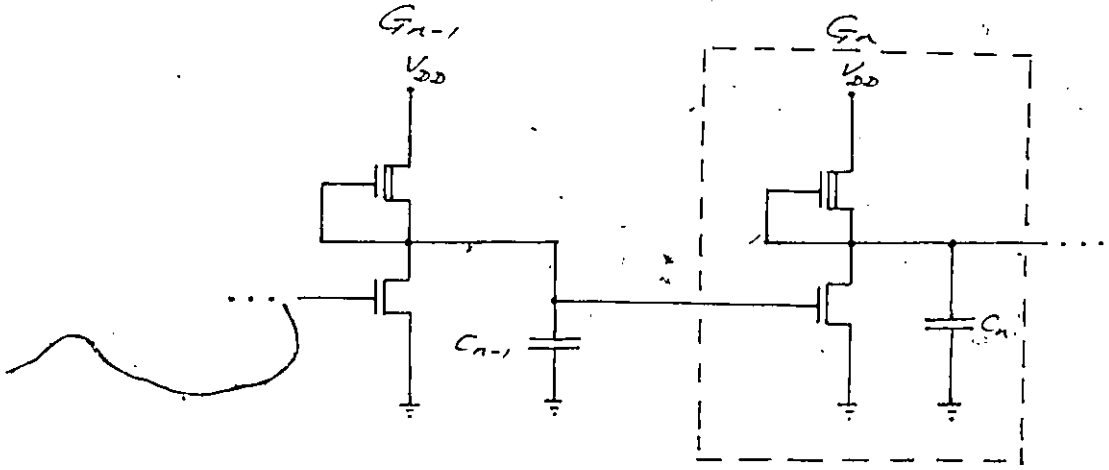


Figure 2.17: NMOS inverter chain.

the load capacitance C_n but less on the input capacitance. The fall delay t_{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.

$$t_r = t_r^* \left(1 + 0.3 \left(t_{fI} / t_r^* \right)^{0.5} \right)$$

$$t_f = t_f^* \left(1 + 0.25 \left(t_{rI} / t_f^* \right)^{0.7} \right)$$

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

$$t_{PHL} = t_f \; / \; [\; 1.9^2 + 0.06 (C_I / C_L)^2 \;]^{1/2}$$

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.

C_{n-1}(pF)	t_r (ns)	t_f (ns)
0.02	1.5	0.4
0.1	7.4	2.0
0.2	14.8	4.1
0.4	29.6	8.1
1.0	74.0	20.3
2.0	147.9	40.6

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_{DD} = 5V, $\mu_n C_{ox}$ = 2×10^{-5} A/V^2

W_ℓ / L_ℓ = 5/10, W_α / L_α = 10/5 (all in μm)

C_n = 0.2 pF

So,

$$t_r^* = 3.6976 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \left(\frac{L_\ell}{W_\ell}\right) \cdot C_n = 14.8 \text{ ns}$$

$$t_f^* = 4.0624 \left(\frac{1}{\mu_n C_{ox} V_{DD}}\right) \cdot \left(\frac{L_\alpha}{W_\alpha}\right) \cdot C_n = 4.1 \text{ ns}$$

and

$$t_r = 14.8 \left[1 + 0.3 \left(t_{fI} / 14.8 \right)^{1.5} \right]$$

$$t_{PLH} = t_r \Big/ \left[2.9^2 + 0.15 (C_{n-1}/0.2)^2 \right]^{1/2}$$

$$t_f = 4.1 \left[1 + 0.25 (t_{rI}/4.1)^{1.1} \right]$$

$$t_{PHL} = t_f \Big/ \left[1.9^2 + 0.06 (C_{n-1}/0.2)^2 \right]^{1/2}$$

The results are given in the following table and plotted in Figure 2.18.

C_{n-1}/C_n	t_r (ns)	t_f (ns)	t_{pLH} (ns)	t_{pHL} (ns)
0.02/0.2	14.8	4.4	5.1	2.3
0.1/0.2	15.0	6.0	5.2	3.2
0.2/0.2	15.4	8.3	5.3	4.3
0.4/0.2	16.6	13.1	5.5	6.7
1.0/0.2	21.9	28.8	6.3	12.7
2.0/0.2	35.0	57.0	7.2	18.4

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

C_{n-1}/C_n	t_r (ns)	t_f (ns)	t_{pLH} (ns)	t_{pHL} (ns)
0.02/0.2	14.9	4.3	5.1	2.3
0.1/0.2	15.5	6.4	5.1	3.9
0.2/0.2	16.3	9.0	5.1	5.4
0.4/0.2	18.3	14.0	5.3	7.0
1.0/0.2	24.0	28.0	6.0	11.5
2.0/0.2	35.0	50.5	6.8	17.0

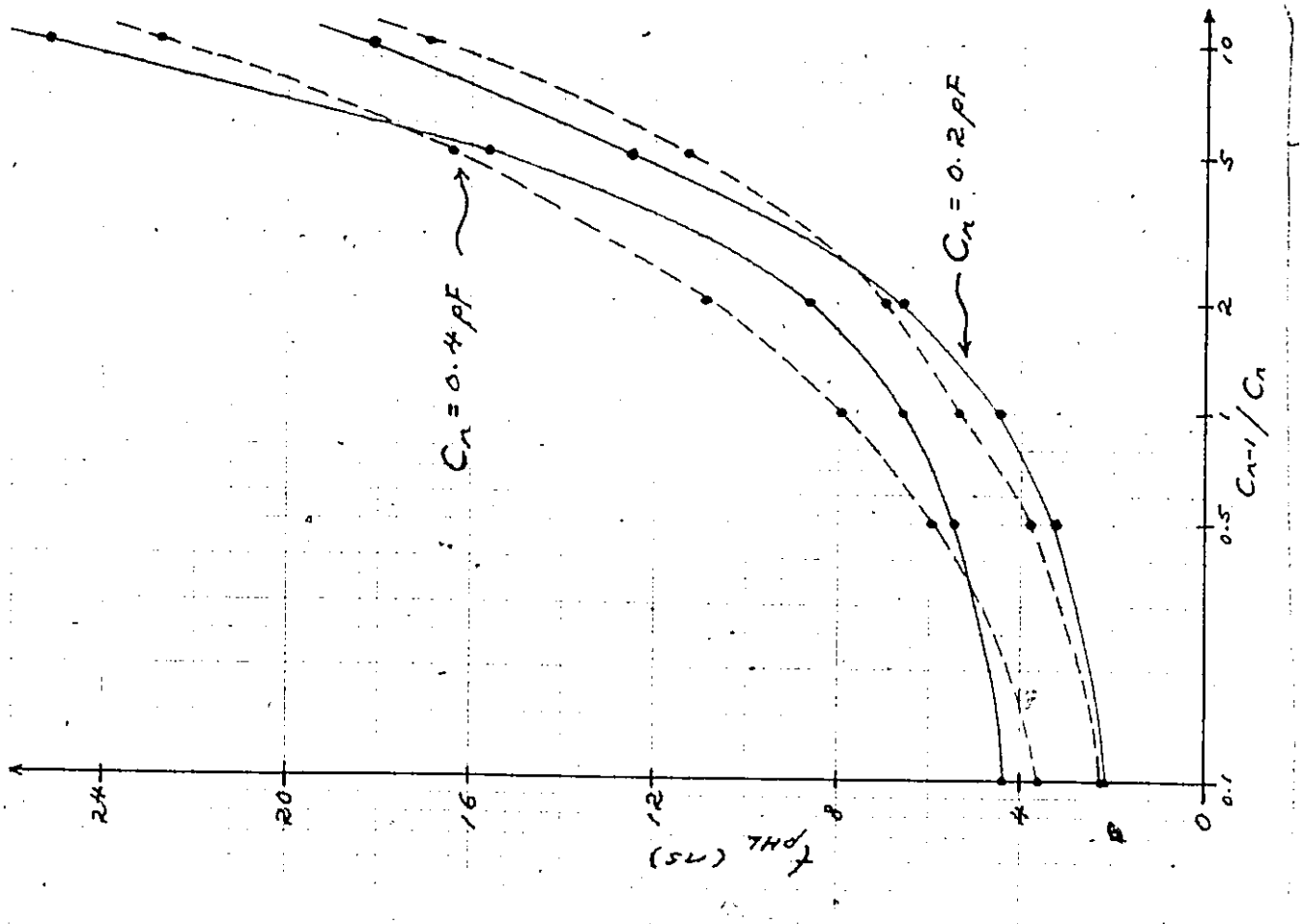
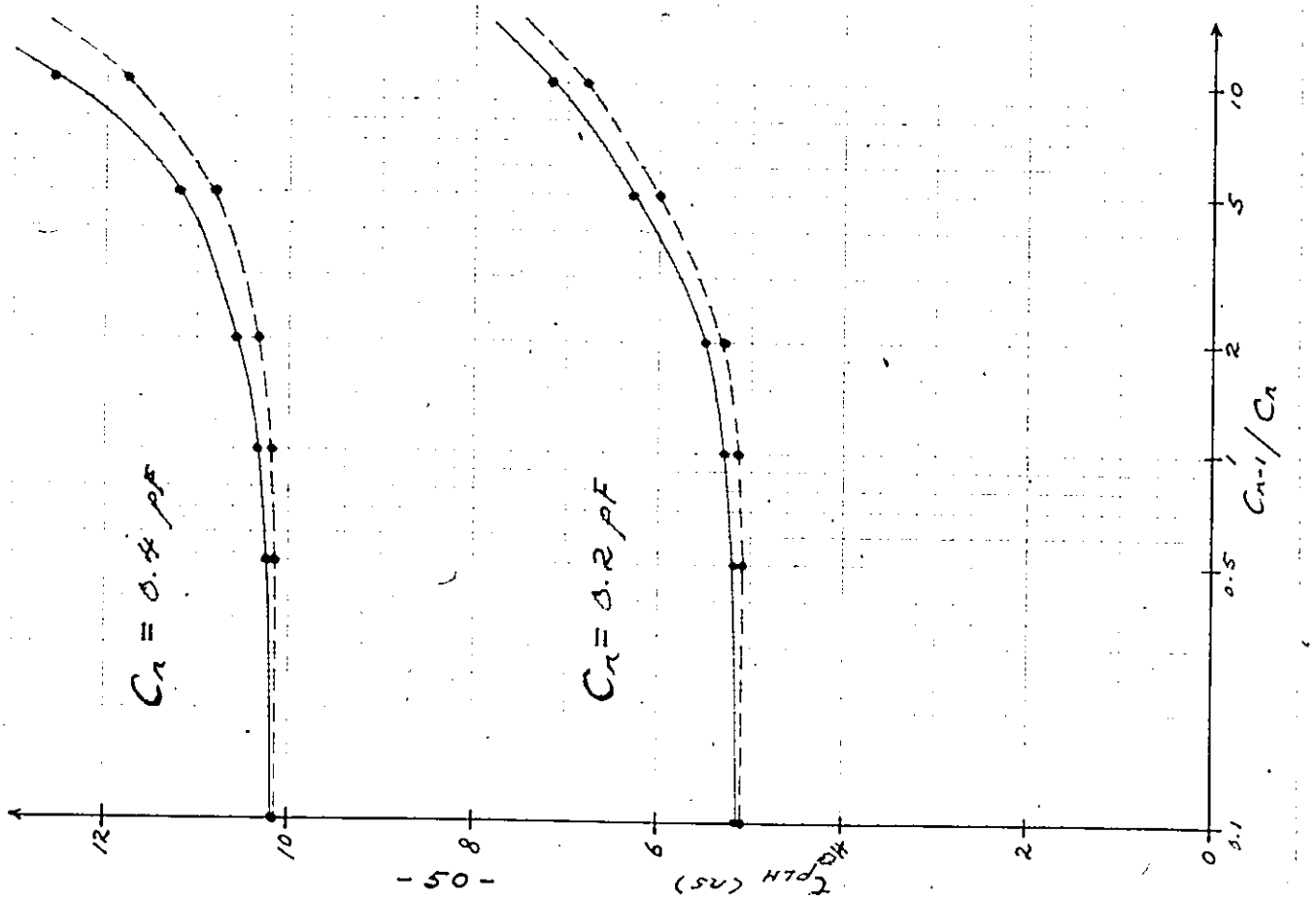


Figure 2.18: Delay times t_{pLH} and t_{pHL} for various capacitive load conditions. The dotted lines are the results obtained by circuit simulation.



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:

$$t_r = t_r^* [\ 1 + 0.3\ (t_{rI}/t_r^*)^{1.5}\ ]$$

$$t_{pLH} = t_r\ /8.9$$

$$t_f = t_f^*\ [\ 1 + (t_{fI}/t_f^*)^{0.7}\ ],$$

$$t_{pHL} = t_f\ /2.5$$

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

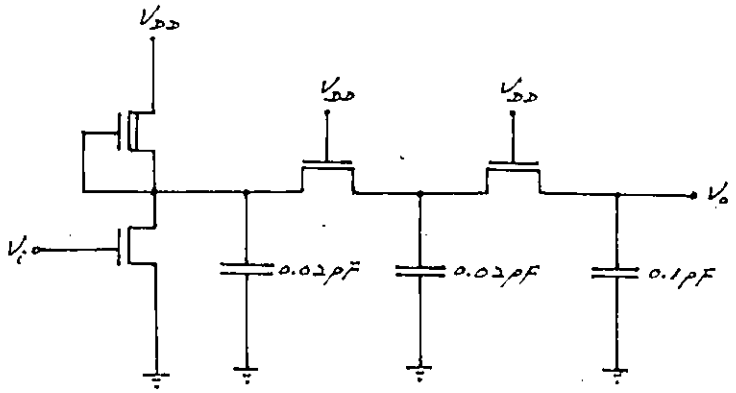


Figure 2.19: A pass transistor chain driven by an inverter.

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:

$V_{DD} = 5V$, $\mu_n C_{ox} = 2 \times 10^{-5}$ A/V^2

$W_L/L_L = 5/10$, $W_\alpha/L_\alpha = 10/5$, $W_P/L_P = 5/5$ (in μm)

$V_{TD} = -0.8V_{DD}$, $V_{TE} = 0.2V_{DD}$

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:

$t_r = 6.7$ ns

$t_f = 1.8$ 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:

$t_r^* = 50.7$ ns

$t_f^* = 7.8$ ns

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

$$t_r = 50.7 \left[1 + 0.3 (6.7/50.7)^{1.5} \right] = 51.4 \text{ ns}$$

$$t_{pLH} = 51.4/8.9 = 5.8 \text{ ns}$$

$$t_f = 7.8 \left[1 + (1.8/7.8)^{0.7} \right] = 10.6 \text{ ns}$$

$$t_{pHL} = 10.6/2.5 = 4.2 \text{ ns}$$

(ns)	Approximate Results	SPICE Results	% error
t_r	51.4	48.5	6
t_f	10.6	10.5	1
t_{pLH}	5.8	5.5	5
t_{pHL}	4.2	5.0	-16

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

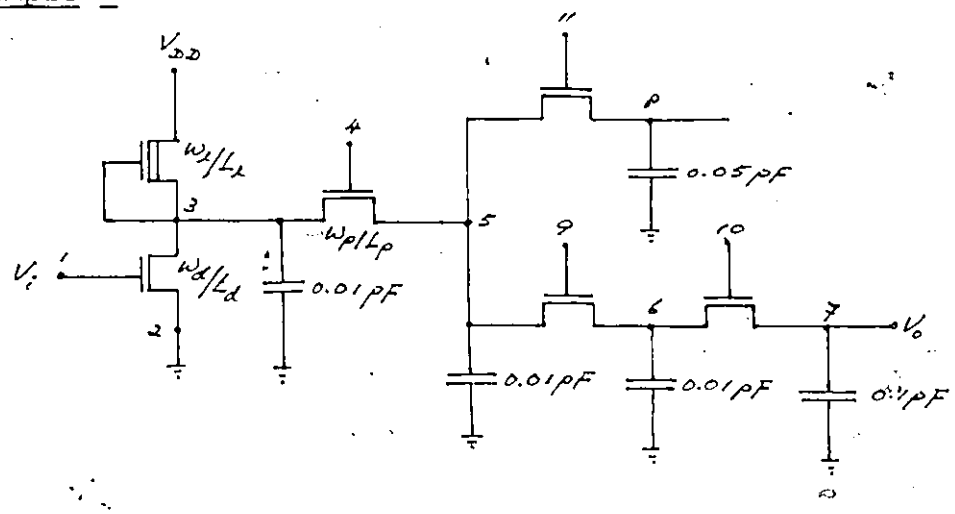


Figure 2.20: A pass transistor tree driven by an inverter.

Given process parameters:

V_{DD} = 5V, $\mu_n C_{ox}$ = 2×10^{-5} A/V^2

W_1/L_1 = 5/10, W_2/L_2 = 10/5, W_p/L_p = 5/5 (in μm)

V_{TD} = $-0.8V_{DD}$, V_{TE} = $0.2V_{DD}$

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

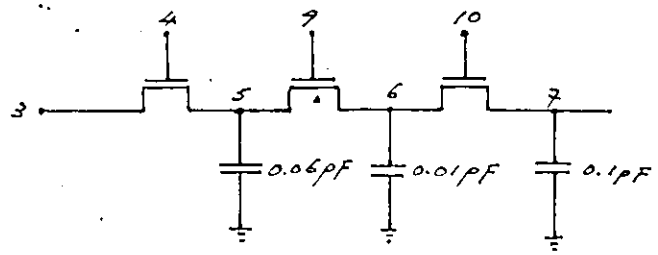
C_{eq} = (0.01 + $\frac{1}{\sqrt{2}}$ 0.06 + $\frac{1}{\sqrt{3}}$ 0.01 + $\frac{1}{\sqrt{4}}$ 0.1) = 0.11 pF

From equations (2.3) and (2.7),

t_r = $37 \times 10^3 \cdot (\frac{10}{5}) \times 0.11 \times 10^{-12}$ = 8.1 ns

t_f = $40.624 \times 10^3 \cdot (\frac{5}{10}) \times 0.11 \times 10^{-12}$ = 2.2 ns

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



So, V_f = $V_{DD} - V_{TE}$ = 4V

$(W/L)_{eq}$ = $\frac{1}{3}(\frac{W_p}{L_p})$ = $\frac{1}{3}$

C_{eq} = $\frac{1}{\sqrt{3}}$(0.06 + $\sqrt{2}$ 0.01 + $\sqrt{3}$ 0.1) = 0.14 pF

t_r^* = 17.778 $(\frac{1}{2 \times 10^{-5} \times 4})(3)(0.14 \times 10^{-12})$ = 93.3 ns

t_f^* = 2.7438 $(\frac{1}{2 \times 10^{-5} \times 4})(3)(0.14 \times 10^{-12})$ = 14.4 ns

Then, using the empirical equations,

$t_r = 93.3\ [\ 1 + 0.3\ (8.1/93.3)^{1.5}\ ] = 94$ ns

$t_{pLH} = 94/8.9 = 10.6$ ns

$t_f = 14.4\ [\ 1 + (2.2/14.4)^{0.7}\ ] = 18.3$ ns

$t_{pHL} = 18.3/2.5 = 7.3$ ns

(ns)	Approximate Results	SPICE Results	% error
t_r	94.0	81.0	16
t_f	18.3	16.5	11
t_{pLH}	10.6	9.0	18
t_{pHL}	7.3	8.3	-12

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.

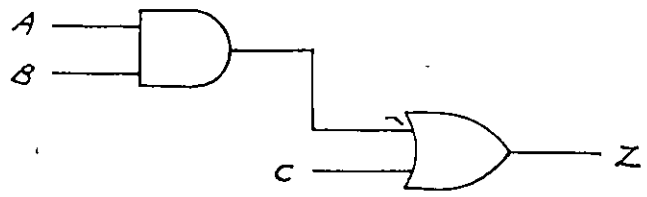


Figure 3.1: Circuit generated from Boolean equation

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.

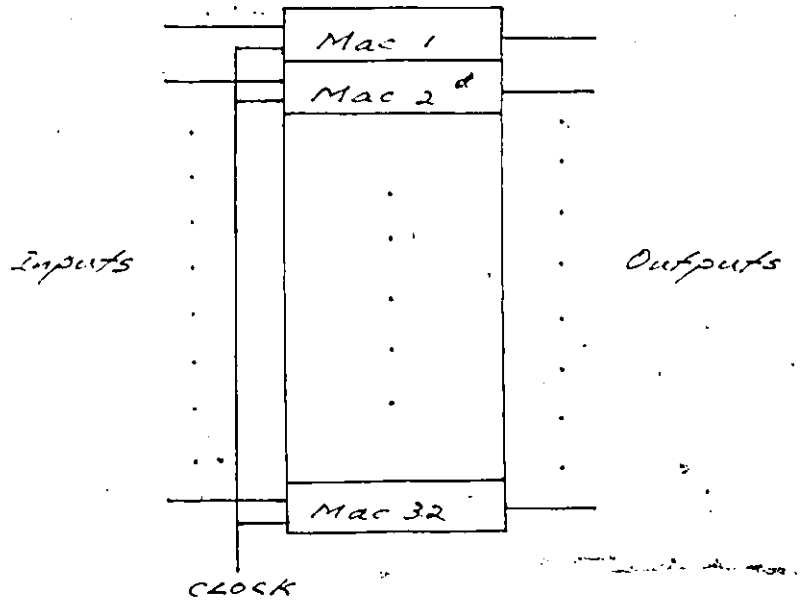


Figure 3.2: Macro representation of a 32-bit register

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.

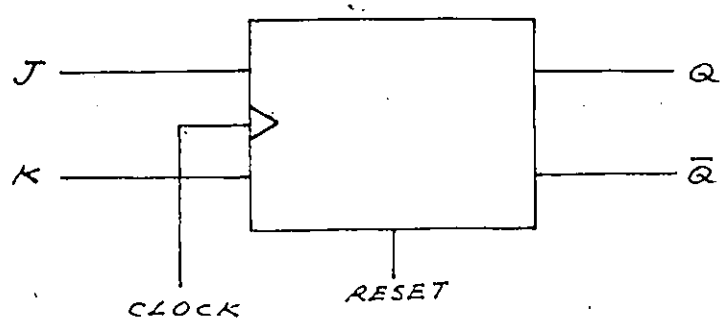


Figure 3.3: Element representation of a JK flip-flop

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.

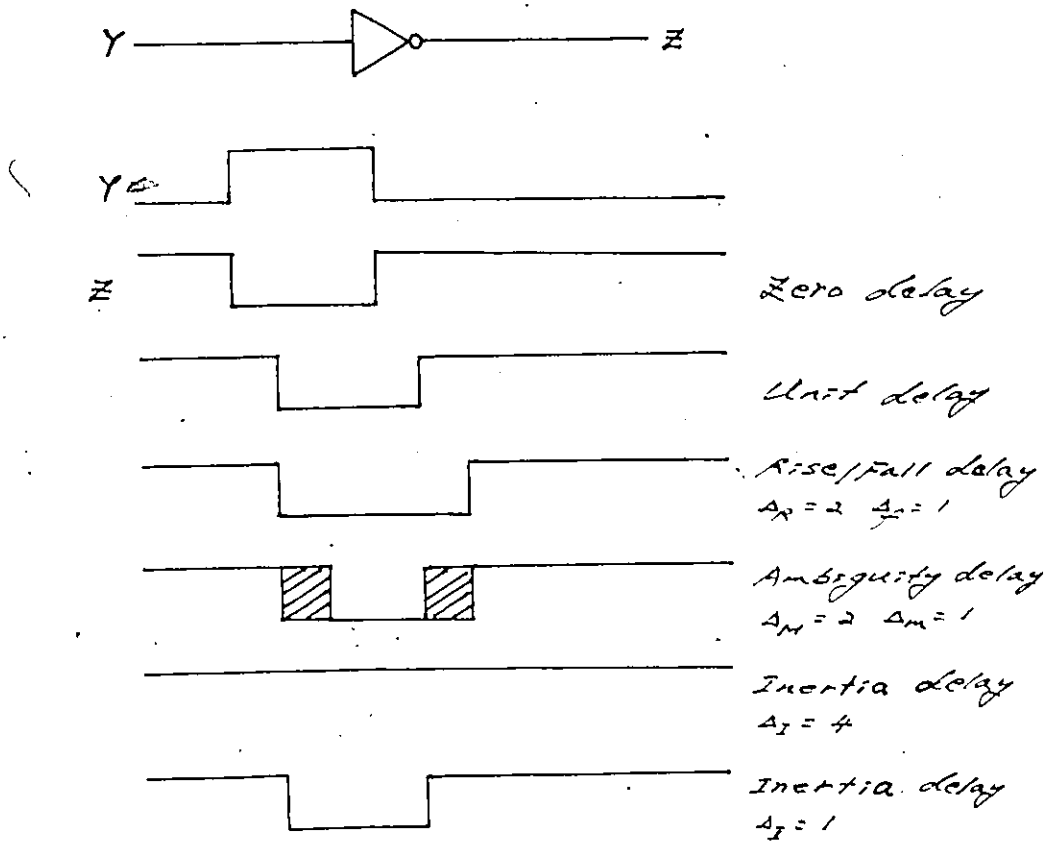


Figure 3.4: Input and output timing diagrams of an inverter for various delay models

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 Δ 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 occuring 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.

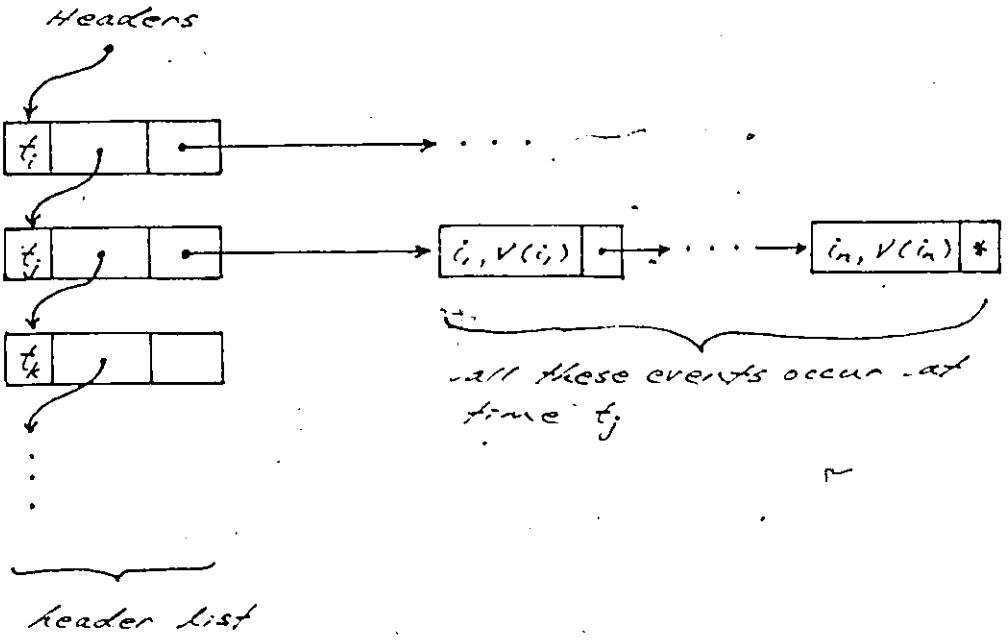


Figure 3.5: List structure scheduler

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, $n_1, n_2, \ldots, n_\lambda$, 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_{DD}, GND, and clocks.
2. A node connected to V_{DD} 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.

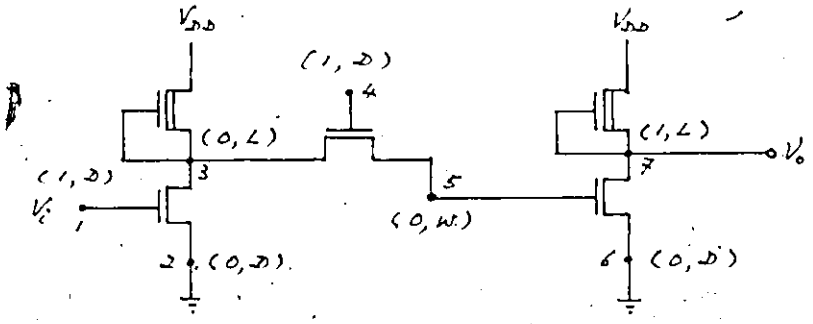


Figure 4.1: Illustrations of the node states and node strengths of a circuit.

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.

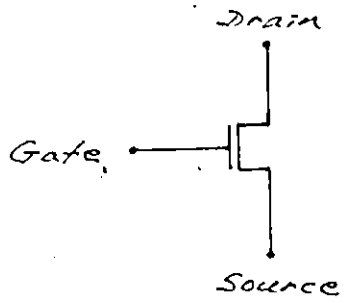


Figure 4.2: A MOS transistor.

n-type		p-type		d-type	
gate node state	transistor state (effect)	gate node state	transistor state (effect)	gate node state	transistor state (effect)
0	open	0	closed	0	closed
1	closed	1	open	1	closed
X	unknown	X	unknown	X	closed

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, Ω, in the following way:

$$\Omega = (N , T)$$

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.

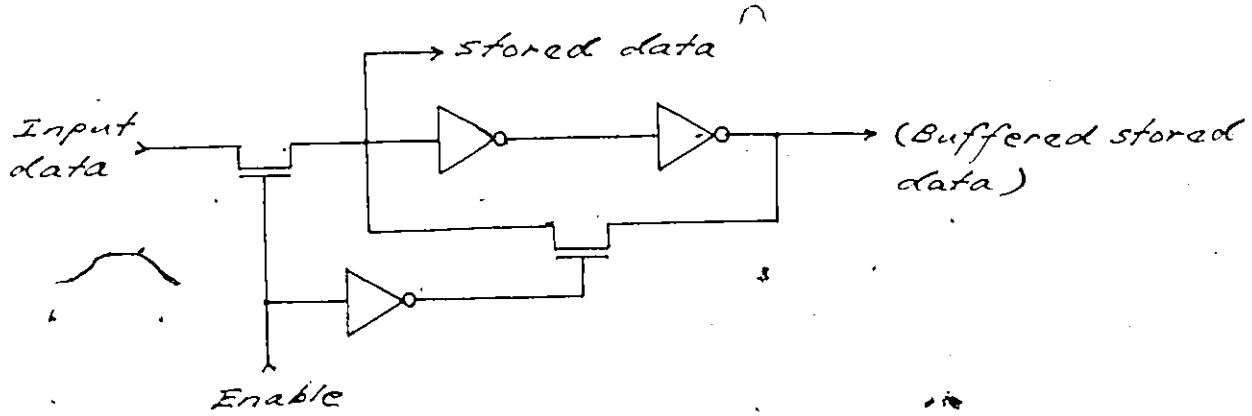
4.2.2 Partitioning of the Network

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:

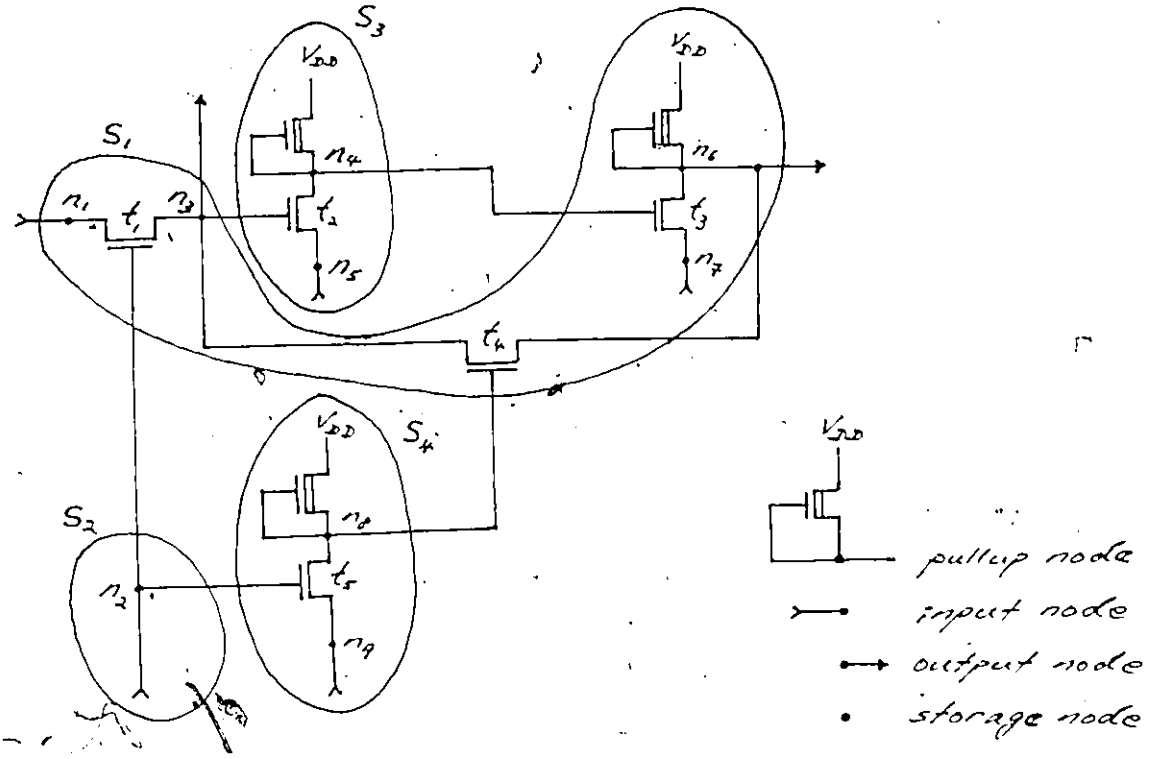
1. The nodes of S_u are obtained by partitioning the graph, formed by creating a vertex for each node and an edge between each pair of vertices that constitutes the source and drain for some transistor, into its connected components.
2. A transistor t_i is in the subnetwork S_u if a node of S_u is a 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

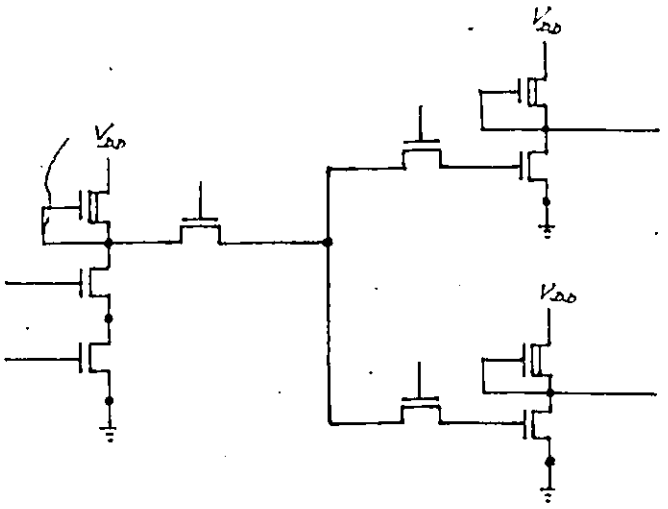
In this section a delay model is described. This delay model is organized around a subnetwork, instead of logic gates. The use of subnetwork permits both logic gates and pass transistors to be handled in a uniform fashion.

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.

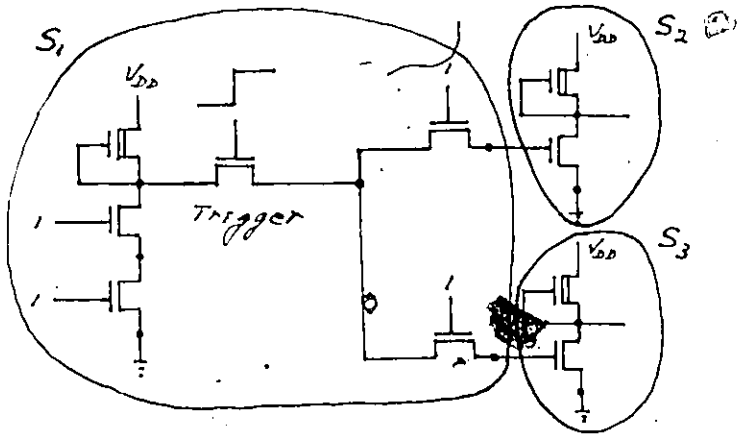
To compute the delay, first, the modeler starts at the source terminal of the trigger transistor and searches through sources and drains of other transistors to find all paths from the trigger source to V_{DD} or GND. For each such path, it then searches from the drain of the trigger transistor through 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.

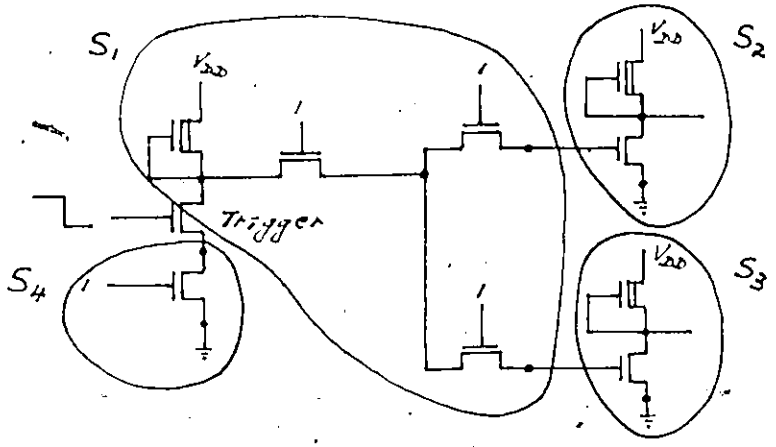
Figure 4.4 (b) shows the trigger transistor and the output nodes of subnetwork S_1. In the case when the trigger transistor is turned off, as shown in Figure 4.4(c), it does not belong to any subnetwork. Then the path from the output node to the V_{DD} is used to compute the delays.



(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 --> 1 and X --> 1 transitions are scheduled with rise delays.
2. 1 --> 0 and X --> 0 transitions are scheduled with fall delays.
3. 0 --> X and 1 --> 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.

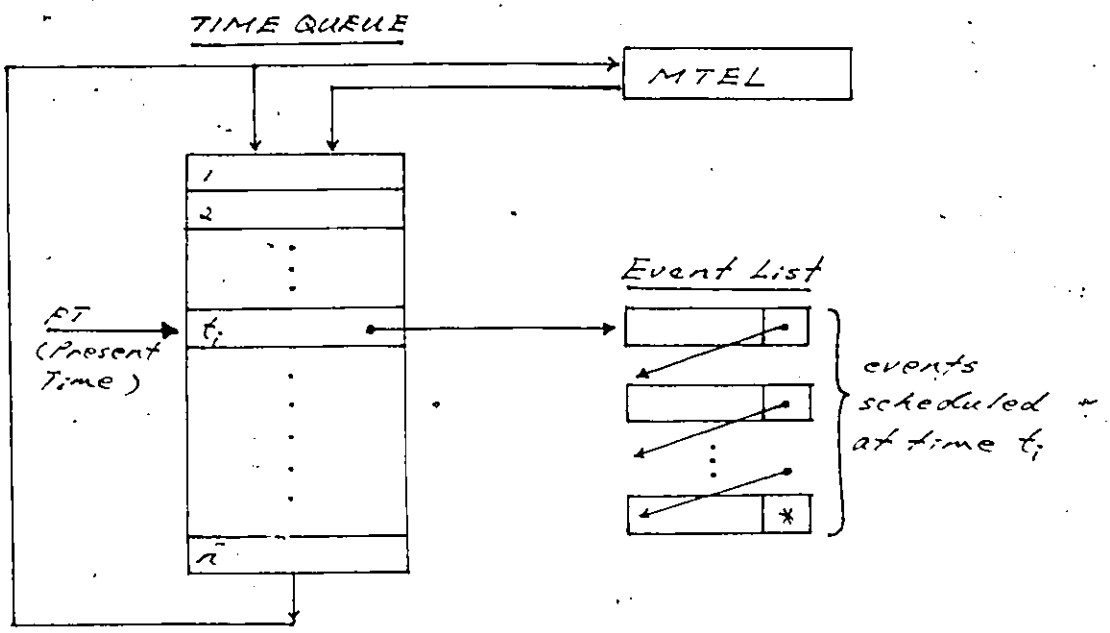


Figure 4.5: Simulation time flow mechanism (Time Queue).

During simulation, errors (e.g. spikes) are indicated by an X. For example, Figure 4.6 shows a circuit with rise delay t_{PLH} = 5 units and a fall delay t_{PHL} = 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.

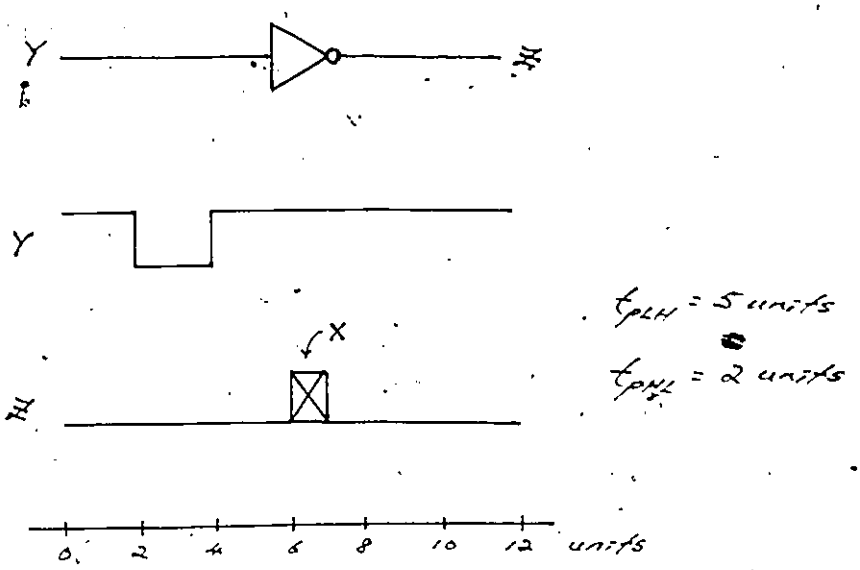


Figure 4.6: Input and output timing diagrams of an error condition.

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.

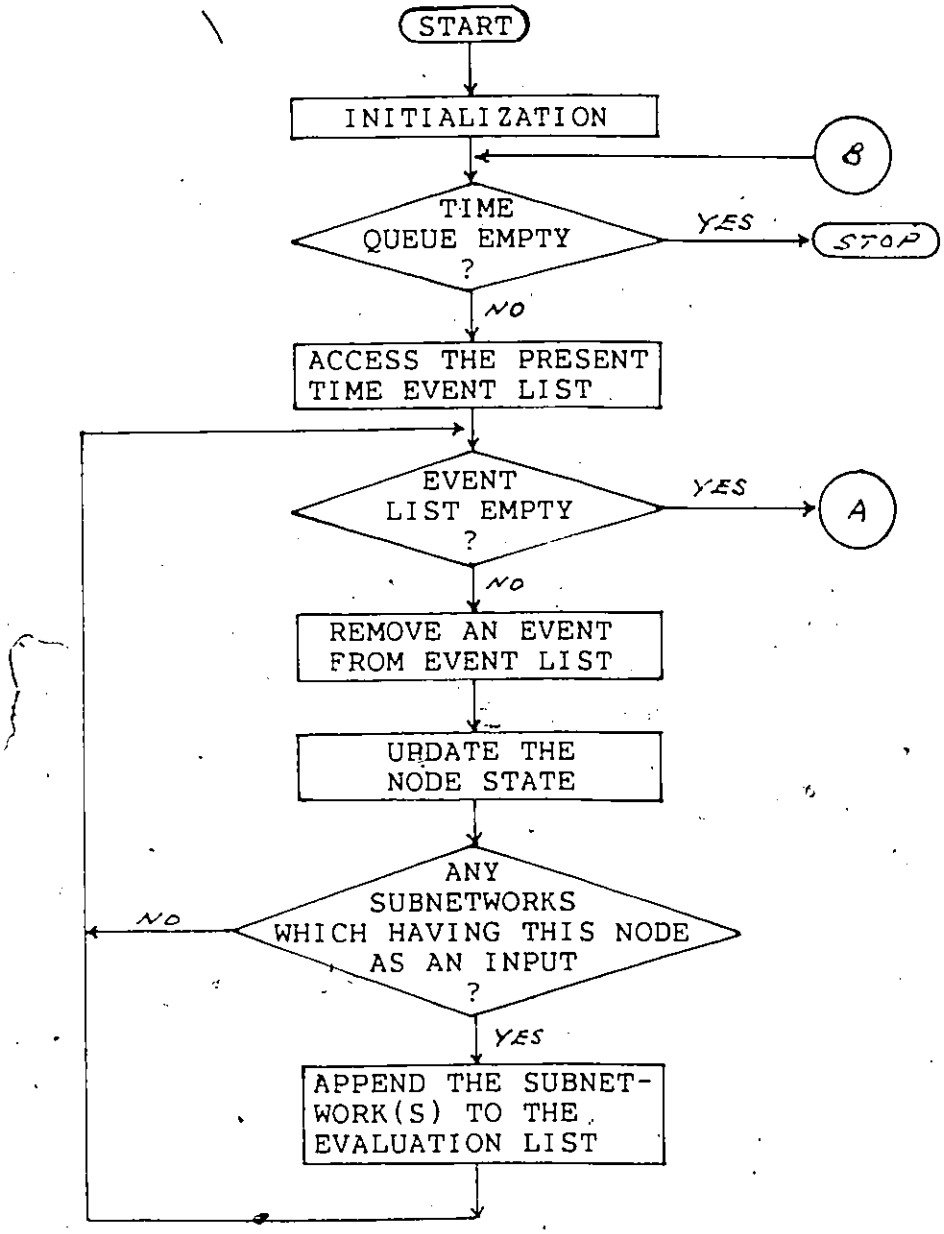


Figure 4.7: Event-Driven Simulation Algorithm.

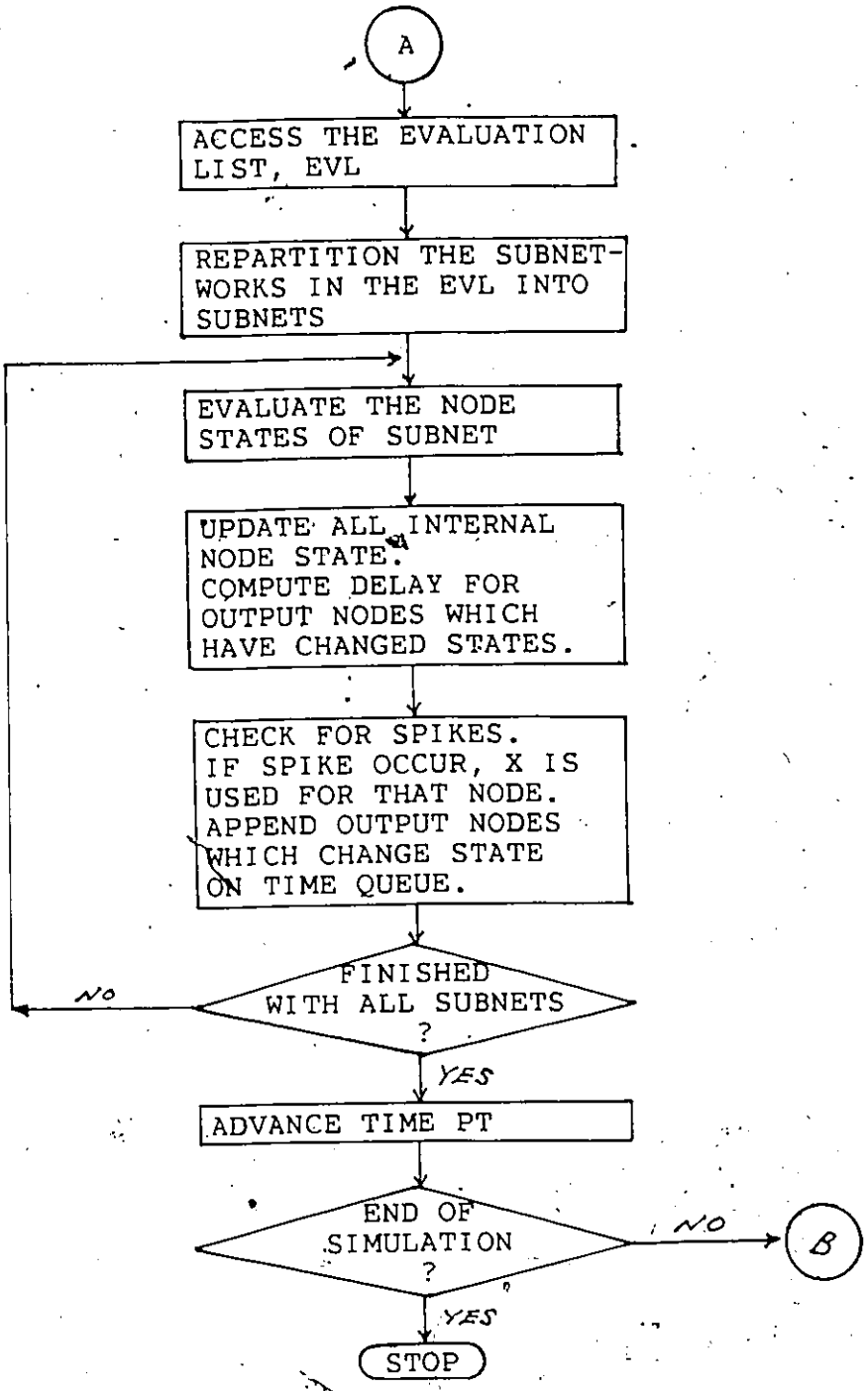


Figure 4.7 (continued)

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 Ω, a network configuration M and an input list A, containing sequence of input events to be simulated $<n_i, y_i, T_j>$.

Output: M is updated by simulating the input events in A.

```
procedure SIMULATION (Ω, M, A);
 begin
  (* initialization phase *)
  G <--- form an undirected graph with V = {n_i} and
  E = { <i,j> | for some t_g : SOURCE(t_g) = n_i and
  DRAIN(t_g) = n_j } ;
  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 <n_i, y_i, T_j> in A do
    TQ <--- <n_i, y_i> at time T_j;

  EVL = 0;
  for all n_i ∈ PT do
    if s_i ≠ y_i then
       begin    (* node state update phase *)
        s_i = y_i ;
        if for some t_g such that DRAIN(t_g) = n_i and
        n_i is input to some S_u then
          APPEND(EVL, S_u);
        if for some t_g such that Gate(t_g) = n_i then
           begin    (* transistor state update *)
            Z_g <--- SET_TRANS(TTYPE(t_g), s_i);
```

```
        if n_i input to some S_u then
          APPEND(EVL, S_u)
        else
          for some t_g such that SOURCE(t_g) ∈ S_v
          and DRAIN(t_g) ∈ S_w do
              begin
               APPEND(EVL, S_v);
               APPEND(EVL, S_w)
              end
      end
    end
G <--- form an undirected graph with V = { n_i | n_i ∈ EVL }
and E = { <i,j> | for some t_g : SOURCE(t_g) = n_i ,
DRAIN(t_g) = n_j and Z_g = 'closed' or 'unknown' } ;
SUBNETS <--- PARTITION(G);  (* subnetwork repartitioning *)
for all S_i ∈ SUBNETS do
   begin
    NODESTATE of S_i ;   (* subnetwork evaluation phase *)
    Delays of output nodes which have changed
    state in S_i ;
    Check for spikes;
    (* output scheduling phase *)
    Schedule events with delay delta at PT + Δ ;
    SUBNETWORKS <--- SUBNETWORKS U {S_i }
   end
while not end of simulation do
  increment PT
```

```
end.

procedure APPEND(EVL, Sx);
 begin
  if Sx ∉ EVL then
    begin
     EVL <--- EVL ∪ {Sx};
     remove Sx 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 behaviours 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 <-- form an undirected graph with V = {n_i} and
  E = { <i,j> | for some t_g : SOURCE(t_g) = n_i ,
  DRAIN(t_g) = n_j and Z_g = 'closed' } ;
  GROUPS <-- PARTITION(G);
  for each G_j ∈ GROUPS do
    strength(G_j), state(G_j) <-- STATE(G_j);
  SG <-- form an undirected graph with V = {G_j} and
  E = { <i,j> | for some t_g : SOURCE(t_g) ∈ G_i and
  DRAIN(t_g) ∈ G_j and Z_g = 'unknown' } ;
  if |SG| > 1 then
     begin
      Y <-- SUPERSTATE( strength, state, SG );
      P <-- {G_j ∈ SG | state(G_j) ≠ Y };
      POISON( P, SG, strength );
      for each G_j ∈ P do state(G_j) <-- X
     end
  for each G_j ∈ GROUPS do
    for each n_i ∈ G_j do
      if f_i ≠ "D" and s_i ≠ state(G_j) then
        y_i <-- state(G_j);
  for each n_i ∈ S_i do
    if s_i ≠ y_i and NTYPE(n_i) ≠ 'external' then
```

s_i <-- 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.

		node(i)		
		Driver (D)	Pullup (L)	Weak (W)
node(j)	Driver (D)	D	D	D
	Pullup (L)	D	L	L
	Weak (W)	D	L	W

(a) Group strength

		node(i)								
		Driver (D)			Pullup (L)			Weak (W)		
node(j)		0	1	X	0	1	X	0	1	X
Driver (D)	0	0	X	X	0	0	0	0	0	0
	1	X	1	X	1	1	1	1	1	1
	X	X	X	X	X	X	X	X	X	X
Pullup (L)	0	0	1	X	0	X	X	0	0	0
	1	0	1	X	X	1	X	1	1	1
	X	0	1	X	X	X	X	X	X	X
Weak (W)	0	0	1	X	0	1	X	0	X	X
	1	0	1	X	0	1	X	X	1	X
	X	0	1	X	0	1	X	X	X	X

(b) Group state

Table 4.2
: Group stength and state table.

Input: A group G_x.

Output: The strength and state of G_x.

```
procedure STATE(Gx);
 begin
  k <-- null;      (* Driver > Pullup > Weak > null  *)
  y <-- X;
  for each ni ∈ Gx do
    if fi > k then
       begin
        k <-- fi ;
        y <-- si
       end
    else
      if fi = k and si ≠ y then
        y <-- X
 end
```

Input: A supergroup, strength and state of each group in the supergroup.

Output: State of supergroup.

```
procedure SUPERSTATE (strength, state, SG);
 begin
  K <-- null;    (* Driver > Pullup > Weak > null  *)
  Y <-- X;
  for each Gi ∈ SG do
    if strength(Gi) > K then
       begin
        K <-- strength(Gi);
        Y <-- state(Gi);
       end
    else
      if strength(Gi) = K and state(Gi) ≠ Y then
        Y <-- X
 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 ≠ 0 do
     begin
      Gj <-- an element of B with maximal strength;
      B  <-- B - {Gj};
      for each GK connected to Gj by an X-transistor do
        if pstrength(GK) < pstrength(Gj)
        or (pstrength(GK) = pstrength(Gj) and GK ∉ P) then
           begin
            P <-- P ∪ {GK};
            B <-- B ∪ {GK};
            pstrength(GK) <-- pstrength(Gj)
           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 <-- 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 <-- {v_i };
      SEARCH(S_j , v_i );
      j <-- 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
```

```
        S <-- S U {w};
        SEARCH(S,w);
        end
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.

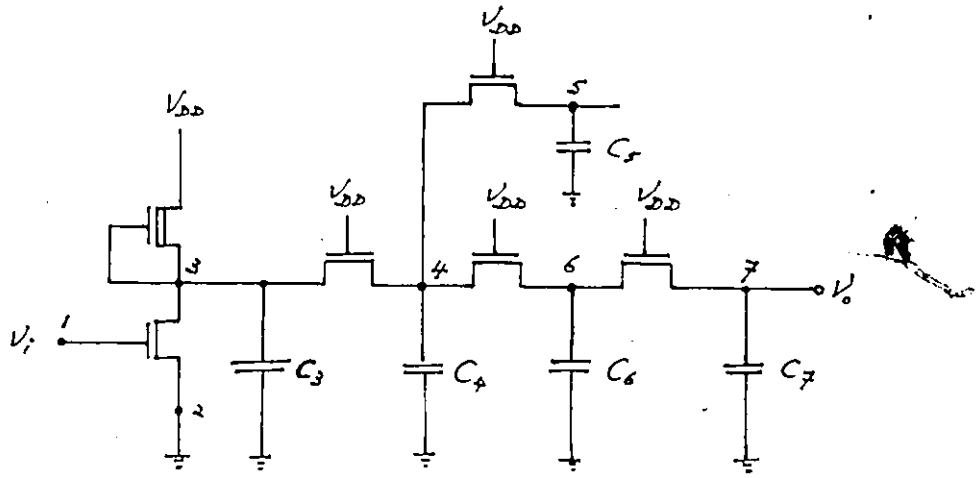


Figure 4.8 : Side capacitances are assumed to be connected to the branch nodes. In computing delays, for direct path from node 3 to node 7, side capacitance C_5 is assumed to be connected to node 4 (total capacitance at node 4 is $C_4 + C_5$).

```
Input: A subnet S
Output: Delays of each output node which has changed state.

procedure DELAY(S);
 begin
  for each n_i ∈ S do
    if NTYPE(n_i) = 'external' and s_i ≠ 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
```

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.

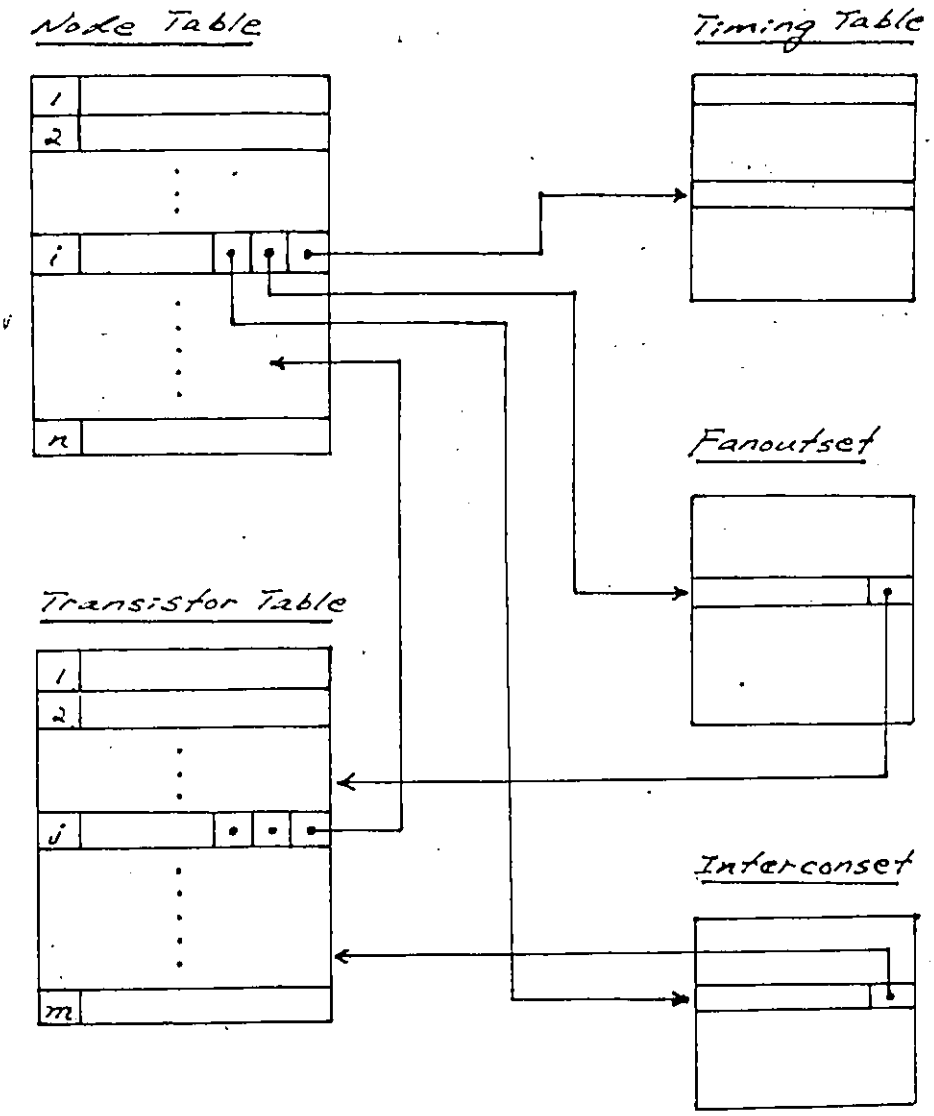


Figure 5.1 : Overall table configuration and interaction

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.

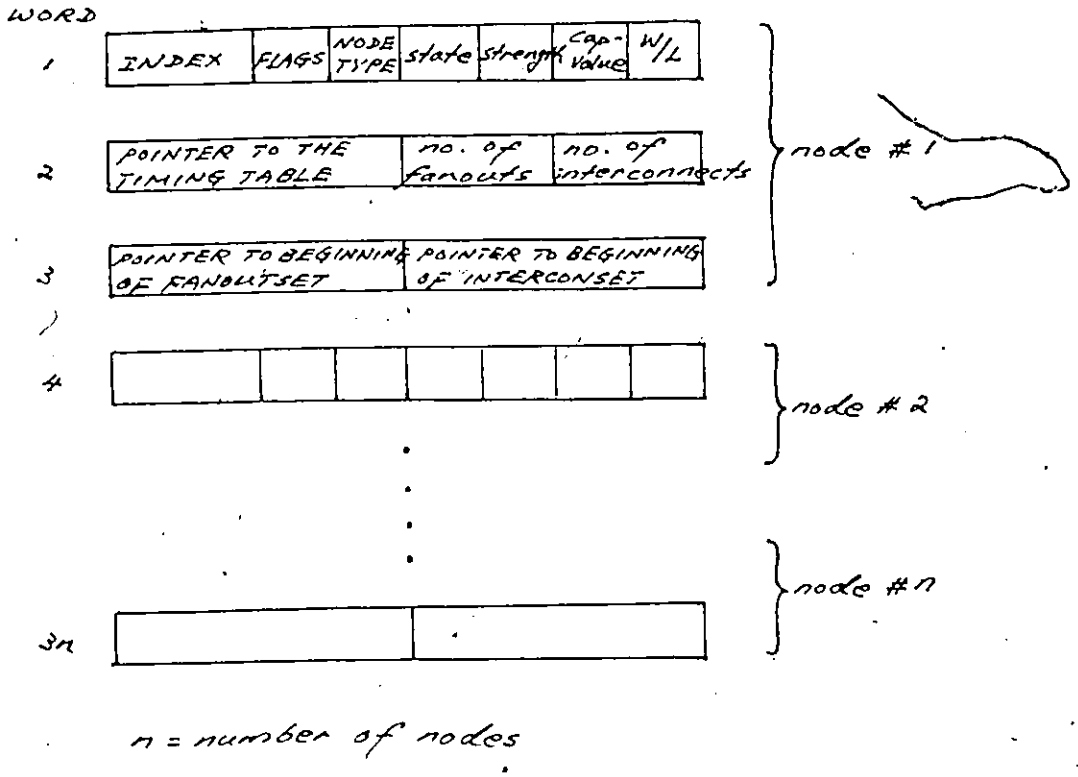


Figure 5.2 : Node table

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

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.

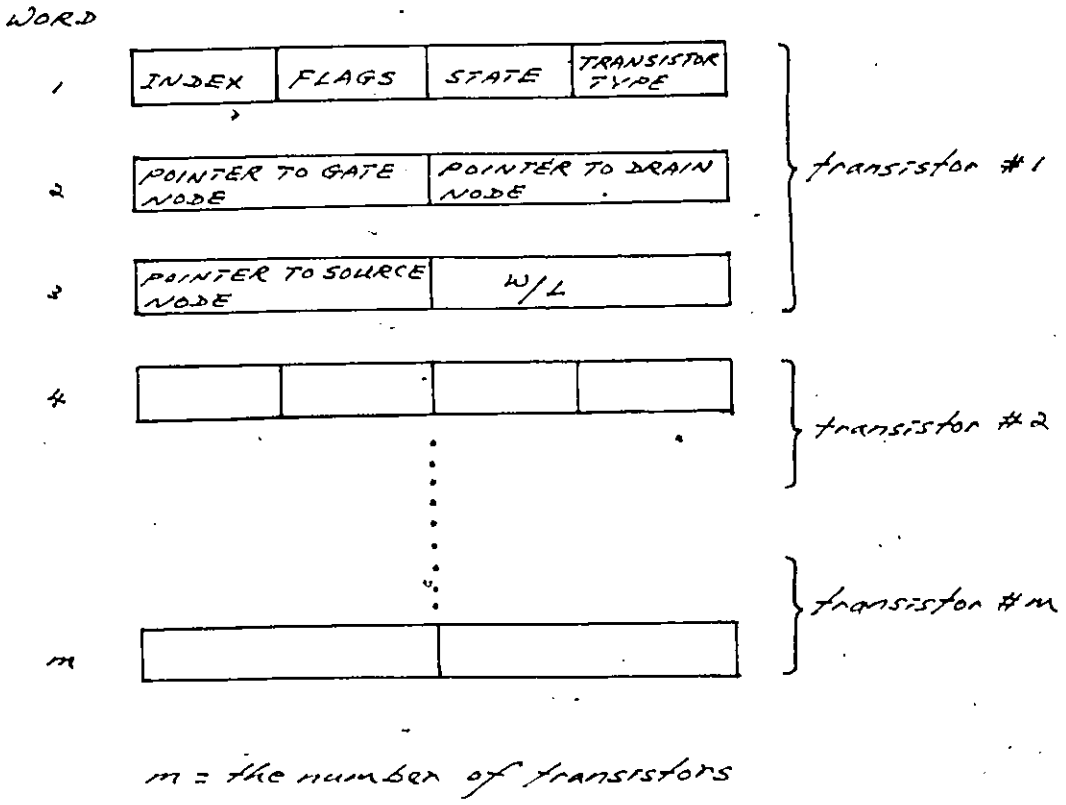


Figure 5.4 : Transistor table

Field	Description	Usage
index	indicates the transistor number	argument in vector arrays to address the piece of information desired
flags	indicators	checking of certain condition
state	transistor state	state evaluation
transistor type	type of transistor	state evaluation
pointer to gate node	pointer points to the gate terminal	state evaluation
pointer to drain node	pointer points to the drain terminal	state evaluation
pointer to source node	pointer points to the source terminal	state evaluation
W/L	width and length of the transistor channel	delay calculation

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.

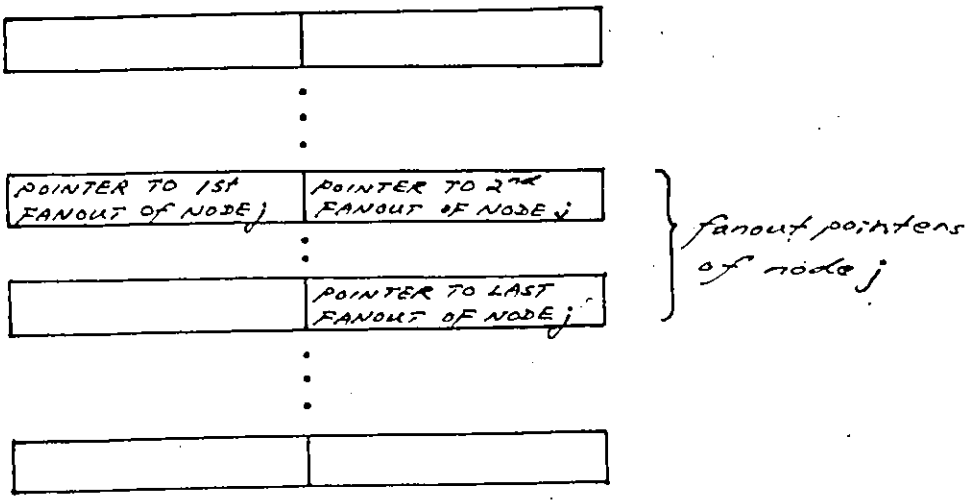


Figure 5.6 : Fanoutset table

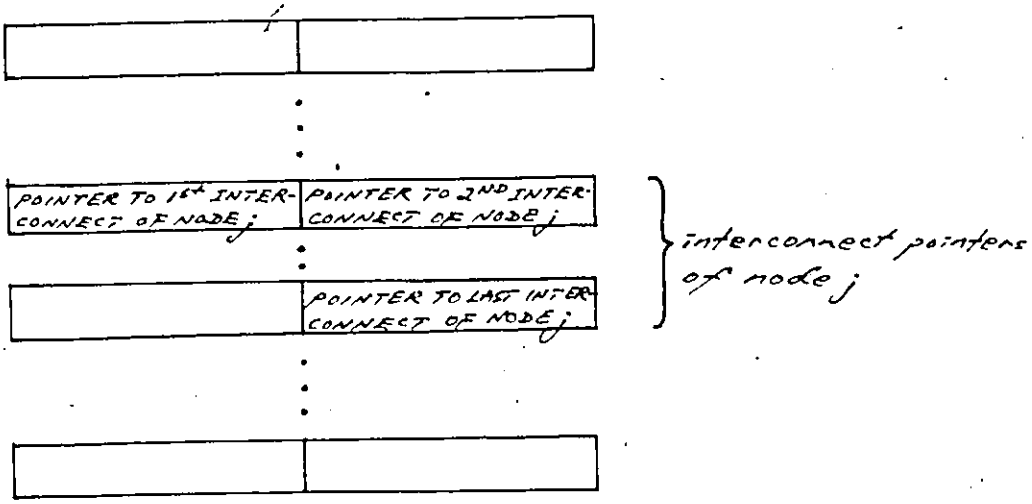


Figure 5.7 : Interconset table

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.

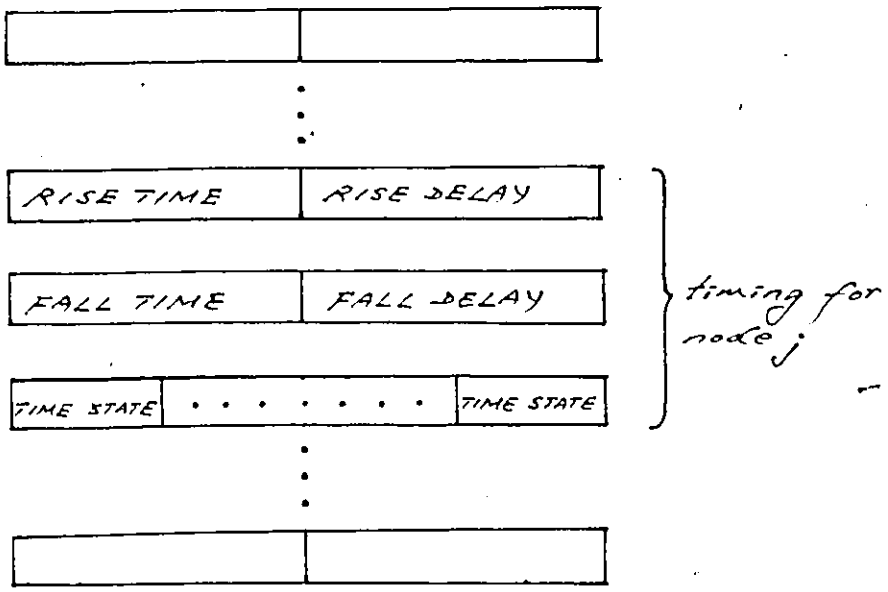


Figure 5.8 : Timing table

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.

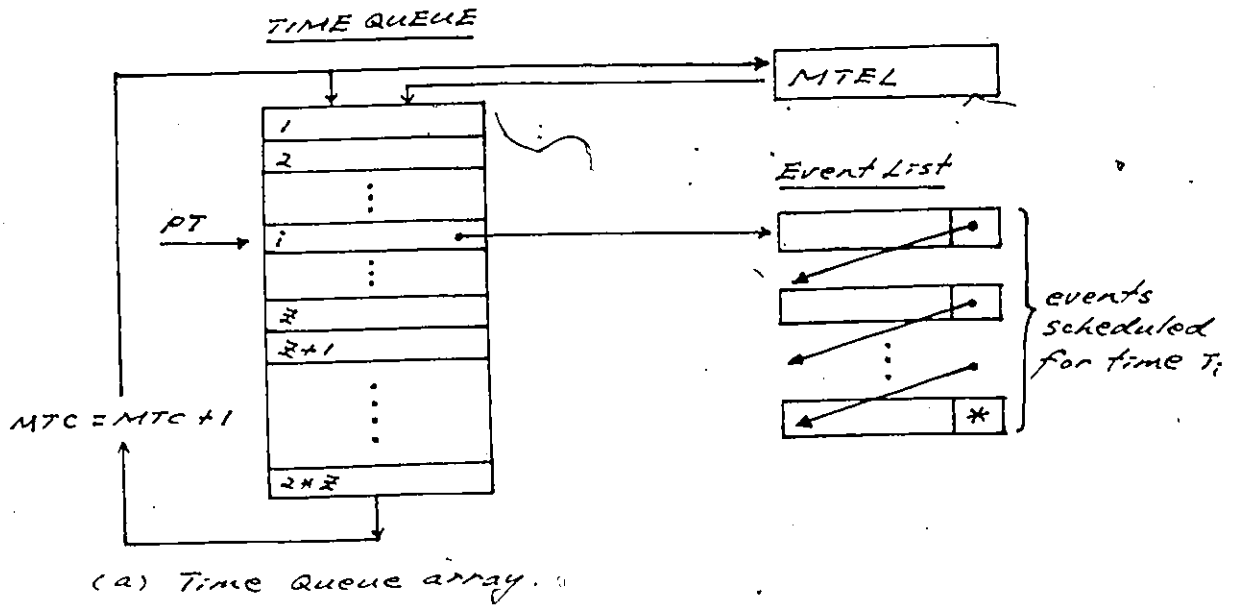


Figure 5.9 : Simulation time flow mechanism

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.

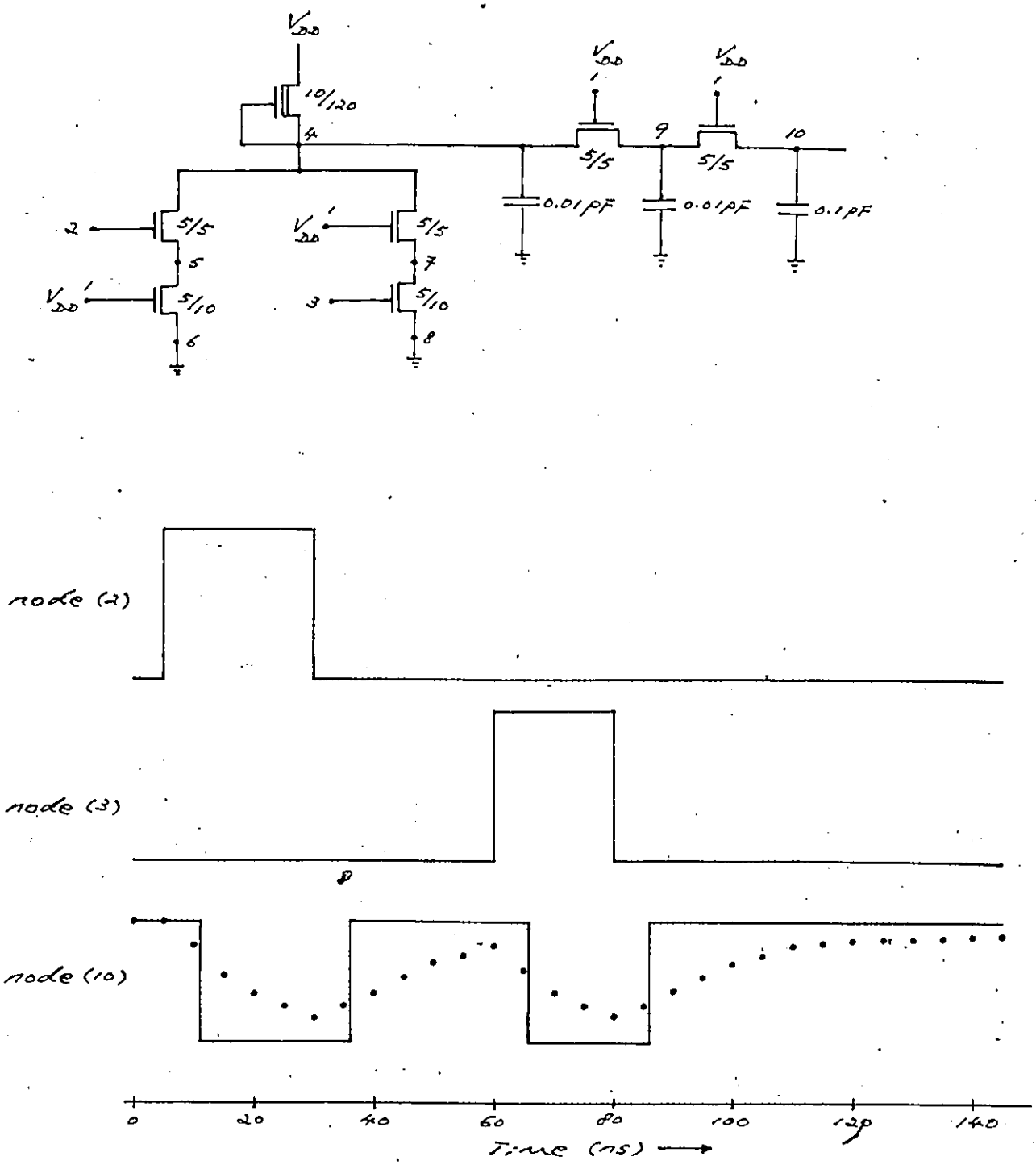
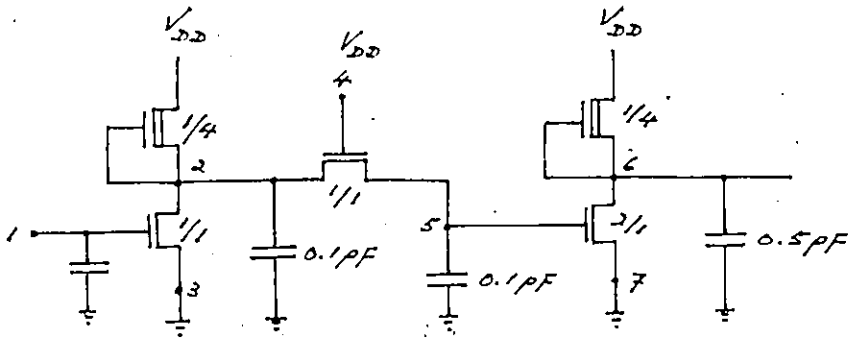


Figure 5.10 : Input and output waveforms for Example 5

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).



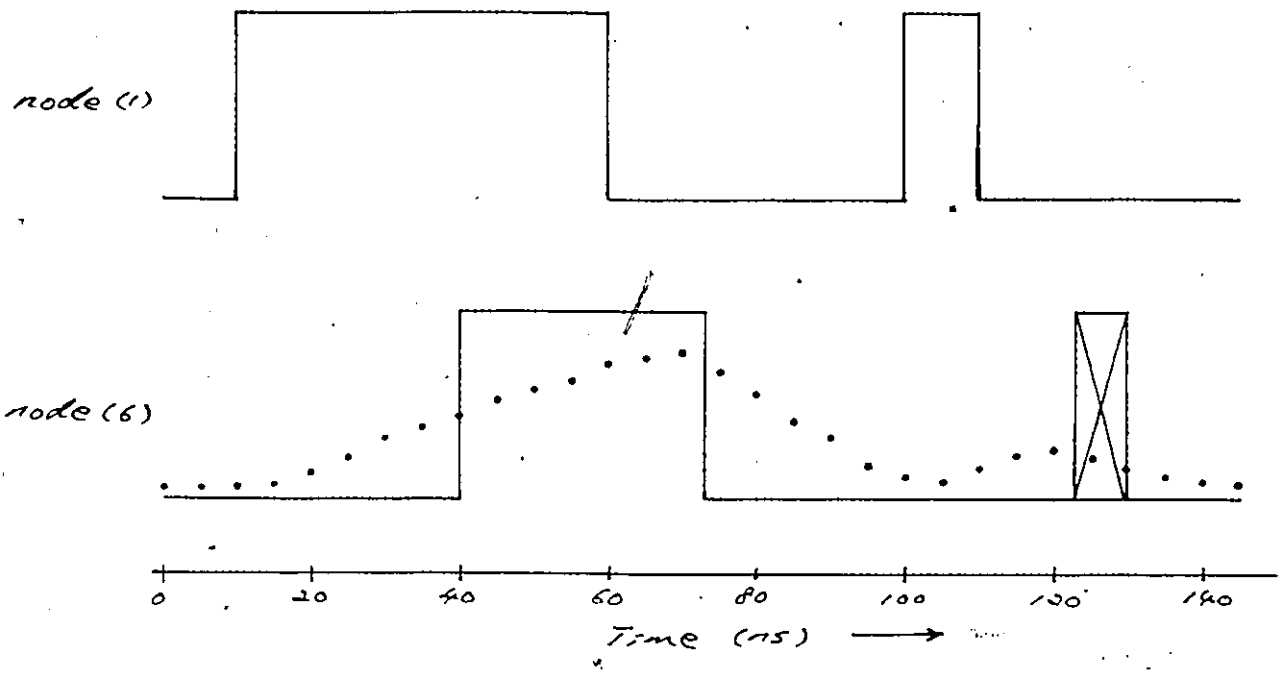


Figure 5.11 : Input and output waveforms for Example 6

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

1. Agrawal, V.D. et al, "A Mixed-Mode Simulator", Proc. of 17th Design Automation Conf., pp.618-625, 1980.

2. Aho, A.V. , Hopcroft, J.E. and Ullman, J.D. , The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

3. Baker, C.M. and Terman, C. , "Tools for Verifying Integrated Circuit Designs", Lambda, pp.22-30, 1980.

4. Breuer, M.A. and Friedman, A.D. , Diagnosis & Reliable Design of Digital Systems, Computer Science Press, Inc., 1976.

5. Bryant, R.E. , "An Algorithm for MOS Logic Simulation", Lambda, pp.46-53, 1980.

6. Bryant, R.E. , "A Switch-Level Model of MOS Logic Circuits", VLSI 81, New York: Academic, pp.329-340, Aug. 1981.

7. Bryant, R.E. , "A Switch-Level Model and Simulator for MOS Digital Systems", IEEE Trans. Computers, Vol.C-33, No.2, pp.160-177, Feb. 1984.

8. Chang, H.Y. et al , "LAMP: System Description", B.S.T.J., Vol.53, pp.1431-1449, Oct. 1974.

9. Chappell, S.G., Elmendorf, C.H. and Schmidt, L.D. , "LAMP: Logic-Circuit Simulators", B.S.T.J., Vol.53, pp.1451-1476, Oct. 1974.

10. Chawla, B.R., Gummel, H.K. and Kozak, P. , "MOTIS- An MOS Timing Simulator", IEEE Trans. Circuits and Systems, Vol.CAS-22, pp.901-910, Dec. 1975.

11. Chawla, B.R. et al , "Operational Features of an MOS Timing Simulator", Proc. of 12th Design Automation Conf., pp.95-101, 1975.

12. Chen, C.F. et al , "The Second Generation MOTIS Mixed-Mode Simulator", Proc. of 21st Design Automation Conf., pp.10-17, 1984.

13. Hodges, D.A. and Jackson, H.G. , Analysis and Design of Digital Integrated Circuits, McGraw-Hill, 1983.

14. Mead, C.A. and Conway, L.A. , Introduction to VLSI Systems, Addison-Wesly, 1980.

15. Nagel, L.W. , SPICE2: A Computer Program to Simulate Semiconductor Circuits, Univ. of California, Berkeley, ERL Memo No.ERL-M520, May 1975.

16. Nham, H.M. and Bose, A.K. , "A Multiple Delay Simulator for MOS LSI Circuits", Proc. of 17th Design Automation Conf., pp.610-617, 1980.

17. Ramanchandran, V. , "An Improved Switch-Level Simulator for MOS Circuits", Proc. of 20th Design Automation Conf., pp.293-299, 1983.

18. Szygenda, S.A. and Thompson, E.W. , "Digital Logic Simulation in a Time-Based, Table-Driven Enviroment, Part 1: Design Verification", Computer, Vol.18, pp.24-36, 1975.

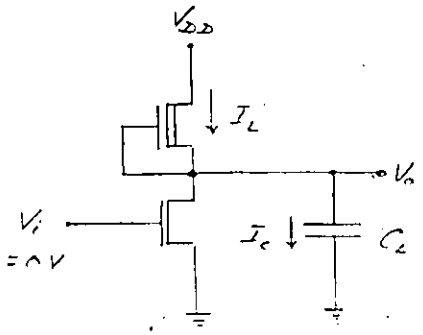
19. Terman, C.J. , "RISM - A Logic-Level Timing Simulator", Proc. ICCD 83, pp.437-440, 1983.

20. Weeks, W.T. et al , "Algorithms for ASTAP - A Network Analysis Program", IEEE Trans. Circuit Theory, Vol.CT-20, pp.628-634, 1973.

21. Wilcox, P. and Rombeek, H. , "F/Logic - An Interactive Fault and Logic Simulation For Digital Circuits", Proc. 13th Design Automation Conf., pp.68-73, 1976.

Appendix A

DERIVATION OF RISE TIME AND RISE DELAY OF AN NMOS INVERTER



$V_o(0) = V_z$

$V_{TD} < 0,\ V_{TE} > 0$

Rise Time

When $V_{GS} = 0V > V_{TD}$, $V_{DS} = V_{DD} - V_o \geq V_{GS} - V_{TD} = -V_{TD}$, the load transistor is in saturation region.

$$I_L = \frac{U_n C_o}{2} \cdot \frac{W_1}{L_1} \cdot (V_{GS} - V_{TD})^2$$

$$= \frac{U_n C_o}{2} \frac{W_1}{L_1} \cdot (-V_{TD})^2$$

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

$$C_L \frac{dV_o}{dt} = \frac{U_n C_o}{2} \frac{W_1}{L_1} \cdot (-V_{TD})^2$$

The time in the saturation region,

$$t_{sat} = \int_{X_1}^{V_{DD} + V_{TD}} \frac{2 C_L}{U_n C_o (W_1/L_1) (-V_{TD})^2} \cdot dV_o$$

$$= \frac{2 C_L}{U_n C_o (W_1/L_1) (-V_{TD})^2} (V_{DD} + V_{TD} - X_1)$$

When $V_{GS} = 0V > V_{TD}$, $V_{DS} = V_{DD} - V_o \leq -V_{TD}$, the load transistor is in linear region,

$$I_L = \frac{U_n C_o}{2} \cdot \frac{W_1}{L_1} \cdot \left[2(V_{GS} - V_{TD})\, V_{DS} - V_{DS}^2 \right]$$

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

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

$$C_L \frac{dV_o}{dt} = \frac{U_n C_o}{2} \cdot \frac{W_1}{L_1} \cdot \left[2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]$$

The time in the linear region,

$$t_{lin} = \int_{V_{DD} - V_{TD}}^{X_2} \frac{2 C_L}{U_n C_o W_1/L_1} \cdot \frac{dV_o}{2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2}$$

$$= \frac{C_L}{U_n C_o (-V_{TD}) W_1/L_1} \cdot \int_{V_{DD} - V_{TD}}^{X_2} \left(\frac{1}{V_{DD} - V_o} + \frac{1}{2(-V_{TD}) - (V_{DD} - V_o)} \right) \cdot dV_o$$

$$= \frac{C_L}{U_n C_o (-V_{TD}) W_1/L_1} \cdot \left[\ln\left(\frac{2(-V_{TD}) - (V_{DD} - V_o)}{(V_{DD} - V_o)} \right) \right]_{V_{DD} - V_{TD}}^{X_2}$$

$$= \frac{C_L}{U_n C_o (-V_{TD}) W_1/L_1} \cdot \ln\left(\frac{2(-V_{TD}) - (V_{DD} - X_2)}{(V_{DD} - X_2)} \right)$$

The total rise time t_r is

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

$$= \frac{2\ C_L}{U_n\ C_o\ V_{DD}\ W_1/\ L_1}\cdot\left[\frac{V_{DD}}{(-V_{TD})^2}\cdot(V_{DD} + V_{TD} - X_1)\right.$$

$$\left. + \frac{V_{DD}/2}{(-V_{TD})}\cdot\ln\left(\frac{2(-V_{TD}) - (V_{DD} - X_2)}{(V_{DD} - X_2)}\right)\right]$$

Rise Delay

When the load transistor is in saturation region,

$$t_{pl} = \int_{V_2}^{V_{DD} + V_{TD}} \frac{2\ C_L}{U_n C_o (-V_{TD})^2\ W_1/L_1}\cdot dV_o$$

$$= \frac{2\ C_L}{U_n C_o (-V_{TD})^2\ W_1/L_1}\cdot(V_{DD} + V_{TD} - V_L)$$

When the load transistor is in linear region,

$$t_{p2} = \int_{V_{DD}+V_{TD}}^{X_{.5}} \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \frac{dV_o}{2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2}$$

$$= \frac{C_L}{U_n C_o (-V_{TD}) W_1/L_1} \cdot \left[\ln\left(\frac{2(-V_{TD}) - (V_{DD} - V_o)}{(V_{DD} - V_o)} \right) \right]_{V_{DD}+V_{TD}}^{X_{.5}}$$

$$= \frac{C_L}{U_n C_o (-V_{TD}) W_1/L_1} \cdot \ln\left(\frac{2(-V_{TD}) - (V_{DD} - X_{.5})}{(V_{DD} - X_{.5})} \right)$$

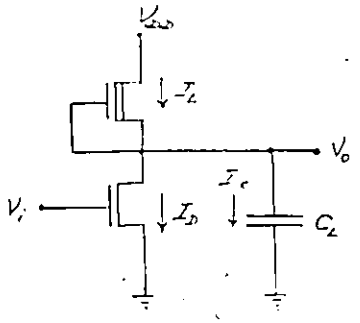
The total rise delay t_{pLH} is

$$t_{pLH} = t_{p1} + t_{p2}$$

$$= \frac{2\,C_L}{U_n\, C_o\, V_{DD}\, W_1/L_1} \cdot \left[\frac{V_{DD}}{(-V_{TD})^2} (V_{DD} + V_{TD} - V_L) + \frac{V_{DD}/2}{(-V_{TD})} \cdot \ln\left(\frac{2(-V_{TD}) - (V_{DD} - X_{.5})}{(V_{DD} - X_{.5})} \right) \right]$$

Appendix B

DERIVATION OF FALL TIME AND FALL DELAY OF AN NMOS INVERTER



$V_o(0) = V_{DD}$

$V_{TD} < 0$, $V_{TE} > 0$

$k = \frac{W_d/L_d}{W_l/L_l}$

Fall Time

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

$$I_L = \frac{U_n C_o}{2} \cdot \frac{W_l}{L_l} \cdot \left[2(-V_{TD})(V_{DD} - V_o) - (V_{DD} - V_o)^2 \right]$$

$$I_D = \frac{U_n C_o}{2} \cdot \frac{W_d}{L_d} \cdot (V_i - V_{TE})^2$$

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

$$C_L \frac{dV_o}{dt} = - \frac{U_n C_o}{2} \frac{W_l}{L_l} \left[(V_{DD} - V_o)^2 - 2(-V_{TD})(V_{DD} - V_o) + k(V_i - V_{TE})^2 \right]$$

$$t_{fl} = - \frac{2 C_L}{U_n C_o W_l / L_l} \cdot \int_{X_2}^{V_i - V_{TE}} \frac{dV_o}{(V_{DD} - V_o)^2 - 2(-V_{TD})(V_{DD} - V_o) + k(V_i - V_{TE})^2}$$

let $V = V_{DD} - V_o$, $dV = -dV_o$

$$t_{fl} = \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \int_{V_{DD}-X_2}^{V_{DD}-(V_i-V_{TE})} \frac{dV}{V^2 - 2(-V_{TD})V + k(V_i - V_{TE})^2}$$

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

$$\int \frac{dX}{aX^2 + bX + c} = \frac{2}{\left[4ac - b^2\right]^{1/2}} \cdot \tan^{-1}\left(\frac{2aX + b}{\left[4ac - b^2\right]^{1/2}} \right)$$

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

$$= \frac{1}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}}$$

$$t_{fl} = \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \frac{2}{\left[4ac - b^2\right]^{1/2}} \cdot \left[\tan^{-1}\left(\frac{2aV + 2V_{TD}}{\left[4ac - b^2\right]^{1/2}}\right) \right]_{V_{DD} - X_2}^{V_{DD} - (V_i - V_{TE})}$$

$$= \frac{V_{DD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}} \cdot \left[\tan^{-1}\left(\frac{V_{DD} - (V_i - V_{TE}) - V_{TD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}}\right) \right.$$

$$\left. - \tan^{-1}\left(\frac{V_{DD} - X_2 + V_{TD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}}\right) \right] \cdot \frac{2\,C_L}{U_n C_o V_{DD} W_1/L_1}$$

When $V_i - V_{TE} \geq V_o \geq V_{DD} + V_{TD}$, both transistors are in linear region,

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

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

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

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

$$t_{f2} = \left(\int_{V_i - V_{TE}}^{V_{DD} + V_{TD}} \frac{dV_o}{(k-1)V_o^2 + (2V_{TD} + 2V_{DD} - 2k(V_i - V_{TE}))V + (-2V_{TD}V_{DD} - V_{DD}^2)} \right) \cdot \frac{2 C_L}{U_n C_o W_1/L_1}$$

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

$$\int \frac{dX}{aX^2 + bX + c} = \frac{1}{\left[b^2 - 4ac\right]^{1/2}} \cdot \ln\left(\frac{2aX + b - \left[b^2 - 4ac\right]^{1/2}}{2aX + b + \left[b^2 - 4ac\right]^{1/2}} \right)$$

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

$$t_{f2} = \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \frac{1}{\left[b^2 - 4ac\right]^{1/2}} \cdot \left[\ln \left| \frac{2aV_o + b - \left[b^2 - 4ac\right]^{1/2}}{2aV_o + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right]_{V_i - V_{TE}}^{V_{DD} + V_{TD}}$$

$$= \frac{V_{DD}}{\left[b^2 - 4ac\right]^{1/2}} \cdot \left[\ln \left| \frac{2a(V_{DD} - V_{TD}) + b - \left[b^2 - 4ac\right]^{1/2}}{2a(V_{DD} - V_{TD}) + b + \left[b^2 - 4ac\right]^{1/2}} \right| - \ln \left| \frac{2a(V_i - V_{TE}) + b - \left[b^2 - 4ac\right]^{1/2}}{2a(V_i - V_{TE}) + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right] \cdot \frac{2\,C_L}{U_n C_o V_{DD} W_1/L_1}$$

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

$$I_L = \frac{U_n C_o}{2} \cdot \frac{W_1}{L_1} \cdot (-V_{TD})^2$$

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

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

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

$$t_{f3} = \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \int_{V_{DD} + V_{TD}}^{X_1} \frac{dVo}{kV_o^2 - 2k(V_i - V_{TE})V_o + (-V_{TD})^2}$$

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

$$t_{f3} = \frac{V_{DD}}{\left[b^2 - 4ac\right]^{1/2}} \cdot \left[\ln \left| \frac{2aX_1 + b - \left[b^2 - 4ac\right]^{1/2}}{2aX_1 + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right.$$

$$\left. - \ln \left| \frac{2a(V_{DD} + V_{TD}) + b - \left[b^2 - 4ac\right]^{1/2}}{2a(V_{DD} + V_{TD}) + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right] \cdot \frac{2\,C_L}{U_n C_o V_{DD} W_1/L_1}$$

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,

$$t_{pl} = \int_{V_{DD}}^{V_i - V_{TE}} - \frac{2\,C_L}{U_n C_o W_1/L_1} \cdot \frac{dV_o}{(V_{DD} - V_o)^2 - 2(-V_{TD})(V_{DD} - V_o) + k(V_i - V_{TE})^2}$$

$$= \frac{V_{DD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}} \cdot \left[\tan^{-1}\left(\frac{V_{DD} - (V_i - V_{TE}) + V_{TD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}} \right) \right.$$

$$\left. - \tan^{-1}\left(\frac{V_{TD}}{\left[k(V_i - V_{TE})^2 - V_{TD}^2\right]^{1/2}} \right) \right] \cdot \frac{2\,C_L}{U_n C_o V_{DD} W_1/L_1}$$

When both transistors are in linear region,

$$t_{p2} = \left(\int_{V_i - V_{TE}}^{X_{.5}} \frac{dV_o}{(k-1)V_o^2 + (2V_{TD} + 2V_{DD} - 2k(V_i - V_{TE}))V_o + (-2V_{TD}V_{DD} - V_{DD}^2)} \right)$$

$$\cdot \frac{2\,C_L}{U_n C_o W_1/L_1}$$

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

$$t_{p2} = \frac{V_{DD}}{\left[b^2 - 4ac\right]^{1/2}} \cdot \left[\ln \left| \frac{2aX_{.5} + b - \left[b^2 - 4ac\right]^{1/2}}{2aX_{.5} + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right.$$

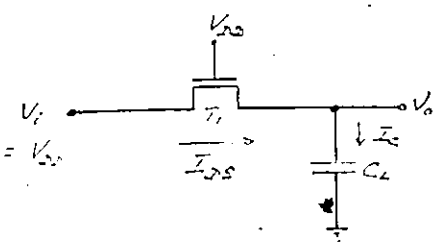
$$\left. - \ln \left| \frac{2a(V_i - V_{TE}) + b - \left[b^2 - 4ac\right]^{1/2}}{2a(V_i - V_{TE}) + b + \left[b^2 - 4ac\right]^{1/2}} \right| \right] \cdot \frac{2\,C_L}{U_n C_o V_{DD} W_1/L_1}$$

The total fall delay,

$$t_{pHL} = t_{p1} + t_{p2}$$

Appendix C

DERIVATION OF TRANSITION AND DELAY TIMES OF A PASS TRANSISTOR



Rise Time

Transistor T_1 is in saturation region.

$$I_{DS} = \frac{U_n C_o}{2} \cdot \frac{W_p}{L_p} \cdot (V_{DD} - V_o - V_{TP})^2$$

$$= \frac{U_n C_o}{2} \cdot \frac{W_p}{L_p} \cdot (V_f - V_o)^2$$

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

$$C_L \frac{dV_o}{dt} = \frac{U_n C_o}{2} \cdot \frac{W_p}{L_p} \cdot (V_f - V_o)^2$$

$$t_r = \int_{X_1}^{X_2} \frac{2 C_L}{U_n C_o W_p / L_p} \cdot \frac{dV_o}{(V_f - V_o)^2}$$

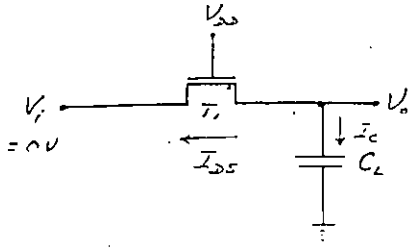
$$= \frac{2 C_L}{U_n C_o W_p / L_p} \cdot \left[\frac{1}{V_f - V_o} \right]_{X_1}^{X_2}$$

$$t_r = \frac{2\,C_L}{U_n C_o W_p / L_p} \cdot \left(\frac{1}{V_f - X_2} - \frac{1}{V_f - X_1} \right)$$

Rise Delay

$$t_{pLH} = \int_{V_L}^{X_{.5}} \frac{2\,C_L}{U_n C_o W_p / L_p} \cdot \frac{dV_o}{(V_f - V_o)^2}$$

$$= \frac{2\,C_L}{U_n C_o W_p / L_p} \cdot \left(\frac{1}{V_f - X_{.5}} - \frac{1}{V_f - V_L} \right)$$



$V_o(0) = V_f$

$V_{TP} > 0$

Fall Time

Transistor T_1 is in linear region.

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

$$= \frac{U_n C_o}{2} \cdot \frac{W_p}{L_p} \cdot (2V_f V_o - V_o^2)$$

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

$$C_L \frac{dV_o}{dt} = \frac{U_n C_o}{2} \cdot \frac{W_p}{L_p} \cdot (V_o^2 - 2V_f V_o)$$

$$t_f = \int_{X_2}^{X_1} \frac{2 C_L}{U_n C_o W_p / L_p} \cdot \frac{dV_o}{(V_o^2 - 2V_f V_o)}$$

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

$$= \frac{C_L}{U_n C_o V_f W_p / L_p} \cdot \left[\ln \left(\frac{2V_f - V_o}{V_o} \right) \right]_{X_2}^{X_1}$$

$$t_f = \frac{C_L}{U_n C_o V_f W_p / L_1} \cdot \left[\ln\left(\frac{2V_f - X_1}{X_1} \right) - \ln\left(\frac{2V_f - X_2}{X_2} \right) \right]$$

Fall Delay

$$t_{pHL} = \int_{V_H}^{X_{.5}} \frac{2 C_L}{U_n C_o W_p / L_p} \cdot \frac{dV_o}{(V_o^2 - 2V_f V_o)}$$

$$= \frac{C_L}{U_n C_o V_f W_p / L_p} \cdot \left[\ln\left(\frac{2 V_f - X_{.5}}{X_{.5}} \right) - \ln\left(\frac{2 V_f - V_H}{V_H} \right) \right]$$

Appendix D

PROGRAM LISTING

```
>ASCAL 8000/2.0B     AAEC  (17FEB82)

    1   0680 --    PROGRAM simulation (input,output);
    2   06A8 --
    3   06A8 --    CONST
    4   06A8 --       nodemax = 100 ;
    5   06A8 --       transmax = 100 ;
    6   06A8 --       Z = 50 ;
    7   06A8 --       Z2 = 100 ;
    8   06A8 --       UC = 2.0E-5 ;
    9   06A8 --       VDD = 5 ;
   10   06A8 --       VTE = 1 ;
   11   06A8 --       VTD = -4 ;
   12   06A8 --
   13   06A8 --    TYPE
   14   06A8 --       nodenumber = 1..nodemax;
   15   06A8 --       transnumber = 1..transmax;
   16   06A8 --       statevalue = 0..2;
   17   06A8 --       strengvalue = 1..3;
   18   06A8 --       ntype = 1..2;
   19   06A8 --       ttype = 1..3;
   20   06A8 --
   21   06A8 --       inputlink = @inputnode ;
   22   06A8 --       inputnode =
   23   06A8 --          RECORD
   24   06A8 --             fptr : inputlink ;
   25   06A8 --             node : integer
   26   06A8 --          END ;
   27   06A8 --
   28   06A8 --       pathlink = @pathnode ;
   29   06A8 --       pathnode =
   30   06A8 --          RECORD
   31   06A8 --             fptr : pathlink ;
   32   06A8 --             node : integer ;
   33   06A8 --             no : integer
   34   06A8 --          END ;
   35   06A8 --
   36   06A8 --       tilink = @timing;
   37   06A8 --       timing =
   38   06A8 --          RECORD
   39   06A8 --             fptr, bptr : tilink;
   40   06A8 --             time : integer;
   41   06A8 --             newstate : statevalue;
   42   06A8 --             tsi : real
   43   06A8 --          END;
   44   06A8 --
   45   06A8 --       trlink = @transint;
   46   06A8 --       transint =
   47   06A8 --          RECORD
   48   06A8 --            fptr : trlink;
   49   06A8 --            trans : integer
   50   06A8 --          END;
   51   06A8 --
   52   06A8 --       glink = @gate;
   53   06A8 --       gate =
   54   06A8 --          RECORD
   55   06A8 --             node : integer;
   56   06A8 --          END;
   57   06A8 --
   58   06A8 --       slink = @source;
```

```
 59   06A8 --        source =
 60   06A8 --           RECORD
 61   06A8 --             node : integer
 62   06A8 --           END;
 63   06A8 --
 64   06A8 --        dlink = @drain;
 65   06A8 --        drain =
 66   06A8 --           RECORD
 67   06A8 --             node : integer
 68   06A8 --           END;
 69   06A8 --
 70   06A8 --        hlink = @headernode ;
 71   06A8 --        headernode =
 72   06A8 --           RECORD
 73   06A8 --              fptr : hlink ;
 74   06A8 --              node : integer
 75   06A8 --           END ;
 76   06A8 --
 77   06A8 --        plink = @plist ;
 78   06A8 --        plist =
 79   06A8 --           RECORD
 80   06A8 --              fptr : plink ;
 81   06A8 --              gr : integer
 82   06A8 --            END ;
 83   06A8 --
 84   06A8 --        evlink = @evinfo ;
 85   06A8 --        evinfo =
 86   06A8 --           RECORD
 87   06A8 --              fptr : evlink ;
 88   06A8 --              node : integer ;
 89   06A8 --              newsta : statevalue ;
 90   06A8 --              time : integer ;
 91   06A8 --              tt : real
 92   06A8 --           END ;
 93   06A8 --
 94   06A8 --        overlink = @overflow ;
 95   06A8 --        overflow =
 96   06A8 --           RECORD
 97   06A8 --              fptr,bptr : overlink ;
 98   06A8 --              node : integer ;
 99   06A8 --              newsta : statevalue ;
100   06A8 --              time : integer ;
101   06A8 --              tt : real
102   06A8 --           END ;
103   06A8 --
104   06A8 --        pendlink = @pend ;
105   06A8 --        pend =
106   06A8 --           RECORD
107   06A8 --              fptr : pendlink ;
108   06A8 --              group : integer ;
109   06A8 --           END ;
110   06A8 --
111   06A8 --        graph = ARRAY[0..nodemax] OF hlink ;
112   06A8 --        gte = ARRAY[0..10] OF statevalue ;
113   06A8 --        gth = ARRAY[0..10] OF strengvalue ;
114   06A8 --        list = ARRAY[1..Z2] OF evlink ;
115   06A8 --        pathline = ARRAY[1..10] OF pathlink ;
116   06A8 --
```

```
117   06A8 --        nodeinfo =
118   06A8 --           RECORD
119   06A8 --             tiptr : tilink ;
120   06A8 --             fcptr,icptr : trlink ;
121   06A8 --             trigger,visit,inev : boolean ;
122   06A8 --             state,newsta : statevalue ;
123   06A8 --             strength : strengvalue ;
124   06A8 --             nodetype : nftype ;
125   06A8 --             capvalue : real ;
126   06A8 --             fc,ic : integer ;
127   06A8 --             W,L : real ;
128   06A8 --             group : integer
129   06A8 --           END;
130   06A8 --
131   06A8 --        transinfo =
132   06A8 --           RECORD
133   06A8 --             gptr : glink ;
134   06A8 --             dptr : 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   3588 --        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 : tilink ;
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   40D8 --        t,tm    : integer ;
157   40E0 --        tt : real ;
158   40E8 --        i,j,k,v,w : integer ;
159   40FC --        sta : statevalue ;
160   4100 --        str : strengvalue ;
161   4104 --        unit : boolean ;
162   4108 --        pullup : boolean ;
163   410C --        q,ii,pn : integer ;
164   4118 --        path : pathline ;
165   4140 --
166   4140 --
167   4140 -- A   PROCEDURE search ( VAR first : hlink ;
168   0040 --                                i : integer ;
169   0040 --                                X0,X1,XX : statevalue ) ;
170   0054 --
171   0054 --     (* search for an adjacent node *)
172   0054 --
173   0054 --     VAR
174   0054 --        l : trlink ;
```

```
  175   0058 --          base : hlink ;
  176   005C --          j,k : integer ;
  177   0064 --
  178   0000 0- A     BEGIN
  179   0012 --          node[i].visit := true ;
  180   0032 --          l := node[i].icptr ;
  181   005C --          IF l <> NIL THEN
  182   006E --             WHILE l <> NIL DO
  183   0080 1-                BEGIN
  184   0080 --                   j := l@.trans ;
  185   0098 --                   IF (trans[j].state = X0) OR (trans[j].state = X1)
  186   0114 --                   OR (trans[j].state = XX) THEN
  187   015C 2-                      BEGIN
  188   015C --                         IF (trans[j].dptr@.node <> i) AND
  189   01A4 --                            (node[trans[j].dptr@.node].visit = false)
  190   01FC --                         THEN
  191   0204 3-                            BEGIN
  192   0204 --                               k := trans[j].dptr@.node ;
  193   023A --                               new(base) ;
  194   0248 --                               base@.fptr := first ;
  195   0268 --                               base@.node := k ;
  196   0280 --                               first := base ;
  197   0296 --                               search(first,k,X0,X1,XX)
  198   02E2 -3                            END
  199   02E6 --                         ELSE
  200   02EA --                            IF (trans[j].sptr@.node <> i) AND
  201   0332 --                               (node[trans[j].sptr@.node].visit = false)
  202   038A --                            THEN
  203   0392 3-                               BEGIN
  204   0392 --                                  k := trans[j].sptr@.node ;
  205   03C8 --                                  new(base) ;
  206   03D6 --                                  base@.fptr := first ;
  207   03F6 --                                  base@.node := k ;
  208   040E --                                  first := base ;
  209   0424 --                                  search(first,k,X0,X1,XX)
  210   0470 -3                               END ;
  211   0474 -2                      END ;
  212   0474 --                l := l@.fptr
  213   047E -1                END
  214   0490 -0 A     END ;
  215   04AC --
  216   04AC -- A     PROCEDURE partition ( VAR header : graph ;
  217   0040 --                               index : integer ;
  218   0040 --                               X0,X1,XX : statevalue ) ;
  219   0054 --
  220   0054 --       (* partition an undirected graph into its connected components *
  221   0054 --
  222   0054 --       VAR
  223   0054 --          start,delete,hd : hlink ;
  224   0060 --          i : integer ;
  225   0064 --
  226   0000 0- A     BEGIN
  227   0012 --          start := header[index] ;
  228   003A --          WHILE start <> NIL DO
  229   004C 1-             BEGIN
  230   004C --                node[start@.node].visit := false ;
  231   0074 --                start := start@.fptr
  232   007E -1             END;
```

```
PASCAL 8000/2.0B     AAEC  (17FEB82)

  233   0094 --          start := header[index] ;
  234   00BC --          m := 0 ;
  235   00C8 --          WHILE start <> NIL DO
  236   00DA 1-             BEGIN
  237   00DA --               IF node[start@.node].visit = false THEN
  238   010E 2-                  BEGIN
  239   010E --                    m := m+1 ;
  240   012C --                    new(hd) ;
  241   013A --                    hd@.fptr := NIL ;
  242   014A --                    hd@.node := start@.node ;
  243   016C --                    i := start@.node ;
  244   0184 --                    search(hd,i,X0,X1,XX) ;
  245   01D4 --                    header[m] := hd ;
  246   0202 -2                  END ;
  247   0202 --               start := start@.fptr
  248   020C -1             END
  249   021E -0 A   END;
  250   023C --
  251   023C -- A   PROCEDURE appendev ( x : integer ;
  252   0040 --                           nesta : statevalue ;
  253   0040 --                           toccur : integer ;
  254   0040 --                           tsi : real ) ;
  255   0058 --
  256   0058 --     (* adding an event to the TQ array *)
  257   0058 --
  258   0058 --     VAR
  259   0058 --        event : evlink ;
  260   005C --        ev,newev : overlink ;
  261   0064 --        j : integer ;
  262   0068 --
  263   0000 0- A   BEGIN
  264   0012 --        j := toccur - ( Z * MTC - 1 ) ;
  265   0040 --        IF j <= Z2 THEN
  266   0052 1-           BEGIN
  267   0052 --              new(event) ;
  268   0060 --              WITH event@ DO
  269   006C 2-                 BEGIN
  270   006C --                    fptr := TQ[j] ;
  271   0096 --                    node := x ;
  272   00A4 --                    newsta := nesta ;
  273   00B6 --                    time := toccur ;
  274   00C4 --                    tt := tsi
  275   00C4 -2                 END ;
  276   00D6 --              TQ[j] := event
  277   00E8 -1           END
  278   0100 --        ELSE
  279   0104 1-           BEGIN
  280   0104 --              new(newev) ;
  281   0112 --              WITH newev@ DO
  282   011E 2-                 BEGIN
  283   011E --                    node := x ;
  284   012C --                    newsta := nesta ;
  285   013E --                    time := toccur ;
  286   014C --                    tt := tsi
  287   014C -2                 END ;
  288   015E --              ev := MTEL ;
  289   0176 2-              REPEAT
  290   0176 --                 ev := ev@.bptr
```

```
291  0180 -2              UNTIL newev@.time >= ev@.time ;
292  01C0 --              newev@.fptr := ev@.fptr ;
293  01E6 --              newev@.bptr := ev ;
294  0202 --              ev@.fptr@.bptr := newev ;
295  0228 --              ev@.fptr := newev
296  0232 -1          END
297  0244 -0 A  END ;
298  0260 --
299  0260 -- A  PROCEDURE spike ( i : integer ;
300  0040 --                       nesta : statevalue ;
301  0040 --                       toc : integer ;
302  0040 --                       ts : real ) ;
303  0058 --
304  0058 --    (* checking for spike *)
305  0058 --
306  0058 --    VAR
307  0058 --       pi,tim : tilink ;
308  0060 --       pointer : evlink ;
309  0064 --       ev : overlink ;
310  0068 --       stop : boolean ;
311  006C --       j : integer ;
312  0070 --
313  0000 0- A  BEGIN
314  0012 --       new(tim) ;
315  0020 --       WITH tim@ DO
316  002C 1-          BEGIN
317  002C --             time := toc ;
318  003A --             newstate := nesta ;
319  004C --             tsi := ts
320  004C -1          END ;
321  005E --       stop := false ;
322  0064 --       pi := node[i].tiptr ;
323  008E 1-       REPEAT
324  008E --          pi := pi@.bptr ;
325  00AA --          IF ( tim@.time <= pi@.time ) AND ( pi <> pi@.fptr ) THEN
326  0110 2-             BEGIN
327  0110 --              IF tim@.newstate <> pi@.newstate THEN
328  013E 3-                 BEGIN
329  013E --                    pi@.newstate := tim@.newstate ;
330  0164 --                    tim@.newstate := 2 ;
331  0176 --                    j := pi@.time - ( Z * MTC - 1 ) ;
332  01AE --                    IF j <= Z2 THEN
333  01C0 4-                       BEGIN
334  01C0 --                          pointer := TQ[j] ;
335  01EA --                          WHILE pointer <> NIL DO
336  01FC 5-                             BEGIN
337  01FC --                                IF pointer@.node = i THEN
338  0220 6-                                   BEGIN
339  0220 --                                      pointer@.newsta := pi@.newstate
340  0246 --                                      pointer := NIL
341  0246 -6                                   END
342  024C --                                ELSE
343  0250 --                                   pointer := pointer@.fptr
344  025A -5                             END
345  026C -4                       END
346  0270 --                    ELSE
347  0274 4-                       BEGIN
348  0274 --                          ev := MTEL ;
```

```
349   028C 5-                          REPEAT
350   028C --                             ev := ev@.bptr
351   0296 -5                          UNTIL ev@.node = i ;
352   02CC --                          ev@.newsta := pi@.newstate
353   02E0 -4                       END
354   02F2 -3                    END
355   02F2 -2                 END
356   02F2 --              ELSE
357   02F6 2-                 BEGIN
358   02F6 --                    tim@.fptr := pi@.fptr ;
359   031C --                    tim@.bptr := pi ;
360   0338 --                    pi@.fptr@.bptr := tim ;
361   035E --                    pi@.fptr := tim ;
362   037A --                    appendev(i,tim@.newstate,toc,ts) ;
363   03C8 --                    stop := true
364   03C8 -2                 END
365   03D0 -1           UNTIL stop = true
366   03DA -0 A      END ;
367   0408 --
368   0408 -- A      PROCEDURE readparameter ;
369   0040 --
370   0040 --        (* read simulation parameters *)
371   0040 --
372   0040 --        VAR
373   0040 --           pi : tilink ;
374   0044 --           base,p : trlink ;
375   004C --           g : glink ;
376   0050 --           d : dlink ;
377   0054 --           s : slink ;
378   0058 --           numnode,numtrans : integer ;
379   0060 --           tc,ti,i,t,num : integer ;
380   0074 --           sta : statevalue ;
381   0078 --           str : strengvalue ;
382   007C --           nty : ntype ;
383   0080 --           tty : ttype ;
384   0084 --           wid,len,cap : real ;
385   00A0 --
386   0000 0  A      BEGIN
387   0012 --           network[0] := NIL ;
388   001E --           readln(numnode,numtrans) ;
389   0042 --           FOR num := 1 TO numnode DO
390   006E 1-              BEGIN
391   006E --                 readln(i,sta,str,nty,cap,tc,ti,wid,len) ;
392   0104 --                 WITH node[i] DO
393   0124 2-                    BEGIN
394   0124 --                       new(h[0]) ;
395   0138 --                       h[0]@.node := i ;
396   0156 --                       h[0]@.fptr := network[0] ;
397   017E --                       network[0] := h[0] ;
398   019C --                       appendev(i,sta,0,0) ;
399   01DC --                       state := 2 ;
400   01E4 --                       newsta := 2 ;
401   01EC --                       strength := str ;
402   01FE --                       nodetype := nty ;
403   0210 --                       capvalue := cap ;
404   0222 --                       fc := tc ;
405   0230 --                       ic := ti ;
406   023E --                       W  := wid ;
```

```
407   0250 --                  L  := len ;
408   0262 --                  inev := false ;
409   0268 --                  new(pi) ;
410   0276 --                  pi@.fptr := pi ;
411   0292 --                  pi@.bptr := pi ;
412   02AE --                  pi@.time := 0 ;
413   02BE --                  pi@.newstate := 2 ;
414   02D0 --                  pi@.tsi := 0 ;
415   02F0 --                  tiptr := pi ;
416   0302 --                  IF fc > 0 THEN
417   0314 3-                     BEGIN
418   0314 --                        base := NIL ;
419   031A --                        FOR t := 1 TO tc DO
420   0346 4-                           BEGIN
421   0346 --                              new(p) ;
422   0354 --                              read(p@.trans) ;
423   0370 --                              p@.fptr := base ;
424   038C --                              base := p
425   038C -4                           END ;
426   03C0 --                        fcptr := p
427   03C0 -3                     END
428   03D2 --                  ELSE fcptr := NIL ;
429   03DC --                  IF ic > 0 THEN
430   03EE 3-                     BEGIN
431   03EE --                        base := NIL ;
432   03F4 --                        FOR t := 1 TO ic DO
433   0420 4-                           BEGIN
434   0420 --                              new(p) ;
435   042E --                              read(p@.trans) ;
436   044A --                              p@.fptr := base ;
437   0466 --                              base := p
438   0466 -4                           END ;
439   049A --                        icptr := p
440   049A -3                     END
441   04AC --                  ELSE icptr := NIL
442   04B0 -2               END
443   04B6 -1           END ;
444   04D8 --       FOR num:=1 TO numtrans DO
445   0504 1-         BEGIN
446   0504 --            readln(i,tty,wid,len) ;
447   054A --            WITH trans[i] DO
448   056A 2-               BEGIN
449   056A --                  state := 2 ;
450   0572 --                  trigger := false ;
451   0578 --                  transtype := tty ;
452   058A --                  W := wid ;
453   059C --                  L := len ;
454   05AE --                  new(g) ;
455   05BC --                  read(g@.node) ;
456   05D8 --                  gptr := g ;
457   05EA --                  new(d) ;
458   05F8 --                  read(d@.node) ;
459   0614 --                  dptr := d ;
460   0626 --                  new(s) ;
461   0634 --                  read(s@.node) ;
462   0650 --                  sptr := s
463   0650 -2               END
464   0662 -1        END
```

```
  465   0662 -0 A  END ;
  466   06A8 --
  467   06A8 -- A  PROCEDURE state ( front : graph ;
  468   0040 --                      ind : integer ;
  469   0040 --                      VAR grstate : statevalue ;
  470   0040 --                      VAR grstrength : strengvalue ) ;
  471   01E0 --
  472   01E0 --    (* compute the state and strength of the group *)
  473   01E0 --
  474   01E0 --    VAR
  475   01E0 --       k,l : (N,W,P,D) ;
  476   01E8 --       y : statevalue ;
  477   01EC --       i : integer ;
  478   01F0 --
  479   0000 0- A  BEGIN
  480   0012 --       k := N ;      (* null strength *)
  481   0018 --       y := 2 ;        (* X-state *)
  482   0020 --       WHILE front[ind] <> NIL DO
  483   0044 1-          BEGIN
  484   0044 --             i := front[ind]@.node ;
  485   006E --             IF node[i].strength = 1 THEN
  486   0098 --                l := W
  487   0098 --             ELSE
  488   00A4 --                IF node[i].strength = 2 THEN
  489   00CE --                   l := P
  490   00CE --                ELSE
  491   00DA --                   IF node[i].strength = 3 THEN
  492   0104 --                      l := D ;
  493   010C --             IF l > k THEN
  494   0126 2-                BEGIN
  495   0126 --                   k := l ;
  496   0134 --                   IF k = p THEN
  497   0146 --                      y := 1
  498   0146 --                   ELSE
  499   0152 --                      y := node[i].state
  500   016A -2                END
  501   017C --             ELSE
  502   0180 --                IF ( l = k ) AND ( node[i].state <> y ) THEN
  503   01E0 --                   y := 2 ;
  504   01E8 --             front[ind] := front[ind]@.fptr
  505   0216 -1          END ;
  506   022C --       grstate := y ;
  507   0242 --       IF k = N THEN
  508   0254 --          writeln(' error in strength value ')
  509   026C --       ELSE
  510   0276 --          IF k = W THEN grstrength := 1
  511   0288 --          ELSE
  512   0298 --             IF k = P THEN grstrength := 2
  513   02AA --             ELSE
  514   02BA --                IF k = D THEN grstrength := 3
  515   02CC -0 A  ENd ;
  516   0314 --
  517   0314 -- A  PROCEDURE poison ( VAR poilist : plink ;
  518   0040 --                      groups : graph ;
  519   0040 --                      gstr : gth ;
  520   0040 --                      pm : integer ) ;
  521   0208 --
  522   0208 --    (* poilist is expanded to include other poisoned groups *)
```

```
523   0208 --
524   0208 --     VAR
525   0208 --        pvector : ARRAY[1..3] OF plink ;
526   0214 --        p1,p2,p3,pp : plink ;
527   0224 --        l : trlink ;
528   0228 --        pq : hlink ;
529   022C --        i,j,k,q,s,t,v : integer ;
530   0248 --        found : boolean ;
531   024C --
532   0000 0- A   BEGIN
533   0012 --        pp := poilist ;
534   0028 --        pvector[1] := NIL ;
535   002E --        pvector[2] := NIL ;
536   0034 --        pvector[3] := NIL ;
537   003A --        WHILE pp <> NIL DO
538   004C 1-          BEGIN
539   004C --            IF gstr[pp@.gr] = 1 THEN
540   007A 2-              BEGIN
541   007A --                new(p1) ;
542   0088 --                p1@.gr := pp@.gr ;
543   00AA --                p1@.fptr := pvector[1] ;
544   00C6 --                pvector[1] := p1
545   00C6 -2              END
546   00D8 --            ELSE
547   00DC --              IF gstr[pp@.gr] = 2 THEN
548   010A 2-                BEGIN
549   010A --                  new(p2) ;
550   0118 --                  p2@.gr := pp@.gr ;
551   013A --                  p2@.fptr := pvector[2] ;
552   0156 --                  pvector[2] := p2
553   0156 -2                END
554   0168 --              ELSE
555   016C 2-                BEGIN
556   016C --                  new(p3) ;
557   017A --                  p3@.gr := pp@.gr ;
558   019C --                  p3@.fptr := pvector[3] ;
559   01B8 --                  pvector[3] := p3
560   01B8 -2                END ;
561   01CA --            pp := pp@.fptr
562   01D4 -1          END ;
563   01EA --        s := 1 ;
564   01F2 --        FOR t := 3 DOWNTO s DO
565   021C --          WHILE pvector[t] <> NIL DO
566   0240 1-            BEGIN
567   0240 --              v := pvector[t]@.gr ;
568   026A --              pvector[t] := pvector[t]@.fptr ;
569   02AA --              WHILE groups[v] <> NIL DO
570   02CE 2-                BEGIN
571   02CE --                  j := groups[v]@.node ;
572   02F8 --                  l := node[j].icptr ;
573   0322 --                  IF l <> NIL THEN
574   0334 --                    WHILE l <> NIL DO
575   0346 3-                      BEGIN
576   0346 --                        i := l@.trans ;
577   035E --                        IF ( trans[i].state = 2 ) THEN
578   038E 4-                          BEGIN
579   038E --                            IF ( trans[i].dptr@.node = j ) THEN
580   03D0 --                              k := trans[i].sptr@.node
```

```
581   03F8 --                                ELSE
582   040A --                                  k := trans[i].dptr@.node ;
583   0440 --                                FOR q := 1 TO pm DO
584   046C 5-                                  BEGIN
585   046C --                                    pq := groups[q] ;
586   0490 --                                    WHILE pq <> NIL DO
587   04A2 6-                                      BEGIN
588   04A2 --                                        IF k = pq@.node THEN
589   04C6 7-                                          BEGIN
590   04C6 --                                            IF ( gstr[q] < gstr[v] ) THEN
591   0504 8-                                              BEGIN
592   0504 --                                                new(pp) ;
593   0512 --                                                pp@.gr := q ;
594   0524 --                                                pp@.fptr := poilist ;
595   0544 --                                                poilist := pp ;
596   055A --                                                gstr[q] := gstr[v] ;
597   0590 --                                                new(pp) ;
598   059E --                                                pp@.gr := q ;
599   05B0 --                                                pp@.fptr := pvector[t] ;
600   05DE --                                                pvector[t] :=pp
601   05F0 -8                                              END
602   0602 --                                            ELSE
603   0606 --                                              IF ( gstr[q] = gstr[v] )
604   064A --                                              THEN
605   0644 8-                                                BEGIN
606   0644 --                                                  pp := poilist ;
607   065A --                                                  found := false ;
608   0660 --                                                  WHILE pp <> NIL DO
609   0672 9-                                                    BEGIN
610   0672 --                                                      IF q = pp@.gr THEN
611   068E 0-                                                        BEGIN
612   068E --                                                          found := true ;
613   0696 --                                                          pp := NIL
614   0696 -0                                                        END
615   069C --                                                      ELSE
616   06A0 --                                                        pp := pp@.fptr
617   06AA -9                                                    END ;
618   06C0 --                                                  IF found = false THEN
619   06D2 9-                                                    BEGIN
620   06D2 --                                                      new(pp) ;
621   06E0 --                                                      pp@.gr := q ;
622   06F2 --                                                      pp@.fptr := poilist ;
623   0712 --                                                      poilist := pp ;
624   0728 --                                                      gstr[q] := gstr[v] ;
625   075E --                                                      new(pp) ;
626   076C --                                                      pp@.gr := q ;
627   077E --                                                      pp@.fptr := pvector[t]
628   07AC --                                                      pvector[t] := pp
629   07BE -9                                                    END
630   07D0 -8                                                END
631   07D0 -7                                            END ;
632   07D0 --                                          pq := pq@.fptr
633   07DA -6                                        END
634   07EC -5                                    END
635   07F0 -4                                  END;
636   0812 --                                  l := l@.fptr
637   081C -3                                END ;
638   0832 --                          groups[v] := groups[v]@.fptr
```

```
  639   0860 -2                END
  640   0872 -1              END
  641   0876 -0 A  END ;
  642   08BC --
  643   08BC -- A  PROCEDURE nodestate ( head : graph ;
  644   0040 --                           inx : integer ) ;
  645   01D8 --
  646   01D8 --    (* compute the node states of the subnetwork *)
  647   01D8 --
  648   01D8 --    VAR
  649   01D8 --       ptr : hlink ;
  650   01DC --       p,pptr : plink ;
  651   01E4 --       gstate : gte ;
  652   0210 --       gstrength : gth ;
  653   023C --       i,r    : integer ;
  654   0244 --
  655   0000 0- A  BEGIN
  656   0012 --       state(head ,inx ,sta,str) ;
  657   004A --       gstate[0] := sta ;
  658   0062 --       gstrength[0] := str ;
  659   007A --       partition(head,inx,1,1,1) ;
  660   00AC --       IF m > 1 THEN
  661   00C4 1-         BEGIN
  662   00C4 --           pptr := NIL ;
  663   00CA --           r := m ;
  664   00DE --           WHILE r > 0 DO
  665   00F0 2-             BEGIN
  666   00F0 --               state(head ,r,sta,str) ;
  667   0128 --               gstate[r] := sta ;
  668   0152 --               gstrength[r] := str ;
  669   017C --               IF gstate[r] <> gstate[0] THEN
  670   01A8 3-                 BEGIN
  671   01A8 --                   new(p) ;
  672   01B6 --                   p@.gr := r ;
  673   01CE --                   p@.fptr := pptr ;
  674   01EA --                   pptr := p
  675   01EA -3                 END ;
  676   01FC --               r := r - 1
  677   0206 -2             END ;
  678   0212 --           poison(pptr,head ,gstrength,m) ;
  679   024A --           WHILE pptr <> NIL DO
  680   025C 2-             BEGIN
  681   025C --               gstate[pptr@.gr] := 2 ;
  682   0280 --               pptr := pptr@.fptr
  683   028A -2             END ;
  684   02A0 --           FOR r := 1 TO m DO
  685   02D2 --             WHILE  head[r] <> NIL DO
  686   02F6 2-               BEGIN
  687   02F6 --                 i :=  head[r]@.node ;
  688   0320 --                 node[i].trigger := false ;
  689   033E --                 IF ( node[i].strength <> 3 ) THEN
  690   0368 --                   IF ( node[i].state <> gstate[r] ) THEN
  691   03AC 3-                     BEGIN
  692   03AC --                       node[i].newsta := gstate[r] ;
  693   03E8 --                       node[i].trigger := true
  694   0400 -3                     END
  695   0408 --                   ELSE
  696   040C --                     IF ( node[i].newsta <> gstate[r] ) THEN
```

```
 697   0450 3-                        BEGIN
 698   0450 --                          node[i].newsta := gstate[r] ;
 699   048C --                          node[i].trigger := true
 700   04A4 -3                        END ;
 701   04AC --                    head[r] := head[r]@.fptr
 702   04DA -2                  END
 703   04EC -1              END
 704   0512 --            ELSE
 705   0516 --              WHILE head[inx] <> NIL DO
 706   053A 1-                BEGIN
 707   053A --                  i := head[inx]@.node ;
 708   0564 --                  node[i].trigger := false ;
 709   0582 --                  IF ( node[i].strength <> 3 ) THEN
 710   05AC --                    IF ( node[i].state <> gstate[0] ) THEN
 711   05DE 2-                    BEGIN
 712   05DE --                      node[i].newsta := gstate[0] ;
 713   0608 --                      node[i].trigger := true
 714   0620 -2                    END
 715   0628 --                    ELSE
 716   062C --                      IF ( node[i].newsta <> gstate[0] ) THEN
 717   065E 2-                      BEGIN
 718   065E --                        node[i].newsta := gstate[0] ;
 719   0688 --                        node[i].trigger := true
 720   06A0 -2                      END ;
 721   06A8 --                  head[inx] := head[inx]@.fptr
 722   06D6 -1                END
 723   06E8 -0 A  END ;
 724   0718 --
 725   0718 -- A  PROCEDURE append ( VAR addl : hlink ;
 726   0040 --                       VAR list : graph ;
 727   0040 --                       x : integer ) ;
 728   004C --
 729   004C --    (* adding an activated group to the list EVL *)
 730   004C --
 731   004C --    VAR
 732   004C --       ad : hlink ;
 733   0050 --       evpen : pendlink ;
 734   0054 --       found : boolean ;
 735   0058 --
 736   0000 0- A  BEGIN
 737   0012 --       found := false ;
 738   0018 --       evpen := evpend ;
 739   0030 --       WHILE evpen <> NIL DO
 740   0042 --          IF evpen@.group = x THEN
 741   0066 1-             BEGIN
 742   0066 --                found := true ;
 743   006E --                evpen := NIL
 744   006E -1             END
 745   0074 --          ELSE
 746   0078 --             evpen := evpen@.fptr ;
 747   0098 --       IF found = false THEN
 748   00AA 1-          BEGIN
 749   00AA --             new(evpen) ;
 750   00B8 --             evpen@.group := x ;
 751   00D0 --             evpen@.fptr := evpend ;
 752   00F2 --             evpend := evpen ;
 753   010A --             ad := list[x] ;
 754   0132 --             WHILE ad@.fptr <> NIL DO
```

```
PASCAL 8000/2.0B     AAEC  (17FEB82)

  755   014E --                ad := ad@.fptr ;
  756   016E --               ad@.fptr := addl ;
  757   018E --               addl := list[x] ;
  758   01BA --               list[x] := NIL
  759   01CC -1             END
  760   01D6 -0 A  END ;
  761   01E8 --
  762   01E8 -- A  PROCEDURE delete ( VAR del : hlink ) ;
  763   0044 --
  764   0044 --    (* delete those used storage area *)
  765   0044 --
  766   0044 --    VAR
  767   0044 --       dele : hlink ;
  768   0048 --
  769   0000 0- A  BEGIN
  770   0012 --       WHILE del <> NIL DO
  771   0028 1-          BEGIN
  772   0028 --             dele := del ;
  773   003E --             del := del@.fptr ;
  774   0062 --             dispose(dele)
  775   0072 -1          END
  776   0072 -0 A  END ;
  777   0084 --
  778   0084 -- A  PROCEDURE searchpath ( VAR path : pathline ;
  779   0040 --                           VAR q : integer ;
  780   0040 --                           m : integer ) ;
  781   004C --
  782   004C --    (* search for direct paths between the input and
  783   004C --       the output nodes *)
  784   004C --
  785   004C --    VAR
  786   004C --      l : trlink ;
  787   0050 --      tem,base  : pathlink ;
  788   0058 --      pass : boolean ;
  789   005C --      i,j,k : integer ;
  790   0068 --
  791   0000 0- A  BEGIN
  792   0012 --      node[m].visit := true ;
  793   0032 --      l := node[m].icptr ;
  794   005C --      IF node[m].ic >= 2 THEN
  795   0086 1-        BEGIN
  796   0086 --          new(tem) ;
  797   0094 --          tem@ := path[q]@ ;
  798   00CC --          pass := false ;
  799   00D2 --          FOR i := 1 TO ( node[m].ic ) DO
  800   0116 2-            BEGIN
  801   0116 --              j := l@.trans ;
  802   012E --              IF (trans[j].trigger = true) AND
  803   0164 --              (node[trans[j].gptr@.node].tiptr@.fptr@.time = ST) THEN
  804   01E6 3-                BEGIN
  805   01E6 --                  ii := trans[j].gptr@.node ;
  806   0222 --                  new(inp) ;
  807   0236 --                  inp@.node := ii ;
  808   025A --                  inp@.fptr := inpu ;
  809   0282 --                  inpu := inp
  810   0282 -3                END ;
  811   02A0 --              IF (trans[j].state = 2) OR (trans[j].state = 1) THEN
  812   0314 3-                BEGIN
```

```
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   03DC --                       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   04EE --                       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 --
852   07D8 -- A   PROCEDURE paths ( head : hlink ) ;
853   0044 --
854   0044 --     (* searching for all the direct paths *)
855   0044 --
856   0044 -      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 ;
```

```
871   0098 --              pullup := true ;
872   00A6 --              point := NIL
873   00A6 -2            END
874   00AC --          ELSE
875   00B0 --            point := point@.fptr
876   00BA -1        END ;
877   00D0 --      IF pullup = false THEN
878   00E8 1-        BEGIN
879   00E8 --          point := head ;
880   00FA --          WHILE point <> NIL DO
881   010C 2-            BEGIN
882   010C --              i := point@.node ;
883   0124 --              IF node[i].strength = 3 THEN
884   014E 3-                BEGIN
885   014E --                  pn := i ;
886   0162 --                  IF node[i].inev = true THEN
887   018C 4-                    BEGIN
888   018C --                      ii := i ;
889   01A0 --                      new(inp) ;
890   01B4 --                      inp@.node := ii ;
891   01D8 --                      inp@.fptr := inpu ;
892   0200 --                      inpu := inp
893   0200 -4                    END ;
894   021E --                  point := NIL
895   021E -3                END
896   0224 --              ELSE
897   0228 --                point := point@.fptr
898   0232 -2            END
899   0244 -1        END ;
900   0248 --      point := head ;
901   025A --      WHILE point <> NIL DO
902   026C 1-        BEGIN
903   026C --          node[point@.node].visit := false ;
904   0294 --          point := point@.fptr
905   029E -1        END ;
906   02B4 --      q := 0 ;
907   02C0 --      node[pn].visit := true ;
908   02E6 --      IF (node[pn].trigger = true) AND
909   031C --         (node[pn].nodetype = 2) THEN
910   035A 1-        BEGIN
911   035A --          q := q + 1 ;
912   0378 --          new(path[q]) ;
913   03A4 --          path[q]@.fptr := NIL ;
914   03D2 --          path[q]@.node := pn ;
915   040E --          path[q]@.no := 0
916   0436 -1        END ;
917   043C --      l := node[pn].icptr ;
918   046C --      FOR i := l TO node[pn].ic DO
919   04B6 1-        BEGIN
920   04B6 --          j := l@.trans ;
921   04CE --          IF (trans[j].trigger = true) AND
922   0504 --          (node[trans[j].gptr@.node].tiptr@.fptr@.time = ST) THEN
923   0586 2-            BEGIN
924   0586 --              ii := trans[j].gptr@.node ;
925   05C2 --              new(inp) ;
926   05D6 --              inp@.node := ii ;
927   05FA --              inp@.fptr := inpu ;
928   0622 --              inpu := inp
```

```
 929   0622 -2            END ;
 930   0640 --          IF trans[j].state <> 0 THEN
 931   0670 2-            BEGIN
 932   0670 --              q := q + 1 ;
 933   068E --              new(path[q]) ;
 934   06BA --              path[q]@.fptr := NIL ;
 935   06E8 --              path[q]@.node := pn ;
 936   0724 --              IF (trans[j].dptr@.node <> pn) AND
 937   0772 --              (node[trans[j].dptr@.node].visit = false ) THEN
 938   07D2 3-                BEGIN
 939   07D2 --                  k := trans[j].dptr@.node ;
 940   0808 --                  new(base) ;
 941   0816 --                  base@.fptr := path[q] ;
 942   0850 --                  base@.node := k ;
 943   0868 --                  path[q] := base ;
 944   0898 --                  searchpath(path,q,k)
 945   08BA -3                END
 946   08BE --              ELSE
 947   08C2 --                IF (trans[j].sptr@.node <> pn) AND
 948   0910 --                (node[trans[j].sptr@.node].visit = false) THEN
 949   0970 3-                  BEGIN
 950   0970 --                    k := trans[j].sptr@.node ;
 951   09A6 --                    new(base) ;
 952   09B4 --                    base@.fptr := path[q] ;
 953   09EE --                    base@.node := k ;
 954   0A06 --                    path[q] := base ;
 955   0A36 --                    searchpath(path,q,k)
 956   0A58 -3                  END
 957   0A5C -2            END ;
 958   0A5C --          l := l@.fptr
 959   0A66 -1        END
 960   0A78 -0 A  END ;
 961   0ACC --
 962   0ACC -- A  PROCEDURE delays ( head : hlink ) ;
 963   0044 --
 964   0044 --    (* computing the delay for each output node which has
 965   0044 --       changed state *)
 966   0044 --
 967   0044 --    VAR
 968   0044 --      pi : tilink ;
 969   0048 --      storage,base,pa,pp : pathlink ;
 970   0058 --      first,point : hlink ;
 971   0060 --      l : trlink ;
 972   0064 --      i,j,k,r,di,beta,delay : integer ;
 973   0080 --      totcap,tot,tstp,tsis,tin,ratio,dratio,tdel : real ;
 974   00C0 --      occur : boolean ;
 975   00C4 --
 976   0000 0- A  BEGIN
 977   0012 --      first := head ;
 978   0024 --      WHILE first <> NIL DO
 979   0036 1-        BEGIN
 980   0036 --          IF (node[first@.node].trigger = true) AND
 981   0070 --          (node[first@.node].nodetype = 2) THEN
 982   00B2 2-            begin
 983   00B2 --            IF (node[first@.node].newsta = 2) THEN
 984   00E6 3-              BEGIN          (* if the new state is X *)
 985   00E6 --                delay := 1 ;
 986   00EE --                tm := ST + delay ;
```

```
 987   0114 --            tt := 0 ;
 988   0130 --            spike(first@.node,node[first@.node].newsta,tm,tt)
 989   01A8 -3          END
 990   01AC --        ELSE
 991   01B0 --          IF (node[first@.node].newsta = 1) THEN
 992   01E4 3-            BEGIN       (* if the new state is 1 *)
 993   01E4 --              paths(head) ;
 994   01FA --              r := 0 ;
 995   0200 --              FOR i := 1 TO q DO
 996   0232 --                IF (node[path[i]@.node].strength = 3 ) OR
 997   0284 --                (( node[path[i]@.node].strength = 1 ) AND
 998   02D6 --                (node[path[i]@.node].nodetype <> 2 )) THEN
 999   0332 --                  path[i] := NIL
1000   0344 --                ELSE
1001   0354 4-                  BEGIN
1002   0354 --                    r := r + 1 ;
1003   0366 --                    path[r] := path[i] ;
1004   03A8 --                    IF path[r]@.node = first@.node THEN
1005   03EE --                      di := r
1006   03EE -4                  END ;
1007   041E --              FOR i := 1 TO r DO
1008   044A 4-                BEGIN
1009   044A --                  j := 0 ;
1010   0450 --                  pa := path[i] ;
1011   047A --                  WHILE pa@.fptr <> NIL DO
1012   0496 5-                    BEGIN
1013   0496 --                      j := j + 1 ;
1014   04A8 --                      pa := pa@.fptr
1015   04B2 -5                    END ;
1016   04C8 --                  pa := path[i] ;
1017   04F2 --                  WHILE pa <> NIL DO
1018   0504 5-                    BEGIN
1019   0504 --                      pa@.no := j ;
1020   051C --                      j := j - 1 ;
1021   052E --                      pa := pa@.fptr
1022   0538 -5                    END
1023   054A -4                END ;
1024   0570 --              IF pullup = true THEN
1025   0588 4-                BEGIN
1026   0588 --                  totcap := 0 ;
1027   059E --                  storage := NIL ;
1028   05A4 --                  FOR i := 1 TO r DO
1029   05D0 5-                    BEGIN
1030   05D0 --                      pa := path[i] ;
1031   05FA --                      WHILE pa <> NIL DO
1032   060C 6-                        BEGIN
1033   060C --                          pp := storage ;
1034   061E --                          occur := false ;
1035   0624 --                          WHILE pp <> NIL DO
1036   0636 --                            IF pa@.node <> pp@.node THEN
1037   0664 --                              pp := pp@.fptr
1038   066E --                            ELSE
1039   0684 7-                              BEGIN
1040   0684 --                                occur := true ;
1041   068C --                                pp := NIL
1042   068C -7                              END ;
1043   0696 --                          IF occur = false THEN
1044   06A8 7-                            BEGIN
```

```
1045  06A8 --                                  new(base) ;
1046  06B6 --                                  base@.node := pa@.node ;
1047  06D8 --                                  base@.no := pa@.no ;
1048  06FA --                                  base@.fptr := storage ;
1049  0716 --                                  storage := base ;
1050  0728 --                                  totcap := ( 1 / sqrt(pa@.no + 1) ) *
1051  076E --                                    node[pa@.node].capvalue + totcap
1052  07A0 -7                                END ;
1053  07B4 --                              pa := pa@.fptr
1054  07BE -6                            END
1055  07D0 -5                          END ;
1056  07F6 --                        ratio := node[pn].W / node[pn].L ;
1057  0854 --                        tstp := 3.6976 * ( 1 / ( UC * VDD ) ) *
1058  0886 --                          ( 1 / ratio ) * totcap ;
1059  08C0 --                        pi := node[ii].tiptr ;
1060  08F0 5-                        REPEAT
1061  08F0 --                          pi := pi@.bptr
1062  08FA -5                        UNTIL pi@.time = ST ;
1063  0936 --                        IF pi@.tsi <> 0 THEN
1064  0966 --                         tsis := ( 1 + 0.3 * (pi@.tsi / tstp) ** 1.5 ) *
1065  09BC --                                 tstp
1066  09BC --                        ELSE
1067  09D4 --                         tsis := tstp ;
1068  09E6 --                        tdel := tsis / sqrt( sqr(2.9) + 0.15 *
1069  09FA --                                sqr(node[ii].capvalue / totcap) )
1070  0A48 -4                      END
1071  0A4E --                    ELSE
1072  0A52 4-                      BEGIN
1073  0A52 --                        pi := node[ii].tiptr ;
1074  0A82 5-                        REPEAT
1075  0A82 --                          pi := pi@.bptr
1076  0A8C -5                        UNTIL pi@.time = ST ;
1077  0AC8 --                        tsis := pi@.tsi ;
1078  0AE4 --                        tdel := 0
1079  0AE4 -4                      END ;
1080  0AFA --                    IF path[di]@.no = 0 THEN
1081  0B2E 4-                      BEGIN
1082  0B2E --                       tm := ST + round(tdel * 1.0E9) ;
1083  0B86 --                       tt := tsis ;
1084  0BBE --                       spike(first@.node,node[first@.node].newsta,tm,tt)
1085  0C16 -4                      END
1086  0C1A --                    ELSE
1087  0C1E 4-                      BEGIN
1088  0C1E --                       totcap := 0 ;
1089  0C34 --                       tin := tsis ;
1090  0C46 --                       pa := path[di] ;
1091  0C70 --                       WHILE pa <> NIL DO
1092  0C82 5-                        BEGIN
1093  0C82 --                          IF pa@.no <> 0 THEN
1094  0C9E --                            totcap := totcap + sqrt(pa@.no) *
1095  0CDA --                                      node[pa@.node].capvalue ;
1096  0D12 --                          pa := pa@.fptr
1097  0D1C -5                        END ;
1098  0D32 --                       FOR i := 1 TO r DO
1099  0D5E --                         IF i <> di THEN
1100  0D70 5-                          BEGIN
1101  0D70 --                            tot := 0 ;
1102  0D86 --                            pa := path[i] ;
```

```
1103   0DB0 --                    WHILE pa <> NIL DO
1104   0DC2 6-                      BEGIN
1105   0DC2 --                        pp := path[di] ;
1106   0DEC --                        occur := false ;
1107   0DF2 --                        WHILE pp <> NIL DO
1108   0E04 --                          IF pa@.node <> pp@.node THEN
1109   0E32 --                            pp := pp@.fptr
1110   0E3C --                          ELSE
1111   0E52 7-                            BEGIN
1112   0E52 --                              occur := true ;
1113   0E5A --                              pp := NIL
1114   0E5A -7                            END;
1115   0E64 --                        IF occur = false THEN
1116   0E76 7-                          BEGIN
1117   0E76 --                            tot := node[pa@.node].capvalue +
1118   0EA6 --                                   tot ;
1119   0EBA --                            pa := pa@.fptr
1120   0EC4 -7                          END
1121   0ED6 --                        ELSE
1122   0EDA 7-                          BEGIN
1123   0EDA --                            totcap := sqrt(pa@.no) * tot +
1124   0F16 --                                      totcap ;
1125   0F2A --                            pa := NIL
1126   0F2A -7                          END
1127   0F30 -6                    END
1128   0F30 -5                  END ;
1129   0F56 --                pa := path[di] ;
1130   0F80 --                l := node[pa@.node].icptr ;
1131   0FB4 --                ratio := trans[l@.trans].W / trans[l@.trans].L ;
1132   1026 --                tstp := 17.778 * ( 1/( UC * ( VDD - VTE ) ) ) *
1133   105C --                        sqrt(path[di]@.no) * ( 1 / ratio ) *
1134   10C8 --                        totcap ;
1135   10DC --                IF tin <> 0 THEN
1136   1102 --                  tsis := ( 1 + 0.3 * ( tin / tstp ) ** 1.5 ) *
1137   114E --                          tstp
1138   114E --                ELSE
1139   1166 --                  tsis := tstp ;
1140   1178 --                tdel := tsis / 8.9 ;
1141   118E --                tm := ST + round(tdel * 1.0E9) ;
1142   11E6 --                tt := tsis ;
1143   11FE --                i := first@.node ;
1144   1216 --                spike(i,node[i].newsta,tm,tt)
1145   127A -4              END
1146   127E -3          END
1147   127E --        ELSE
1148   1282 3-          BEGIN       (* if the new state is 0 *)
1149   1282 --            paths(head) ;
1150   1298 --            r := 0 ;
1151   129E --            dratio := 0 ;
1152   12B4 --            FOR i := 1 TO q DO
1153   12E6 4-              BEGIN
1154   12E6 --                ratio := 0 ;
1155   12FC --                IF (node[path[i]@.node].strength = 3) THEN
1156   1348 5-                  BEGIN
1157   1348 --                    pa := path[i] ;
1158   1372 --                    WHILE pa <> NIL DO
1159   1384 6-                      BEGIN
1160   1384 --                        IF pa@.node <> pn THEN
```

```
1161   13AE 7-                          BEGIN
1162   13AE --                            l := node[pa@.node].icptr ;
1163   13E2 --                            WHILE l <> NIL DO
1164   13F4 8-                              BEGIN
1165   13F4 --                                IF (trans[l@.trans].dptr@.node
1166   1426 --                                    = pa@.fptr@.node) OR
1167   145A --                                    (trans[l@.trans].sptr@.node
1168   148C --                                    = pa@.fptr@.node) THEN
1169   14C8 --                                  ratio :=( trans[l@.trans].L /
1170   14FE --                                            trans[l@.trans].W )
1171   1536 --                                          + ratio ;
1172   154A --                                l := l@.fptr
1173   1554 -8                              END
1174   1566 -7                          END ;
1175   156A --                        pa := pa@.fptr
1176   1574 -6                      END ;
1177   158A --                    dratio := ( dratio + ( 1 / ratio ) ) ;
1178   15C2 --                    path[i] := NIL
1179   15D4 -5                  END
1180   15E0 --                ELSE
1181   15E4 5-                  begin
1182   15E4 --                  IF (node[path[i]@.node].strength = 1) AND
1183   1636 --                  (node[path[i]@.node].nodetype <> 2) THEN
1184   1690 --                    path[i] := NIL
1185   16A2 --                  ELSE
1186   16B2 6-                    BEGIN
1187   16B2 --                      r := r + 1 ;
1188   16C4 --                      path[r] := path[i] ;
1189   1706 --                      IF path[r]@.node = first@.node THEN
1190   174C --                        di := r
1191   174C -6                    END
1192   175A -5                end
1193   175A -4              END ;
1194   177C --            FOR i := 1 TO r DO
1195   17A8 4-              BEGIN
1196   17A8 --                j := 0 ;
1197   17AE --                pa := path[i] ;
1198   17D8 --                WHILE pa@.fptr <> NIL DO
1199   17F4 5-                  BEGIN
1200   17F4 --                    j := j + 1 ;
1201   1806 --                    pa := pa@.fptr
1202   1810 -5                  END ;
1203   1826 --                pa := path[i] ;
1204   1850 --                WHILE pa <> NIL DO
1205   1862 5-                  BEGIN
1206   1862 --                    pa@.no := j ;
1207   187A --                    j := j - 1 ;
1208   188C --                    pa := pa@.fptr
1209   1896 -5                  END
1210   18A8 -4              END ;
1211   18CE --            If pullup = true THEN
1212   18E6 4-              BEGIN
1213   18E6 --                totcap := 0 ;
1214   18FC --                storage := NIL ;
1215   1902 --                FOR i := 1 TO r DO
1216   192E 5-                  BEGIN
1217   192E --                    pa := path[i] ;
1218   1958 --                    WHILE pa <> NIL DO
```

```
1219  196A 6-                BEGIN
1220  196A --                  pp := storage ;
1221  197C --                  occur := false ;
1222  1982 --                  WHILE pp <> NIL DO
1223  1994 --                    IF pa@.node <> pp@.node THEN
1224  19C2 --                      pp := pp@.fptr
1225  19CC --                    ELSE
1226  19E2 7-                      BEGIN
1227  19E2 --                        occur := true ;
1228  19EA --                        pp := NIL
1229  19EA -7                      END;
1230  19F4 --                  IF occur = false THEN
1231  1A06 7-                    BEGIN
1232  1A06 --                      new(base) ;
1233  1A14 --                      base@.node := pa@.node ;
1234  1A36 --                      base@.no := pa@.no ;
1235  1A58 --                      base@.fptr := storage ;
1236  1A74 --                      storage := base ;
1237  1A86 --                      totcap := ( 1 / sqrt(pa@.no + 1) ) *
1238  1ACC --                                node[pa@.node].capvalue +
1239  1AFE --                                totcap
1240  1AFE -7                    END ;
1241  1B12 --                  pa := pa@.fptr
1242  1B1C -6                END
1243  1B2E -5              END ;
1244  1B54 --            ratio := node[pn].W / node[pn].L ;
1245  1BB2 --            beta := round(dratio / ratio) ;
1246  1BFE --            IF ( ( beta >= 2 ) AND ( beta < 6 ) ) THEN
1247  1C36 --              tstp := 4.0624 * ( 1 / ( UC * VDD ) ) *
1248  1C68 --                      ( 1 / dratio ) * totcap
1249  1C8E --            ELSE
1250  1CA6 --              IF ( ( beta >= 6 ) AND ( beta < 10 ) ) THEN
1251  1CDE --                tstp := 3.8758 * ( 1 / ( UC * VDD ) ) *
1252  1D10 --                        ( 1 / dratio ) * totcap
1253  1D36 --              ELSE
1254  1D4E --                writeln(' beta =  ',beta,' error ') ;
1255  1D96 --            pi := node[ii].tiptr ;
1256  1DC6 5-            REPEAT
1257  1DC6 --              pi := pi@.bptr
1258  1DD0 -5            UNTIL pi@.time = ST ;
1259  1E0C --            IF pi@.tsi <> 0 THEN
1260  1E3C --              tsis := ( 1 + 0.25 * (pi@.tsi / tstp) ** 1.1 )
1261  1E92 --                      * tstp
1262  1E92 --            ELSE
1263  1EAA --              tsis := tstp ;
1264  1EBC --            tdel := tsis / sqrt( sqr(1.9) + 0.06 *
1265  1ED0 --                    sqr(node[ii].capvalue / totcap) )
1266  1F1E -4        END
1267  1F24 --      ELSE
1268  1F28 4-        BEGIN
1269  1F28 --          pi := node[ii].tiptr ;
1270  1F58 5-          REPEAT
1271  1F58 --            pi := pi@.bptr
1272  1F62 -5          UNTIL pi@.time = ST ;
1273  1F9E --          tsis := pi@.tsi ;
1274  1FBA --          tdel := 0
1275  1FBA -4        END ;
1276  1FD0 --      IF path[di]@.no = 0 THEN
```

```
1277  2004 4-                    BEGIN
1278  2004 --                     tm := ST + round(tdel * 1.0E9) ;
1279  205C --                     tt := tsis ;
1280  2074 --                     spike(first@.node,node[first@.node].newsta,tm,tt)
1281  20EC -4                    END
1282  20F0 --                  ELSE
1283  20F4 4-                    BEGIN
1284  20F4 --                     totcap := 0 ;
1285  210A --                     tin := tsis ;
1286  211C --                     pa := path[di] ;
1287  2146 --                     WHILE pa <> NIL DO
1288  2158 5-                       BEGIN
1289  2158 --                       IF pa@.no <> 0 THEN
1290  2174 --                         totcap := totcap + sqrt(pa@.no) *
1291  21B0 --                                   node[pa@.node].capvalue ;
1292  21E8 --                       pa := pa@.fptr
1293  21F2 -5                     END ;
1294  2208 --                     FOR i := 1 TO r DO
1295  2234 --                       IF i <> di THEN
1296  2246 5-                         BEGIN
1297  2246 --                           tot := 0 ;
1298  225C --                           pa := path[i] ;
1299  2286 --                           WHILE pa <> NIL DO
1300  2298 6-                             BEGIN
1301  2298 --                               pp := path[di] ;
1302  22C2 --                               occur := false ;
1303  22C8 --                               WHILE pp <> NIL DO
1304  22DA --                                 IF pa@.node <> pp@.node THEN
1305  2308 --                                   pp := pp@.fptr
1306  2312 --                                 ELSE
1307  2328 7-                                   BEGIN
1308  2328 --                                     occur := true ;
1309  2330 --                                     pp := NIL
1310  2330 -7                                   END ;
1311  233A --                               IF occur = false THEN
1312  234C 7-                                 BEGIN
1313  234C --                                   tot := node[pa@.node].capvalue +
1314  237C --                                          tot ;
1315  2390 --                                   pa := pa@.fptr
1316  239A -7                                 END
1317  23AC --                               ELSE
1318  23B0 7-                                 BEGIN
1319  23B0 --                                   totcap := sqrt(pa@.no) * tot +
1320  23EC --                                             totcap ;
1321  2400 --                                   pa := NIL
1322  2400 -7                                 END
1323  2406 -6                           END
1324  2406 -5                       END ;
1325  242C --                     pa := path[di] ;
1326  2456 --                     l := node[pa@.node].icptr ;
1327  248A --                     ratio := trans[l@.trans].W / trans[l@.trans].L ;
1328  24FC --                     tstp := 2.7436 * ( 1/( UC * ( VDD - VTE ) ) ) *
1329  2532 --                             sqrt(path[di]@.no) * ( 1 / ratio ) *
1330  259E --                             totcap ;
1331  25B2 --                     IF tin <> 0 THEN
1332  25D8 --                       tsis := ( 1 + ( tin / tstp ) ** 0.7 ) *
1333  2620 --                               tstp
1334  2620 --                     ELSE
```

```
1335   263C --                          tsis := tstp ;
1336   264E --                        tdel := tsis / 2.5 ;
1337   2664 --                        tm := ST + round(tdel * 1.0E9) ;
1338   26BC --                        tt := tsis ;
1339   26D4 --                        i := first@.node ;
1340   26EC --                        spike(i,node[i].newsta,tm,tt)
1341   2750 -4                      END
1342   2754 -3                  END
1343   2754 -2                end ;
1344   2754 --                first := first@.fptr
1345   275E -1              END
1346   2770 -0 A  END ;
1347   286C --
1348   286C --
1349   286C --
1350   286C -- A  PROCEDURE simulate ;
1351   0040 --
1352   0040 --    (* perform the simulation process *)
1353   0040 --
1354   0040 --    VAR
1355   0040 --       TQempty,stop,outev,share : boolean ;
1356   0050 --       i,k,delay : integer ;
1357   005C --
1358   0000 0- A  BEGIN
1359   0012 1-      REPEAT
1360   0012 --       evpend := NIL ;
1361   001E --       EVL := NIL ;
1362   002A --       ST := Z * MTC + t - 1 ;
1363   0064 --       IF TQ[t] <> NIL THEN
1364   0094 2-         BEGIN
1365   0094 --           WHILE TQ[t] <> NIL DO
1366   00C4 3-             BEGIN
1367   00C4 --               i := TQ[t]@.node ;
1368   00FA --               IF ( node[i].state <> TQ[t]@.newsta ) THEN
1369   0154 4-                 BEGIN
1370   0154 --                   node[i].state := TQ[t]@.newsta ;
1371   01A6 --                   IF ( node[i].strength = 3 ) AND ( node[i].ic > 0 )
1372   0206 --                   THEN
1373   020E 5-                     BEGIN
1374   020E --                       node[i].inev := true ;
1375   022E --                       j := node[i].group ;
1376   025A --                       append(EVL,subnet,j)
1377   0282 -5                     END ;
1378   0286 --                   fp := node[i].fcptr ;
1379   02B6 --                   WHILE fp <> NIL DO
1380   02CE 5-                     BEGIN
1381   02CE --                       IF ( trans[fp@.trans].transtype = 1 ) THEN
1382   030E 6-                         BEGIN
1383   030E --                           trans[fp@.trans].trigger := true ;
1384   0344 --                           trans[fp@.trans].state := node[i].state
1385   0384 -6                         END
1386   039C --                       ELSE
1387   03A0 --                         IF ( trans[fp@.trans].transtype = 2 ) THEN
1388   03E0 6-                           BEGIN
1389   03E0 --                             trans[fp@.trans].trigger := true ;
1390   0416 --                             IF ( node[i].state = 0 ) THEN
1391   0440 --                               trans[fp@.trans].state := 1
1392   0468 --                             ELSE
```

```
1393   047A --                              IF ( node[i].state = 1 ) THEN
1394   04A4 --                                trans[fp@.trans].state := 0
1395   04CC --                              ELSE
1396   04DC --                                trans[fp@.trans].state := 2
1397   0504 -6                            END
1398   0512 --                          ELSE
1399   0516 6-                            BEGIN
1400   0516 --                              trans[fp@.trans].trigger := true ;
1401   054C --                              trans[fp@.trans].state := 1
1402   0574 -6                            END ;
1403   0582 --                        v := trans[fp@.trans].dptr@.node ;
1404   05CE --                        w := trans[fp@.trans].sptr@.node ;
1405   061A --                        IF ( node[v].group = node[w].group ) THEN
1406   0670 --                          append(EVL,subnet,node[v].group)
1407   06B0 --                        ELSE
1408   06B8 6-                          BEGIN
1409   06B8 --                            append(EVL,subnet,node[v].group) ;
1410   06FC --                            append(EVL,subnet,node[w].group)
1411   073C -6                          END ;
1412   0740 --                        fp := fp@.fptr
1413   0750 -5                      END
1414   0768 -4                    END ;
1415   076C --                  TQ[t] := TQ[t]@.fptr
1416   07AC -3                END ;
1417   07C8 --              sub[0] := EVL ;
1418   07E6 --              partition(sub,0,1,1,2) ;
1419   0812 --              delete(sub[0]) ;
1420   0820 --              inpu := NIL ;
1421   082C --              FOR k := 1 to m DO
1422   085E 3-                BEGIN
1423   085E --                  n := n + 1 ;
1424   087C --                  subnet[n] := sub[k] ;
1425   08C4 --                  point := subnet[n] ;
1426   08FA --                  WHILE point <> NIL DO
1427   0912 4-                    BEGIN
1428   0912 --                      i := point@.node ;
1429   0930 --                      node[i].group := n ;
1430   095C --                      point := point@.fptr
1431   096C -4                    END ;
1432   0988 --                  nodestate(subnet,n) ;
1433   09B4 --                  point := subnet[n] ;
1434   09EA --                  outev := false ;
1435   09F0 --                  WHILE point <> NIL DO
1436   0A08 4-                    BEGIN
1437   0A08 --                      i := point@.node ;
1438   0A26 --                      IF ( node[i].trigger = true ) THEN
1439   0A50 --                        IF ( node[i].nodetype = 1 ) THEN
1440   0A7A --                          node[i].state := node[i].newsta
1441   0AAA --                        ELSE
1442   0AC0 --                          IF unit = true THEN
1443   0AD8 5-                            BEGIN
1444   0AD8 --                              delay := 1 ;
1445   0AE0 --                              tm := ST + delay ;
1446   0B06 --                              tt := 0 ;
1447   0B22 --                              appendev(i,node[i].newsta,tm,tt)
1448   0B86 -5                            END
1449   0B8A --                          ELSE
1450   0B8E --                            outev := true ;
```

```
1451  0B96 --                point := point@.fptr
1452  0BA6 -4              END ;
1453  0BC2 --            share := true ;
1454  0BCA --            IF ( outev = true ) THEN
1455  0BDC 4-              BEGIN
1456  0BDC --                point := subnet[n] ;
1457  0C12 --                WHILE point <> NIL DO
1458  0C2A 5-                  BEGIN
1459  0C2A --                    IF (node[point@.node].strength = 2) OR
1460  0C6A --                    (node[point@.node].strength = 3) THEN
1461  0CB2 6-                     BEGIN
1462  0CB2 --                       share := false ;
1463  0CB8 --                       point := NIL
1464  0CB8 -6                     END
1465  0CC4 --                   ELSE
1466  0CC8 --                     point := point@.fptr
1467  0CD8 -5                 END ;
1468  0CF4 --               IF share = true THEN
1469  0D06 5-                 BEGIN
1470  0D06 --                   point := subnet[n] ;
1471  0D3C --                   WHILE point <> NIL DO
1472  0D54 6-                    BEGIN
1473  0D54 --                      IF node[point@.node].nodetype = 2 THEN
1474  0D8E 7-                       BEGIN
1475  0D8E --                         i := point@.node ;
1476  0DAC --                         tm := ST + 1 ;
1477  0DCA --                         spike(i,node[i].newsta,tm,0)
1478  0E2C -7                       END ;
1479  0E30 --                     point := point@.fptr
1480  0E40 -6                    END
1481  0E58 -5                 END
1482  0E5C --               ELSE
1483  0E60 --                 delays(subnet[n])
1484  0E90 -4             END
1485  0E94 -3         END ;
1486  0EB6 --       WHILE inpu <> NIL DO
1487  0ECE 3-         BEGIN
1488  0ECE --           i := inpu@.node ;
1489  0EEC --           IF node[i].inev = true THEN
1490  0F16 --             node[i].inev := false ;
1491  0F34 --           pi := node[i].tiptr@.fptr ;
1492  0F6E --           IF pi@.time = ST THEN
1493  0F9E 4-             BEGIN
1494  0F9E --               node[i].tiptr@.fptr := pi@.fptr ;
1495  0FE2 --               pi@.fptr@.bptr := node[i].tiptr
1496  1014 -4             END ;
1497  1026 --           inpu := inpu@.fptr
1498  1036 -3         END ;
1499  1052 --       FOR k := 1 TO m DO
1500  1084 3-         BEGIN
1501  1084 --           point := sub[k] ;
1502  10B4 --           WHILE point <> NIL DO
1503  10CC 4-             BEGIN
1504  10CC --               l := node[point@.node].icptr ;
1505  110C --               WHILE l <> NIL DO
1506  1124 5-                 BEGIN
1507  1124 --                   IF trans[l@.trans].trigger = true THEN
1508  1164 --                     trans[l@.trans].trigger := false ;
```

```
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 --         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   12E4 -2           END ;
1520   12EA --           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   14D8 --                   new(event) ;
1535   14EC --                   WITH event@ DO
1536   14FE 4-                     BEGIN
1537   14FE --                       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   1698 -2             END ;
1549   169C --           i := 1 ;
1550   16A4 --           TQempty := true ;
1551   16AC --           WHILE i <= Z2 DO
1552   16BE --             IF TQ(.i.) = NIL THEN
1553   16E8 --               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 ;
1565   17FC --
1566   17FC --
```

```
1567   17FC --    (* start of the main program *)
1568   17FC --
1569   0000 0-    BEGIN
1570   0066 --       n := 0 ;
1571   0072 --       MTC := 0 ;
1572   007E --       t := 1 ;
1573   008C --       new(MTEL) ;          (* create an empty overflow event list *)
1574   00A0 --       WITH MTEL@ DO
1575   00B2 1-         BEGIN
1576   00B2 --           fptr := MTEL ;
1577   00CA --           bptr := MTEL ;
1578   00E2 --           node := 0 ;
1579   00E8 --           newsta := 0 ;
1580   00EE --           time := 0 ;
1581   00F4 -1         END ;
1582   00F4 --       FOR i := 1 TO Z2 DO       (* create an empty TQ array *)
1583   011E --         TQ[i] := NIL ;
1584   0176 --       readparameter ;
1585   017A --       sub[0] := network[0] ;
1586   0198 --       partition(sub,0,1,1,2) ;
1587   01C4 --       delete(sub[0]) ;
1588   01D2 --       FOR i := 1 TO m DO
1589   021C 1-         BEGIN
1590   021C --           n := n + 1 ;
1591   023A --           subnet[n] := sub[i] ;
1592   0288 --           point := subnet[n] ;
1593   02BE --           WHILE point <> NIL DO
1594   02D6 2-             BEGIN
1595   02D6 --               j := point@.node ;
1596   02FA --               node[j].group := n ;
1597   032C --               point := point@.fptr
1598   033C -2             END
1599   0354 -1         END ;
1600   0392 --       unit := true ;
1601   03A0 --       simulate ;
1602   03A4 --       read(evnum) ;
1603   03BC --       FOR j := 1 TO evnum DO
1604   0406 1-         BEGIN
1605   0406 --           read(i,sta,tm,tt) ;
1606   045A --           new(tim) ;
1607   046E --           WITH tim@ DO
1608   0480 2-             BEGIN
1609   0480 --               time := tm ;
1610   0494 --               newstate := sta ;
1611   04AC --               tsi := tt
1612   04AC -2             END ;
1613   04C4 --           pi := node[i].tiptr ;
1614   04FA 2-           REPEAT
1615   04FA --             pi := pi@.bptr
1616   050A -2           UNTIL tim@.time >= pi@.time ;
1617   055C --           tim@.fptr := pi@.fptr ;
1618   058E --           tim@.bptr := pi ;
1619   05B6 --           pi@.fptr@.bptr := tim ;
1620   05E8 --           pi@.fptr := tim ;
1621   0610 --           appendev(i,sta,tm,tt)
1622   0668 -1         END ;
1623   06A6 --       MTC := 0 ;
1624   06B2 --       t := 1 ;
```

```
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*
```