A Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems

Thesis by

Tzu-Mu Lin

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1985

(Submitted August 2, 1984)

Acknowledgments

Special acknowledgment is due to Professor Carver Mead, my adviser, for his guidance, encouragement and many insights which lead to the result of this research. It has been a most exciting and rewarding period to work under his guidance.

I also wish to thank the following people on whose earlier work this research is based:

Prof. Paul Penfield, Jr. and Dr. Jorge Rubinstein: RC delay model,

Prof. Randy Bryant: switch-level simulation model, and

Prof. Marina Chen: universal hierarchical simulation.

I am grateful to Chao-Lin Chiang for his helpful discussions and experiments on RC delay model and circuit simulation, to John Wawrzynek for his aide and collaboration, to Wen-Chi Chen for his friendship and constant encouragement, and to Ward Cunningham for his support in the implementation of the HITSIM simulator.

I am most indebted to my parents, my wife and her parents for their love, understanding and encouragement.

This work is sponsored by the System Development Foundation.

Abstract

A hierarchical timing simulation model for digital MOS circuits and systems is presented. This model supports the structured design methodology, and can be applied to both "structure" and "behavior" representations of designs in a uniform manner. A simulator based on this model can run several orders of magnitude faster than any other simulators that offer the same amount of information.

At the structure (transistor) level, the transient behavior of a digital MOS circuit is approximated by that of an RC network for estimating delays. The Penfield-Rubinstein RC tree model is extended to include the effects of parallel paths and initial charge distributions. As far as delay is concerned, a two-port RC network is characterized by three parameters: R: series resistance, C: loading capacitance and D: internal delay. These parameters can be determined hierarchically as networks are composed in various ways. The composition rules are derived directly from the Kirchoff's current and voltage laws, so that the consistency with physics is established.

The (R, C, D) characterization of two-port RC networks is then generalized to describe the behavior of semantic cells at any level of representation. A semantic cell is a functional block which can be abstracted by its steady-state behavior to interface with other cells in the system. As semantic cells are composed, the parameters of the composite cell can be determined from those of the the component cells either analytically or by simulation. A Smalltalk implementation of the hierarchical timing simulation model is also presented.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Structured Design Methodology and Hierarchical Simulation

It is by now common knowledge that the evolution of VLSI fabrication technology follows Moore's law — the complexity of commercially available chips doubles every year [34]. With the state-of-art technology, it is possible to integrate hundreds of thousands of transistors in one chip that is functionally as capable as an entire computing system [35]. Many algorithms that are traditionally only implementable by software programs are now built directly on silicon wafers with much faster execution speed [25]. The explosive advances in VLSI technology have generated opportunities for revolutionary system designs.

In order to exploit these opportunities successfully, there must be correspondingly aggressive advances in the supporting technologies and sciences. One question that must be addressed is how to manage the increasing complexity of VLSI systems. Under the well-known structured design methodology [30,32], a design is partitioned into several levels of hierarchy, typically from the architecture level, block level, logic level to circuit level. This partitioning helps designers focus on one particular level of design at any given time, so that the complexity of a large design may be managed effectively.

A hierarchical design is best supported by a hierarchical simulator for determining its functionality and performance. The difficulty of a hierarchical simulator, however, is that consistency between different levels of representation cannot be easily ensured. As pointed out in [32], to ensure this consistency requires a good deal of discipline — in particular, a well-defined and consistent timing convention, and well-defined data types. If these disciplines are followed, then a system can be partitioned successively into hierarchical levels of semantic cells [1]. A semantic cell at any level of representation can be abstracted by its steady-state behavior to interface with other semantic cells. Furthermore, any legal interconnection of several semantic cells is itself a semantic cell, and the behavior of the composite cell can be derived from the behaviors of the component cells in a consistent manner. Based upon the fixed-point algorithm [42] to abstract the behavior of a cell from its implementation, Chen developed a Universal Hierarchical Simulator (UHS) that can be applied to designs from transistor circuit-level to high-level communicating sequential processes [8]. The hierarchical nature of the UHS allows the implementation details to be hidden, and therefore yields a clear conceptualization of the design and a very efficient simulation.

1.2 A Hierarchical Timing Simulation Model

The main concern of the UHS is the functional behavior of a design, not the delay in physical time units. Time delay information is very important to designers because a chip is not correct if it does not run at the desired speed. On the other hand, most simulators that offer time delay information [7,36] tend to carry too much analog detail; no simple abstraction or composition rules have been derived from these simulators to allow hierarchical treatment of a complicated design. As the complexity of VLSI systems increases, the demand becomes more and more urgent for a UHS style simulation model with the capability of generating physical timing relations.

This thesis presents such a timing simulation model. MOS (Metal-Semiconductor-Oxide) is selected as the target technology because of its increasing importance in fabricating digital systems. The general principle, however, is applicable to other technologies, as well. The timing model reported here is distinguished from other timing or circuit

[1] As opposed to syntactic cells, which are only used for ease of specification, and do not provide any abstraction. The results are due to Chen [8,9].

simulation models by one or more of the following features:

1. **High-level Primitives:** Traditional circuit or timing simulators use very low-level primitives, such as node voltages and branch currents in modelling the circuit behavior. The advantage is that these primitives are general enough that, potentially, any design implemented in any technology can be dealt with. However, the computation involved in solving the circuit equations (usually in the form of nonlinear algebraic-differential equations) is so intensive that, practically, these simulators can be applied only to designs of very small size. By concentrating on digital MOS circuits, higher-level primitives are derived and used as the basis of our timing simulation model. "Delay" (D) is the main primitive used. To allow the characterization of a cell independent of its composition environment, two more primitives are also included: driving resistance (R) and loading capacitance (C). In terms of these primitives, the circuit behavior is expressed in the form of linear algebraic equations, which can be solved much more efficiently[2]. Moreover, once the solution is obtained, the circuit behavior is solved for a time interval (length determined by the time constant natural to the circuit), rather than a single instant of time. The R, C and D primitives of our timing model satisfy the following two requirements, which are considered crucial to the successful development of a simulation model.

   - **theoretical soundness:** The primitives are precisely defined. The equations that govern the behavior of these primitives are derived analytically from the fundamental laws that govern the low-level primitives, i.e., the Ohm's law and the Kirchoff's current law (Chapter 2 and 3).

   - **practical applicability:** The circuit dynamics is expressible in terms of these

[2] The R, C and D primitives only model the integral behavior of MOS circuits, without considering the detailed waveforms. The essential information of digital systems such as the timing of signal transitions and glitches are all covered by these primitives.

primitives so that simulation algorithms can be derived. The simulation algorithms presented in this thesis are capable of estimating the circuit behavior (both logic and timing) with reasonable accuracy. On the other hand, its computational complexity is low enough for it to be applied to very complicated VLSI designs (Chapter 4).

2. **Uniformity and Hierarchy:** Our timing model supports both "structure" and "behavior" representations of VLSI designs in a uniform manner. The importance of including this capability has been pointed out by a number of researchers [32,47]. In this thesis, the terms "structure" and "behavior" are used to mean the following two types of representations of a cell:

- **structure representation:** the cell is expressed as a net list of transistors and wires.
- **behavior representation:** the cell is represented by its behavior description, usually in an executable form, without specifying its structure or implementation. Note that behavior representations are associated only with semantic cells (Chapter 5).

In supporting structure representations, the transient behavior of a transistor network is characterized by a set of R, C, D parameters. The values of these parameters can be determined algorithmically as transistors and wires are composed in various ways. The composition algorithms are derived directly from the Kirchoff's voltage and current laws, and thus the consistency of our timing model with physics is established.

Without digging into the detailed implementation, the behavior representation of a cell is used for interfacing with other cells. When several cells are connected together, the behavior of the composite cell can be derived from the behaviors of the component cells based on our timing model. By decomposing a system successively and then recomposing the behavior, the complexity of a complicated design can be managed effectively.

To extend the (R, C, D) characterization of transistor networks to behavior-level representations, the three primitives are expressed as functions rather than as single values. However, the physical meaning of these primitives remain the same. Note that interconnection of cells is established simply by wires, so that the composition algorithms for transistor networks can be applied directly to behavior-level representations. Based on our timing model, the behavior representation of a cell can be derived from its structure counterpart. The level of accuracy guaranteed by these two types of representations is the same (Chapter 5 and 6).

3. Semantic Composition with Syntactic Checking: To guarantee the consistency between the structure and behavior representations, some disciplines must be followed in the partitioning of a system. These disciplines As an example, a set of rules are given in Chapter 5 for partitioning an arbitrary synchronous system so that every partitioned cell is a semantic cell. Under our timing model, the behavior representation of a semantic cell is valid for any legal composition of the cell. To check whether a composition is legal or not, only local and syntactic procedures are required.

1.3 Organization of the Thesis

Three pieces of work are done to establish our hierarchical timing model:

1. a transistor-level (bottom-level, structure-level) model that serves as the basis,
2. a general technique to abstract the behavior and timing of a cell from its structure, and
3. a general composition rule for deriving the behavior and timing of a high-level cell from those of the lower-level component cells.

Chapter 2 presents the transistor-level timing model. The transient behavior of an MOS circuit is approximated by that of an RC network for estimating delays. Three parameters R, C and D are used to represent the delay characteristics of a two-port RC network. As two-port RC networks are composed in various ways, these parameters can be determined in a hierarchical manner. This model extends the RC tree model of Penfield-Rubinstein [38,40], and is capable of handling any general RC network with parallel connections and initial charge distributions.

Chapter 3 introduces the concept of tree decomposition and load redistribution. Based on this concept, a relaxation algorithm for calculating delays in any general RC network is presented. This algorithm uses local information only during the relaxation process, the convergence of which is guaranteed.

Chapter 4 applies the delay calculation algorithms to timing simulation of digital MOS circuits at the transistor level. A prototype simulator, called SDS (Signal Delay Simulator), has been developed. For all the examples (transistor counts ranging from 10 to 100) tested so far, this simulator runs two to three orders of magnitude faster than SPICE [36], and detects all transitions and glitches at approximately the correct time. This ratio grows drastically as the size of circuits increases.

Chapter 5 generalizes the (R, C, D) characterization of two-port RC networks to semantic cells at any level of representation. The behavior of a semantic cell is characterized by a set of R, C, D parameters, the number of which is proportional to the number of connection ports of the cell. As semantic cells are composed, the parameters of the composite cell can be derived from those of the component cells either analytically or by simulation. As an example, the behaviors of a static nMOS PLA and a bit-serial multiplier are derived analytically from the behaviors of their subcomponents. The size of these subcomponents is small, and their behaviors can be easily obtained from their structures.

Chapter 6 discusses a Smalltalk [18,19] implementation of our behavior-level timing simulator (called HITSIM for Hierarchical Timing Simulator). The object oriented programming model of Smalltalk matches with our semantic cell-oriented simulation model. The environment in which users set up and modify the simulation for their designs, investigate the simulation results, etc., can be naturally done using the hierarchical "inspector" of Smalltalk. The internal nodes and substructures of a design can be accessed in the same hierarchical order that the design is specified. All dialogues are by way of the graphical interfaces provided by the Smalltalk system.

Chapter 7 concludes this thesis by presenting an integrated design system that supports the structured design methodology. Areas for further investigation are also discussed.

Chapter 2

Composition of Delay in RC Networks

Modelling digital MOS circuits by RC networks has become a well accepted practice for estimating delays [31,38]. In 1981, Penfield and Rubinstein (P-R) proposed a method to bound the waveforms of nodes in an RC tree network [38,40]. Two approximations are made in the P-R method: 1) modelling the input of transistors by step waveforms, and 2) modelling conducting transistors by linear resistors. Later, Horowitz (H) extended this method to include both effects of slow inputs and nonlinearity of MOS transistors [20,21]. The P-R-H approach is conceptually simple and computationally efficient, and has been incorporated into many timing-analysis programs [23,45].

One deficiency of the work of P-R-H is that only RC tree networks are dealt with, not general RC meshes. Furthermore, the effect of initial charge is only considered for a special case, that an RC tree without any initial charge is driven through another RC tree that is fully charged initially [21]. No generalization is made to deal with networks with arbitrary initial charge distributions. In this chapter and the following, general RC networks with parallel (and bridge) connections and arbitrary initial charge distributions are considered. The two assumptions made by P-R are also assumed during the discussion. The main results of these two chapters have been published in [27,28].

In section 2.1, a timing model for MOS transistor networks is presented. The transient behavior of a transistor network is approximated by that of an RC network for estimating delays. The definition of delay is based upon the Elmore's delay [15], modified to correctly treat non-monotonic responses (section 2.2). This value of delay is shown always to fall within the P-R bounds for RC tree networks. In section 2.3, transmission matrices are used to express the transfer behavior of two-port RC networks. As far as delay is concerned, a two-port RC network is characterized by three parameters: $\overline{R}$: series resistance, $\overline{C}$: effective capacitance, and $\overline{D}$: internal delay. These three parameters can be calculated hierarchically as the corresponding two-port RC networks are composed in various ways. The composition rules agree with those described in [38], except that stored charge is properly taken into consideration. We also add composition rules for parallel connections. An algorithm for calculating delays of all nodes in an arbitrary RC tree network is presented in section 2.4. General RC networks are dealt with in the next chapter.

2.1 Transistor-level Timing Model

The timing model for MOS transistor networks is based upon the switch model proposed by Bryant [4,5]. In this model, a network is represented by a set of transistors $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$, and a set of nodes $\mathcal{O} = \{n_1, n_2, \ldots, n_n\}$. With each node are associated a capacitance and one of three different states corresponding to the node voltage: 1 (high voltage), 0 (low voltage) or X (in transition). The other end of the node capacitor is always connected to the ground, and no floating capacitors are allowed in the network. With each transistor is associated an ON-resistance (or two different values of ON-resistances: one is used in case of pull up and the other is used in case of pull down [20,46]). A transistor may be either ON or OFF depending on the state of the node controlling its gate. A transistor is treated as a resistance equal to its ON-resistance if it is on or to ∞ if it is off. Instead of the order of magnitude (or logic) conductances and capacitances used in Bryant's switch model, precise values of the resistances and capacitances are kept for determining logic levels as well as for estimating delays. Although the capacitance of a node and the resistance of a transistor are voltage-dependent, they are treated as constants here. This approximation is considered adequate for our purpose, since only the delay values are of interest, not the detailed waveforms. The evolution of an MOS circuit is approximated by a sequence of RC networks. Various node capacitors are charged to VDD and discharged to GND through the network. This charging-discharging process may change the state of a node which in turn changes the topology of the RC network. Under the unit delay model which is employed by Bryant and others, all such nodes change state at the same time. In our model, different nodes are charged or discharged at different rates which depend on the topology and the initial charge distribution of the RC network. When the gate node of at least one transistor changes state, a new network results. A partially charged or discharged node which connects to the gate of a transistor does not change the state of that transistor. However, the charge stored in the nodes will be taken into account when the nodes are again charged or discharged through the new network. The whole process continues until the topology of the network no longer changes.

With the approximation introduced above, the problem of estimating the delay of an MOS circuit reduces to that of an RC network. In this thesis, the term "RC network" refers only to those networks that are approximations of an MOS circuit, i.e., resistor networks where there is a capacitor between every node and GND. Note that the approximating RC networks of different transistor groups[1] of an MOS circuit are disconnected, and their delays can be evaluated independently. In our model, an RC network is always driven by one and only one source (VDD or GND) which is referred to as the source of the RC network. The other two possibilities presented below are not considered.

1. Neither VDD nor GND is driving the RC network: this undriven situation may cause static charge sharing among nodes [4].
2. Both VDD and GND are driving the RC network: In most practical situations, one source is dominant over the other with respect to a node; otherwise, a conflict condition occurs and the logic state of the node is unpredictable. Although the presence of the

[1] Two nodes are in the same transistor group if and only if they are laterally connected through transistors. As the size of MOS circuits increases, that of a transistor group remains almost constant.

other source may affect the delay of the response to the dominant one, the effect is usually small, as various experiments indicate. In our model, this RC network is approximated by two independent RC networks, one driven by VDD and the other by GND (detailed in Chapter 4).

A more constructive definition of RC networks is given in section 2.3.

To measure the delay of a node in an RC network, it suffices to consider the normalized case where the node voltage starts from some initial value between 0 and 1, and is driven towards the final value 1. The results obtained in this normalized case are easily adapted to both charging and discharging processes, and to any values of supply voltages. Normalized variables are used throughout the context; that is, V is dimensionless and therefore Q is of the same dimension as C.

The delay values estimated under an RC-based model like ours are relative rather than absolute. In some sense, the values obtained are normalized with respect to the threshold voltage of a certain transistor type. The effect of different threshold voltages of different transistor types are reflected by adjusting the values of their ON-resistances. As introduced in [31], the delay time τ of an inverter (the simplest transistor group) is linearly related to the RC time constant, where R is the ON-resistance of the driving transistor, and C is the load capacitance of the output node. The nonlinear part of the circuit behavior can be absorbed in the coefficient, which can be determined from more detailed circuit analysis or simulation [36]. The delay time is also additive in that the delay of a chain of such inverters can be obtained by summing up the delays of the individual ones. The motivation behind our work is to extend this linear and additive property to more general transistor networks. The resistances of wires, contacts, etc., can be treated in the same manner as transistor ON-resistances. The lumped approximation of these distributed elements are investigated by Chiang [12]. Simulation results based upon this model and their comparison with SPICE outputs are given in section 4.3.

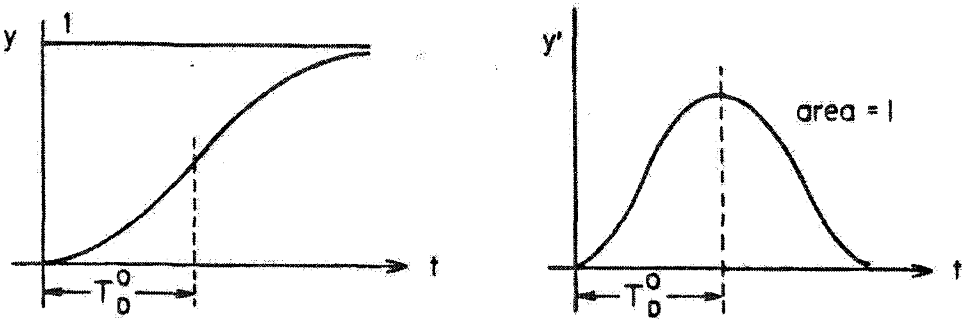


Figure 2.1. Curves illustrating the Elmore's delay

2.2 Definition of Delay

Prior to the actual analysis, it is necessary to have a consistent and unambiguous definition of delay. There are a number of such definitions in practical use, for instance, the time required for a response to reach the threshold voltage of an MOS transistor. Although this kind of definition is useful for certain simulators whose delay calculations are based upon empirical data, it is extremely awkward for theoretical investigation. On the other hand, the Elmore's delay [15] is very efficient in this respect, and it is defined as

$$T_D^0 = \int_0^\infty t y'(t)\, dt. \tag{2.1}$$

where $y'(t)$ is the derivative of the transient response $y(t)$ of some node of a network. The superscript ^{0} indicates zero initial charge, which condition is always assumed by Elmore (also by P-R). This definition of delay is based upon the observation that, if $y(t)$ is monotonic in time, T_D^0 is the centroid of $y'(t)$, and is very close to what is commonly conceived as "delay" (Figure 2.1). The great usefulness of the Elmore's delay lies in its close connection to the Laplace transform $\mathcal{L}$ of the response. In an RC network, $g(s) = \mathcal{L}(y(t))$ can always be expressed in the form $g(s) = \dfrac{1 + a_1 s + a_2 s^2 + \cdots + a_m s^m}{s(1 + b_1 s + b_2 s^2 + \cdots + b_n s^n)}$. Note that

$sg(s)|_{s\to 0} = y(t)|_{t\to\infty} = 1$, because there is no floating capacitor in the network. If there is no initial charge stored in the network, then $T_D^0 = b_1 - a_1$ [15].

Although $g(s)$ is in general a very complicated expression, $T_D^0 = b_1 - a_1$ is very easy to obtain analytically. Penfield and Rubinstein have shown that a general expression of T_D^0 exists for any node in an RC tree network, and the expression can be determined in a hierarchical manner [38]. In this chapter, the result is extended to more general RC networks with parallel connections and nonzero initial charge. To do this, a modification of the Elmore's delay is necessary because the original formulation (2.1) makes sense only when y(t) is monotonic. In an RC tree network without any inital charge, the step response of any node is guaranteed to be monotonic [40]; however, monotonicity is not true in general. To deal with general RC networks, the term delay is redefined as

$$T_D = \int_0^\infty 1 - y(t)dt. \tag{2.1'}$$

This expression is just the area above the response y(t), but below 1, as indicated in Figure 2.1. In the case of zero initial charge, T_D is equal to T_D^0. In [40], this result was proved for the case of RC trees. For general networks, the result is proved as follows:

$$\begin{aligned}\frac{1}{s} - g(s) &= \int_0^\infty [1 - y(t)] \exp^{-st} dt \\ &= T_D - s \cdot \int_0^\infty [1 - y(t)]tdt + \ldots\end{aligned}$$

so that $T_D = \lim_{s\to 0}(\frac{1}{s} - g(s)) = b_1 - a_1 = T_D^0$. ∎

From the above discussion, T_D is consistent with and more general than T_D^0. To justify the usage of T_D, consider the case of RC tree networks with no initial charge. Referring to Figure 2.1, $T_D = T_D^0$ deviates from the standard visualization of delay only when the response curve is highly asymmetric. Fortunately this deviation does not occur in an RC network in the following sense. Suppose that the response curve of a node is approximated by a single exponential function with time constant T_D; then the delay time t_d for the response to reach a threshold voltage v is $T_D \ln(\frac{1}{1-v})$. The value t_d of any node in the network always lies within the upper and lower bounds given by Penfield and Rubinstein [38] for any voltage level v, as the following theorem indicates. These bounds are far tighter than other approximations in the simulation procedure.

Theorem 2.1. Consider a node in an RC tree network with no initial charge. Let t_1, t_2, t_3, t_4 be the four bounds of time defined in [38], i.e.

$$\begin{aligned}
t_1(v) &= T_D - T_P(1-v) \\
t_2(v) &= T_R \ln \frac{T_D}{T_P(1-v)} \\
t_3(v) &= \frac{T_D}{1-v} - T_R \\
t_4(v) &= T_P - T_R + T_P \ln \frac{T_D}{T_P(1-v)}
\end{aligned}$$

where $T_D = T_D^0$ is the Elmore's delay. T_R and T_P are defined in [38], which satisfy $T_R \leq T_D \leq T_P$ for any node in an RC tree network. Let $t_d(v) = T_D \ln(\frac{1}{1-v})$. Then $t_d \geq t_1, t_d \geq t_2, t_d \leq t_3$ for all values of v, and $t_d \leq t_4$ for $v \geq 1 - \frac{T_D}{T_P}$. ▮

The proof of this theorem is given in section 2.5. T_D in case of nonzero stored charge is still consistent with the Elmore's delay T_D^0, as discussed in section 4.2. The usage of T_D is also very effective in detecting glitches produced by the dynamic charge sharing effect.

To understand more about the definition of delay (2.1), consider the voltage of any node e in an RC network (let V_e denote this voltage). Note that V_e can be found by replacing each capacitor in the network by its equivalent current source, $i_n = -C_n \frac{dV_n}{dt}$, and then using linear superposition. The voltage drop of node e due to current i_k equals $-R_{k,e} C_k \frac{dV_k}{dt}$, where $R_{k,e}$ is the mutual resistance between node e and node k. Summing over all capacitors in the network gives

$$V_e = -\sum_k R_{k,e} C_k \frac{dV_k}{dt}. \tag{2.2}$$

One assumption made implicitly in the definition of (2.1) is that

- $\frac{dV_k}{dt}$ for all nodes k in the network are roughly equal to $\frac{dV_e}{dt}$. (2.3)

Substituting $\frac{dV_k}{dt}$ by $\frac{dV_e}{dt}$ in (2.2), V_e can be solved exactly, and the solution is $1 - \exp^{(-t/T_D)}$, an exponential function with time constant equal to

$$T_D = \sum_k R_{k,e} C_k. \tag{2.1''}$$

This definition is equivalent to (2.1). Given a threshold voltage v, the delay of V_e equals $T_D \ln(\frac{1}{1-v})$, which is linearly proportional to the Elmore's delay. In most MOS circuits, the assumption (2.3) is satisfied. For more accurate results, delay models with two or more time constants are required at the cost of much more complicated computations [21]. In this thesis, only single time constant is considered, and the definition of delay (2.1′) is used.

2.3 Composition of Delay Parameters of Two-port RC Networks

A well-known result from circuit theory [14] states that the (voltage-current) transfer behavior of a two-port linear network can be described by the following equation:

$$\begin{pmatrix} V_o(s) \\ I_o(s) \end{pmatrix} = \begin{pmatrix} T_1(s) & T_2(s) \\ T_3(s) & T_4(s) \end{pmatrix} \begin{pmatrix} V_i(s) \\ I_i(s) \end{pmatrix} + \begin{pmatrix} U_1(s) \\ U_2(s) \end{pmatrix} \tag{2.4}$$

where the subscripts $_o$ and $_i$ indicate the output and input ports, respectively. This equation can be expressed either in the time domain or in the Laplace domain; however, it is more convenient to use the Laplace domain, as indicated in section 2.2. The matrices $\begin{pmatrix} T_1(s) & T_2(s) \\ T_3(s) & T_4(s) \end{pmatrix}$ and $\begin{pmatrix} U_1(s) \\ U_2(s) \end{pmatrix}$ are referred to as the T-matrix (transmission matrix) and the U-matrix, respectively. The T-matrix is only a function of the network, while the U-matrix depends on both the network and the initial conditions. In general, $T_i(s)_{i=1,2,3,4}$ and $U_i(s)_{i=1,2}$ are very complicated polynomials in s; however, the delay T_D depends only on the constant and s terms of these polynomials, so higher terms can always be omitted. As shown in Theorem 2.2, the T-matrix and U-matrix of any two-port RC network are characterized, up to the s term, by the following five parameters of the network: the series resistance R, the total capacitance C, the internal delay D due to input, the total stored charge Q, and the internal delay D^* due to stored charge. These five parameters can be determined hierarchically as the corresponding two-port RC networks are composed in various ways. Among these five parameters, R, C, and D are only functions of the network, and Q and D^* also depend on initial charge. As far as delay is concerned, the number of parameters reduces to only three, as indicated by Theorem 2.2′ and Theorem 2.5′. Prior to any further discussion, a more constructive definition of RC networks than the one described in section 2.1 is given. First, a two-port RC network with its input and output ports is recursively defined.

A two-port RC network is one of the following:

1. a resistor in series with a capacitor: The common node of the two is the output, and the other end of the resistor is the input of the network. The other end of the capacitor is grounded (Figure 2.2.a).
2. a series connection of two two-port RC networks N_1 and N_2: The input of N_2 and the output of N_1 merge internally; the input of N_1 becomes the input, and the output of N_2 becomes the output of the resulting network (Figure 2.2.b).
3. a parallel connection of n two-port RC networks $N_{1,\ldots,n}$: The inputs and outputs of these networks merge and become the input and output of the resulting network (Figure 2.2.c).
4. a two-port RC network N_S with a side branch N_L of which the output is open, and

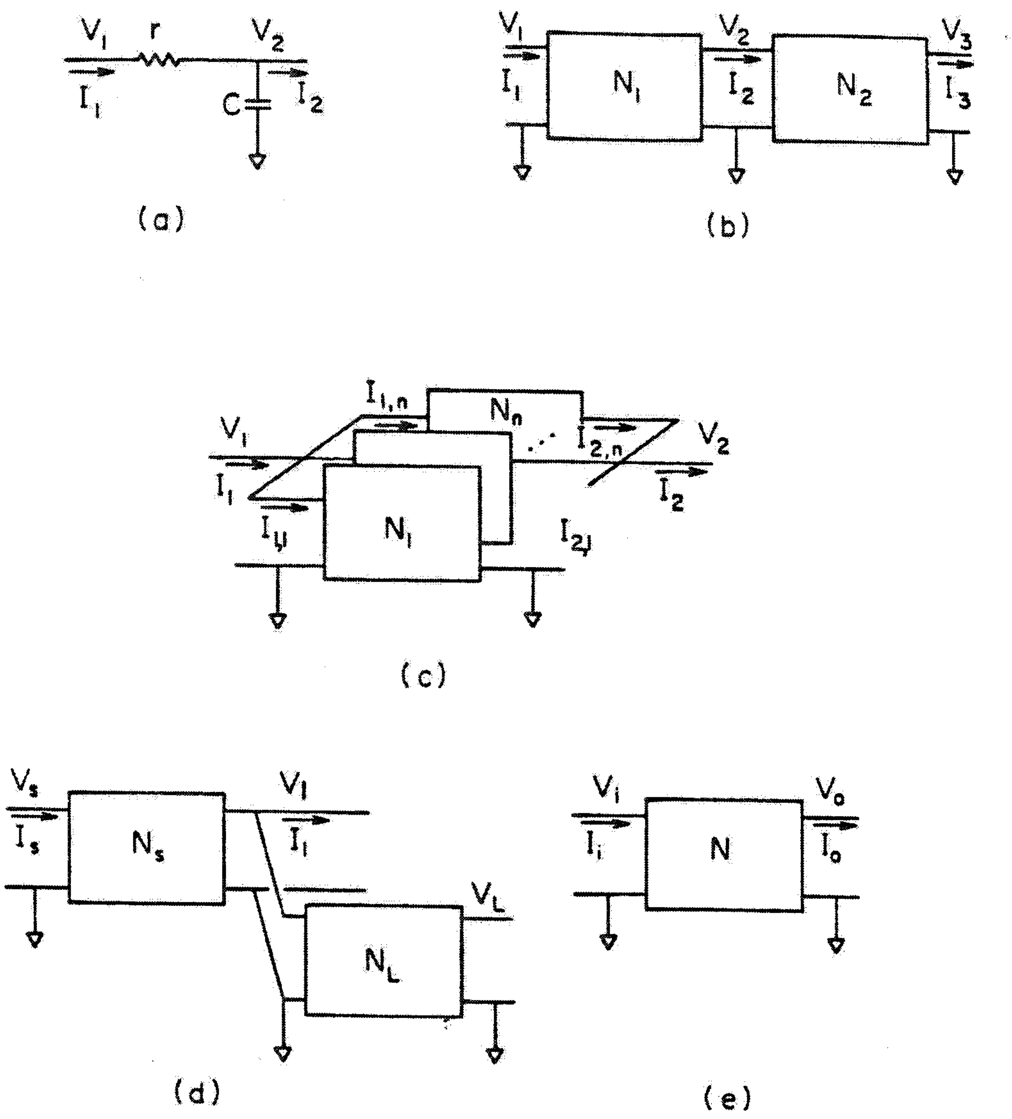


Figure 2.2. The five cases of a two-port RC network

the input is connected to the output of N_S (Figure 2.2.d).

5. a two-port RC network with input and output ports interchanged. Although this construction is not necessary in characterizing an RC network, it is very useful in practice for those networks where the directions of signal flows are dynamically changing (Figure 2.2.e). (2.5)

Finally, an "RC network" is defined as a two-port RC network with the output port open and input port connected to the source.

Remarks:

- In terms of two-port RC networks, the two cases not considered in our model (section 2.1) are described as follows:
  1. no driving source: a two-port RC network with output port open and input port connected to a capacitor.
  2. multiple sources: two two-port RC networks with their output ports connected together and input ports connected to VDD and GND, respectively.
- This definition of RC networks does not cover all possible network topologies because bridge connections may exist, which make the configuration neither series nor parallel. Simple bridge connections may be dealt with easily, and one such example will be given in section 3.4. As an extension of the $\Delta - Y$ transformation of resistor networks [14], we conjecture the existence of a transformation from a number of Y-connected networks to the same number of Δ-connected networks, and vice versa. This transformation is in terms of the (R, C, D, Q, D^*)-parameters of these networks such that, as far as delay is concerned, the two sets of networks are equivalent. If this conjecture is true, then these definitions do cover all possible network topologies. Moreover, a technique exists that is capable of dealing with general RC networks, and requires only the information of the first four cases of (2.5). This technique is described in the next chapter.

Consider an arbitrary node as the output. A two-port RC network between the source and the node can be constructed step by step using the process of (2.5). In Theorem 2.2, the relationship between the five parameters R, C, D, Q, and D^* and the transfer equation (2.4) of a two-port RC network is established for each of the five cases of (2.5). Then in Theorem 2.5, a formula for determining the delay of a node is derived from the two-port RC network between the source and the node. Although one such network is enough for this purpose, a more general result which also includes an explicit loading network is presented. This general result is very useful in the case of a tree network where the driving and loading networks are well-defined: they are the subtrees above and under the node, respectively. In such networks, the delay of every node can be obtained simultaneously and incrementally, as indicated in Theorem 2.9. Proofs of these three theorems are presented in section 2.6.

Theorem 2.2. Up to the first order, the transfer equation (2.4) of a two-port RC network is of the following form:

$$\begin{pmatrix} V_o(s) \\ I_o(s) \end{pmatrix} \approx \begin{pmatrix} 1 + s(RC - D) & -R + sb \\ -sC & 1 + sD \end{pmatrix} \begin{pmatrix} V_i(s) \\ I_i(s) \end{pmatrix} + \begin{pmatrix} -D^* + sh \\ Q + sf \end{pmatrix} \tag{2.6}$$

Symbol $\approx$ in (2.6) indicates that the two formulae are equal up to the s term. The parameters [2] R, C, D, Q, and D^* for each of the five cases of (2.5) are determined as follows:

[2] The parameters b, h, and f are of no concern in this context because they do not appear in the formula for T_D for either simple or composite networks.

1. primitive case:

$$\begin{aligned} R &= r \\ C &= c \\ D &= rc \\ Q &= sv_0 \\ D^* &= 0 \end{aligned} \tag{2.7}$$

where r, c and v_0 are the values of the resistance, the capacitance and the initial voltage of the capacitor, respectively.

In the following four cases, a subscript is associated with each parameter, indicating to which network this parameter belongs. In particular, subscript $_T$ indicates the resulting network of each composition (or operation).

2. series connection of N_1 and N_2:

$$\begin{aligned} R_T &= R_1 + R_2 \\ C_T &= C_1 + C_2 \\ D_T &= D_1 + D_2 + R_1C_2 \\ Q_T &= Q_1 + Q_2 \\ D_T^* &= D_1^* + D_2^* + R_2Q_1 \end{aligned} \tag{2.8}$$

3. parallel connection of $N_{1,\dots,n}$:

$$\begin{aligned}
R_T &= \frac{1}{\sum_1^n \frac{1}{R_i}} \\
C_T &= \sum_1^n C_i \\
D_T &= R_T\left(\sum_1^n \frac{D_i}{R_i}\right) \\
Q_T &= \sum_1^n Q_i \\
D_T^* &= R_T\left(\sum_1^n \frac{D_i^*}{R_i}\right)
\end{aligned} \tag{2.9}$$

4. N_S with side branch N_L:

$$\begin{aligned}
R_T &= R_S \\
C_T &= C_S + C_L \\
D_T &= D_S + R_S C_L \\
Q_T &= Q_S + Q_L \\
D_T^* &= D_S^*
\end{aligned} \tag{2.10}$$

5. input and output ports interchanged:

$$\begin{aligned}
R_T &= R \\
C_T &= C \\
D_T &= RC - D \\
Q_T &= Q \\
D_T^* &= RQ - D^*
\end{aligned} \tag{2.11}$$

▮

Corollary 2.3. If there is no initial charge in the two-port RC network, then the parameters Q and D^* are both 0. ∎

Corollary 2.4. If the nodes of the two-port RC network are all charged to 1 initially, then the parameters Q and D^* are equal to C and $RC - D$, respectively. ∎

Theorem 2.5. If a node connects to the source through a network N_S with parameters R_S, C_S, D_S, Q_S, and D_S^*, and is loaded by another network N_L with parameters R_L, C_L, D_L, Q_L, and D_L^*, then the delay T_D of this node is $(D_S + D_S^* - R_S Q_S) + R_S(C_L - Q_L)$. ∎

Corollary 2.6. If there is no initial charge in both N_S and N_L of Theorem 2.5, then $T_D = D_S + R_S C_L$. ∎

Among the five parameters of Theorem 2.2, D^* and Q can be expressed in terms of R, C and D for special initial charge distributions (Corollary 2.3 and Corollary 2.4). In fact, this reduction of parameters is possible in general, and only three parameters are necessary to represent any two-port RC network to calculate delays. Suggested by the result of Theorem 2.5, these three reduced parameters are

$$\begin{aligned} \overline{R} &= R \\ \overline{C} &= C - Q \\ \overline{D} &= D + D^* - RQ \end{aligned} \tag{2.12}$$

In case of zero initial charge, $\overline{R}$, $\overline{C}$, and $\overline{D}$ reduce to R, C, and D, respectively. In terms of these three reduced parameters, Theorem 2.2 and Theorem 2.5 are restated.

Theorem 2.2′. The three parameters $\overline{R}$, $\overline{C}$ and $\overline{D}$ of a two-port RC network can be determined as follows:

1. primitive case:

$$\overline{R} = r$$
$$\overline{C} = c(1 - v_0) \qquad (2.7')$$
$$\overline{D} = rc(1 - v_0)$$

The composition rules of these three parameters are the same as those of R, C and D in Theorem 2.2; i.e.,

2. series connection of N_1 and N_2:

$$\begin{aligned} \overline{R}_T &= \overline{R}_1 + \overline{R}_2 \\ \overline{C}_T &= \overline{C}_1 + \overline{C}_2 \\ \overline{D}_T &= \overline{D}_1 + \overline{D}_2 + \overline{R}_1\overline{C}_2 \end{aligned} \qquad (2.8')$$

3. parallel connection of $N_{1,\ldots,n}$:

$$\begin{aligned} \overline{R}_T &= \frac{1}{\sum_1^n \frac{1}{\overline{R}_i}} \\ \overline{C}_T &= \sum_1^n \overline{C}_i \\ \overline{D}_T &= \overline{R}_T \left(\sum_1^n \frac{\overline{D}_i}{\overline{R}_i} \right) \end{aligned} \qquad (2.9')$$

4. N_S with side branch N_L:

$$\begin{aligned} \overline{R}_T &= \overline{R}_S \\ \overline{C}_T &= \overline{C}_S + \overline{C}_L \\ \overline{D}_T &= \overline{D}_S + \overline{R}_S\overline{C}_L \end{aligned} \qquad (2.10')$$

5. input and output ports interchanged:

$$\begin{aligned} \overline{R}_T &= \overline{R} \\ \overline{C}_T &= \overline{C} \\ \overline{D}_T &= \overline{RC} - \overline{D} \end{aligned} \qquad (2.11')$$

∎

Theorem 2.5′. If a node connects to the source through a network N_S with parameters $\overline{R}_S$, $\overline{C}_S$, $\overline{D}_S$, and is loaded by another network N_L with parameters $\overline{R}_L$, $\overline{C}_L$, $\overline{D}_L$, then the delay T_D of this node is $\overline{D}_S + \overline{R}_S\overline{C}_L$. ∎

Corollary 2.7. If there is no explicit loading network N_L, then $T_D = \overline{D}_S$. ∎

Corollary 2.8. Consider a two-port RC network with series resistance R, total capacitance C, and total charge Q. Let T_1 denote the delay of the output port when the input port is driven and the output port is open, and T_2 that of the input port when the output port is driven and the input port is open. Then $T_1 + T_2 = \overline{R}\,\overline{C} = R(C - Q)$. ∎

Remarks:

- It is quite obvious that $\overline{R}$ and $\overline{C}$ alone are not enough to characterize the delay behavior of a two-port RC network: the capacitance in the network may be distributed differently, which results in different delays. The amazing thing is that, by adding only one more parameter $\overline{D}$, the delay of the network can be completely characterized, no matter how large the network is, or how the network is going to be composed.
- Many circuit analysis programs [37,46] use $R \cdot C$ for estimating delays. Corollary 2.8 indicates that this estimation is conservative, on the average, by a factor of two.
- The separation of driving network and loading network for a given node is by no means unique. For instance, N_L in Theorem 2.5′ can be considered as a side branch of N_S so that there is no explicit loading network at all. That the value of the delay is the same for both cases is shown as follows: Let subscript $_T$ denotes the network N_S with N_L merged inside. By case 4 of Theorem 2.2′, $\overline{D}_T = \overline{D}_S + \overline{R}_S\overline{C}_L$, and $\overline{R}_T = \overline{R}_S$. Then by Corollary 2.7, $T_D = \overline{D}_T = \overline{D}_S + \overline{R}_S\overline{C}_L$ which is the same as the result given in Theorem 2.5′.
- In most cases, there is more than one branch (transistor) incident on a node, and these branches belong to different two-port networks. The capacitance of the node can be arbitrarily distributed among these branches without affecting the result.

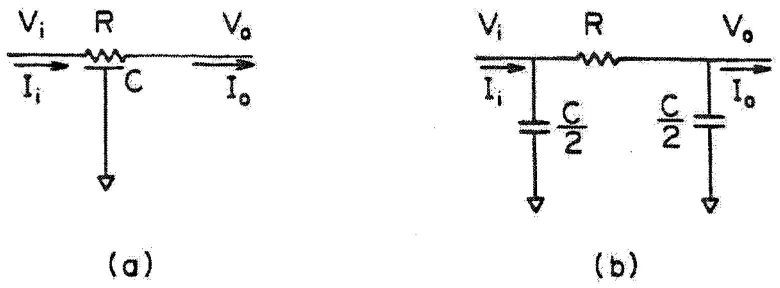


Figure 2.3. Approximation of a distributed RC line by lumped elements

- The T-matrix of a uniformly distributed RC line (Figure 2.3.a) is $\begin{pmatrix} \cosh\Gamma & -\frac{\sinh\Gamma}{Z} \\ -\sinh\Gamma & \cosh\Gamma \end{pmatrix}$, where $\Gamma = \sqrt{sRC}$ and $Z = \sqrt{\frac{R}{sC}}$ [16]. R and C are the total resistance and capacitance of the line, respectively. Up to the s term, the T-matrix is the same as $\begin{pmatrix} 1+\frac{sRC}{2} & -R-\frac{sR^2C}{6} \\ -sC & 1+\frac{sRC}{2} \end{pmatrix}$. That is to say, as far as delay is concerned, the RC line can be approximated by two capacitors and one resistor connected as shown in Figure 2.3.b [12].

2.4 The TREE Algorithm – Delay Calculation in RC Tree Networks

From Theorem 2.5′, the delay of a node depends on both the driving and loading networks of the node. Parallel (and bridge) connections couple all nodes together so that every node is driving and loading every other node at the same time. As a result, the calculation of delays in general needs to be carried out independently for each individual node. However, no node in a tree network both drives and loads another node, and the delays of all the nodes can be calculated simultaneously and incrementally.

Theorem 2.9. Suppose node N_1 and node N_2 are cascaded in an RC tree network. N_1 is nearer to the source and is connected to N_2 through a resistor of value r. The total capacitance and total charge of the loading network of N_2 are C_L and Q_L, respectively. If the delay of node N_1 is T_1, then the delay of node N_2 is $T_1 + r\overline{C}_L = T_1 + r(C_L - Q_L)$. ∎

Corollary 2.10. The delay of a node i in a tree network is $\sum_k R_{i,k}(C_k - Q_k)$, where $R_{i,k}$ is the mutual resistance between node i and k, i.e., the resistance of the (unique) path between the source and node i, that is in common with the (unique) path between the source and the node k. C_k and Q_k are the capacitance and initial charge of node k, respectively. The summation carries over all nodes k in the network [38]. ∎

The following algorithm (TREE) calculates the delays of all the nodes in a tree network.

1. The loading information is accumulated and propagated from the loading ends towards the driving end of the network. To be more precise, a value C_i^L is associated with each node i, and

$$C_i^L = \begin{cases} C_i - Q_i & \text{if node } i \text{ is a leaf;} \\ C_i - Q_i + \sum_j C_j^L & \text{otherwise,} \end{cases}$$

   where index j ranges over all the succeeding nodes of node i. C_i and Q_i are the node capacitance and stored charge of node i, respectively.

2. The delay of each node is calculated incrementally from the driving end towards the loading ends; i.e., $T_i = T_{p(i)} + r_i C_i^L$, where $p(i)$ is the parent node of node i, and r_i is the resistance between node i and node $p(i)$.

The correctness of this algorithm is guaranteed by Theorem 2.9. The time complexity is $O(n)$, where n is the number of nodes in the tree network.

2.5 Summary

Digital MOS circuits are approximated by RC networks for estimating delays. Every transistor group corresponds to one RC network, which is assumed to be driven by one and only one source (VDD or GND). The area criterion of (2.1′) is used as the definition of delay. This definition is consistent with and more general than the Elmore's delay because it also handles nonmonotonic responses. To justify the usage of this definition, we proved that, given any threshold voltage, the delay always falls within the Penfield-Rubinstein bounds for any RC tree networks.

The Penfield-Rubinstein RC tree model is successfully extended to include the effects of parallel paths and initial charge distributions. The formulation of a transfer equation leads naturally to the composition rules of the R, C and D parameters of two-port RC networks. These composition rules extend the classical circuit theory for composing resistance of an arbitrary resistive network between two ports. Theorem 2.2′ represents the first success in characterizing the transient behavior of any reasonable class of circuits in a general and rigorous way.

With Theorem 2.2′ and Theorem 2.5′, the two-port RC network between the source and a node can be constructed, and the delay of the node can be calculated. This process is direct, constructive, and requires information regarding the global topology of the network. Presented in Chapter 3 is another approach to delay calculation that is iterative and distributive in nature. Each node or transistor is itself a process, which communicates only with its neighboring nodes and transistors. The delays of all the nodes are determined in a collective manner. This approach is capable of dealing with general RC networks without sacrificing the desirable property of tree networks.

2.6 Appendix: Proofs of Theorems

Theorem 2.1. Consider a node in an RC tree network with no initial charge. Let

$$
\begin{aligned}
t_1(v) &= T_D - T_P(1-v) \\
t_2(v) &= T_R \ln \frac{T_D}{T_P(1-v)} \\
t_3(v) &= \frac{T_D}{1-v} - T_R \\
t_4(v) &= T_P - T_R + T_P \ln \frac{T_D}{T_P(1-v)}
\end{aligned}
$$

where $T_D = T_D^0$ is the Elmore's delay. T_R, and T_P are defined in [38], which satisfy $T_R \leq T_D \leq T_P$ for any node in an RC tree network. Let $t_d(v) = T_D \ln(\frac{1}{1-v})$. Then $t_d \geq t_1, t_d \geq t_2, t_d \leq t_3$ for all values of v, and $t_d \leq t_4$ for $v \geq 1 - \frac{T_D}{T_P}$.

Proof:

The following three inequalities regarding the natural log function are noted first:

1. $\ln \frac{1}{1-x} \geq x$, for $0 \leq x \leq 1$
2. $x - \ln x \geq 1$, for $x \geq 1$
3. $1 \geq x(1 - \ln x)$, for $0 \leq x \leq 1$

The proof of the theorem itself goes as follows:

$$
\begin{aligned}
t_d - t_1 &= T_D \ln\left(\frac{1}{1-v}\right) - T_D + T_P(1-v) \\
&\geq T_D \left(\ln \frac{1}{1-v} - v\right) \\
&\geq 0
\end{aligned}
$$

$$
\begin{aligned}
t_d - t_2 &= (T_D - T_R) \ln \frac{1}{1-v} + T_R \ln \frac{T_P}{T_D} \\
&\geq 0
\end{aligned}
$$

$$\begin{aligned} t_3 - t_d &= \frac{T_D}{1-v} - T_R - T_D \ln \frac{1}{1-v} \\ &= T_D\left(\frac{1}{1-v} - \ln\frac{1}{1-v}\right) - T_R \\ &\geq T_D - T_R \\ &\geq 0 \end{aligned}$$

$$\begin{aligned} t_4 - t_d &= T_P - T_R + T_P \ln \frac{T_D}{T_P(1-v)} - T_D \ln \frac{1}{1-v} \\ &\geq T_P - T_R + T_D \ln \frac{T_D}{T_P} \qquad \cdots\cdots \qquad \left(\frac{T_D}{T_P(1-v)} \geq 1\right) \\ &\geq T_P\left(1 - \frac{T_D}{T_P}\left(1 - \ln\frac{T_D}{T_P}\right)\right) \\ &\geq 0 \end{aligned}$$

∎

Theorem 2.2. Up to the first order, the transfer equation (2.4) of a two-port RC network is of the following form:

$$\begin{pmatrix} V_o(s) \\ I_o(s) \end{pmatrix} \approx \begin{pmatrix} 1 + s(RC - D) & -R + sb \\ sC & 1 + sD \end{pmatrix} \begin{pmatrix} V_i(s) \\ I_i(s) \end{pmatrix} + \begin{pmatrix} -D^* + sh \\ Q + sf \end{pmatrix} \tag{2.6}$$

The parameters R, C, D, Q, and D^* for each of the five cases in (2.5) are determined as follows:

1. primitive case: $R = r$, $C = c$, $D = rc$, $Q = cv_0$, $D^* = 0$. (2.7)
2. series connection of N_1 and N_2: $R_T = R_1 + R_2$, $C_T = C_1 + C_2$, $D_T = D_1 + D_2 + R_1C_2$, $Q_T = Q_1 + Q_2$, $D_T^* = D_1^* + D_2^* + R_2Q_1$. (2.8)
3. parallel connection of $N_{1,\ldots,n}$: $R_T = \frac{1}{\sum_1^n \frac{1}{R_i}}$, $C_T = \sum_1^n C_i$, $D_T = R_T\left(\sum_1^n \frac{D_i}{R_i}\right)$, $Q_T = \sum_1^n Q_i$, $D_T^* = R_T\left(\sum_1^n \frac{D_i^*}{R_i}\right)$. (2.9)
4. N_S with side branch N_L: $R_T = R_S$, $C_T = C_S + C_L$, $D_T = D_S + R_SC_L$, $Q_T = Q_S + Q_L$, $D_T^* = D_S^*$. (2.10)

5. input and output ports interchanged: $R_T = R$, $C_T = C$, $D_T = RC - D$, $Q_T = Q$, $D_T^* = RQ - D^*$. (2.11)

Proof:

1. primitive case: From Fig.2.2.a, v_1, v_2, i_1, and i_2 are related by the following equations:

$$\begin{aligned} v_1(t) - v_2(t) &= r i_1(t) \\ i_1(t) - i_2(t) &= c\frac{d}{dt}v_2(t). \end{aligned} \tag{2.13}$$

Expressed in the matrix form in the Laplace domain, (2.13) becomes

$$\begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} = \begin{pmatrix} 1 & -r \\ -sc & 1+src \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} + \begin{pmatrix} 0 \\ cv_0 \end{pmatrix}.$$

This is of the form (2.6) with parameters given in (2.7).

2. series connection: Assume that

$$\begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} \approx \begin{pmatrix} 1+s(R_1C_1 - D_1) & -R_1 + sb_1 \\ -sC_1 & 1+sD_1 \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} + \begin{pmatrix} -D_1^* + sh_1 \\ Q_1 + sf_1 \end{pmatrix}$$

$$\begin{pmatrix} V_3(s) \\ I_3(s) \end{pmatrix} \approx \begin{pmatrix} 1+s(R_2C_2 - D_2) & -R_2 + sb_2 \\ -sC_2 & 1+sD_2 \end{pmatrix} \begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} + \begin{pmatrix} -D_2^* + sh_2 \\ Q_2 + sf_2 \end{pmatrix}$$

then

$$\begin{aligned} \begin{pmatrix} V_3(s) \\ I_3(s) \end{pmatrix} &\approx \begin{pmatrix} 1+s(R_2C_2 - D_2) & -R_2 + sb_2 \\ -sC_2 & 1+sD_2 \end{pmatrix} \begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} + \begin{pmatrix} -D_2^* + sh_2 \\ Q_2 + sf_2 \end{pmatrix} \\ &\approx \begin{pmatrix} 1+s(R_2C_2 - D_2) & -R_2 + sb_2 \\ -sC_2 & 1+sD_2 \end{pmatrix} \begin{pmatrix} 1+s(R_1C_1 - D_1) & -R_1 + sb_1 \\ -sC_1 & 1+sD_1 \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} \\ &\quad + \begin{pmatrix} 1+s(R_2C_2 - D_2) & -R_2 + sb_2 \\ -sC_2 & 1+sD_2 \end{pmatrix} \begin{pmatrix} -D_1^* + sh_1 \\ Q_1 + sf_1 \end{pmatrix} + \begin{pmatrix} -D_2^* + sh_2 \\ Q_2 + sf_2 \end{pmatrix} \\ &\approx \begin{pmatrix} 1+s(R_TC_T - D_T) & -R_T + sb_T \\ -sC_T & 1+sD_T \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} + \begin{pmatrix} -D_T^* + sh_T \\ Q_T + sf_T \end{pmatrix}. \end{aligned}$$

This is of the form (2.6). The corresponding parameters may be easily checked against those in (2.8).

3. parallel connection: Assume that

$$\begin{pmatrix} V_2(s) \\ I_{2,i}(s) \end{pmatrix} \approx \begin{pmatrix} 1 + s(R_iC_i - D_i) & -R_i + sb_i \\ -sC_i & 1 + sD_i \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_{1,i}(s) \end{pmatrix} + \begin{pmatrix} -D_i^* + sh_i \\ Q_i + sf_i \end{pmatrix}.$$

This equation may be transformed into the following G-matrix form [14]:

$$\begin{aligned} \begin{pmatrix} I_{1,i}(s) \\ I_{2,i}(s) \end{pmatrix} &\approx \begin{pmatrix} -\frac{1+s(R_iC_i-D_i)}{-R_i+sb_i} & \frac{1}{-R_i+sb_i} \\ -sC_i - \frac{(1+sD_i)(1+s(R_iC_i-D_i))}{-R_i+sb_i} & \frac{1+sD_i}{-R_i+sb_i} \end{pmatrix} \begin{pmatrix} V_1(s) \\ V_2(s) \end{pmatrix} + \\ &\quad \begin{pmatrix} -\frac{-D_i^*+sh_i}{-R_i+sb_i} \\ -\frac{(1+sD_i)(-D_i^*+sh_i)}{-R_i+sb_i} + Q_i + sf_i \end{pmatrix} \\ &\approx \begin{pmatrix} \frac{1}{R_i} + s\left(C_i - \frac{D_i}{R_i} + \frac{b_i}{R_i^2}\right) & -\left(\frac{1}{R_i} + s\frac{b_i}{R_i^2}\right) \\ \frac{1}{R_i} + s\frac{b_i}{R_i^2} & -\left(\frac{1}{R_i} + s\left(\frac{D_i}{R_i} + \frac{b_i}{R_i^2}\right)\right) \end{pmatrix} \begin{pmatrix} V_1(s) \\ V_2(s) \end{pmatrix} + \\ &\quad \begin{pmatrix} -\frac{D_i^*}{R_i} + sm_i \\ -\frac{D_i^*}{R_i} + Q_i + sl_i \end{pmatrix} \end{aligned}$$

where $m_i = \left(\frac{h_i}{R_i} - \frac{b_iD_i^*}{R_i^2}\right)$, and $l_i = \frac{-D_iD_i^*}{R_i} + \frac{b_iD_i^*}{R_i^2} + \frac{h_i}{R_i} + f_i$. Since $I_1(s) = \sum_1^n I_{1,i}(s)$ and $I_2(s) = \sum_1^n I_{2,i}(s)$,

$$\begin{aligned} \begin{pmatrix} I_1(s) \\ I_2(s) \end{pmatrix} &\approx \begin{pmatrix} \sum\left(\frac{1}{R_i} + s\left(C_i - \frac{D_i}{R_i} + \frac{b_i}{R_i^2}\right)\right) & -\sum\left(\frac{1}{R_i} + s\frac{b_i}{R_i^2}\right) \\ \sum\left(\frac{1}{R_i} + s\frac{b_i}{R_i^2}\right) & -\sum\left(\frac{1}{R_i} + s\left(\frac{D_i}{R_i} + \frac{b_i}{R_i^2}\right)\right) \end{pmatrix} \begin{pmatrix} V_1(s) \\ V_2(s) \end{pmatrix} \\ &\quad + \begin{pmatrix} \sum\left(-\frac{D_i^*}{R_i} + sm_i\right) \\ \sum\left(-\frac{D_i^*}{R_i} + Q_i + sl_i\right) \end{pmatrix}. \end{aligned}$$

Transforming from the G-matrix formulation back to the T-matrix formulation,

$$\begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} \approx \begin{pmatrix} \dfrac{\sum\left(\frac{1}{R_i}+s\left(C_i-\frac{D_i}{R_i}+\frac{b_i}{R_i^2}\right)\right)}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)} & -\dfrac{1}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)} \\ \sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)-\dfrac{\sum\left(\frac{1}{R_i}+s\left(\frac{D_i}{R_i}+\frac{b_i}{R_i^2}\right)\right)\sum\left(\frac{1}{R_i}+s\left(C_i-\frac{D_i}{R_i}+\frac{b_i}{R_i^2}\right)\right)}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)} & \dfrac{\sum\left(\frac{1}{R_i}+s\left(\frac{D_i}{R_i}+\frac{b_i}{R_i^2}\right.\right.}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)} \end{pmatrix}$$

$$\begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} + \begin{pmatrix} \dfrac{\sum\left(-\frac{D_i^*}{R_i}+sm_i\right)}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)} \\ -\dfrac{\sum\left(\frac{1}{R_i}+s\left(\frac{D_i}{R_i}+\frac{b_i}{R_i^2}\right)\right)\sum\left(-\frac{D_i^*}{R_i}+sm_i\right)}{\sum\left(\frac{1}{R_i}+s\frac{b_i}{R_i^2}\right)}+\sum\left(-\frac{D_i^*}{R_i}+Q_i+sl_i\right) \end{pmatrix}$$

$$\approx \begin{pmatrix} 1+s\left(\frac{\sum C_i}{\sum\frac{1}{R_i}}-\frac{\sum\frac{D_i}{R_i}}{\sum\frac{1}{R_i}}\right) & -\frac{1}{\sum\frac{1}{R_i}}+s\frac{\sum\frac{b_i}{R_i^2}}{\left(\sum\frac{1}{R_i}\right)^2} \\ -s\sum C_i & 1+s\frac{\sum\frac{D_i}{R_i}}{\sum\frac{1}{R_i}} \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} + \begin{pmatrix} -\frac{\sum\frac{D_i^*}{R_i}}{\sum\frac{1}{R_i}}+sh_T \\ \sum Q_i+sf_T \end{pmatrix}.$$

This is of the form (2.6) with parameters given in (2.9).

4. with open side branch N_L: Assume that

$$\begin{pmatrix} V_1(s) \\ I_1(s)-I_L(s) \end{pmatrix} \approx \begin{pmatrix} 1+s(R_SC_S-D_S) & -R_S+sb_S \\ -sC_S & 1+sD_S \end{pmatrix} \begin{pmatrix} V_S(s) \\ I_S(s) \end{pmatrix} + \begin{pmatrix} -D_S^*+sh_S \\ Q_S+sf_S \end{pmatrix}$$

$$\begin{pmatrix} V_L(s) \\ 0 \end{pmatrix} \approx \begin{pmatrix} 1+s(R_LC_L-D_L) & -R_L+sb_L \\ -sC_L & 1+sD_L \end{pmatrix} \begin{pmatrix} V_1(s) \\ I_L(s) \end{pmatrix} + \begin{pmatrix} -D_L^*+sh_L \\ Q_L+sf_L \end{pmatrix}.$$

After a few steps of simplification,

$$\begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} \approx \begin{pmatrix} 1+s(R_TC_T-D_T) & -R_T+sb_T \\ -sC_T & 1+sD_T \end{pmatrix} \begin{pmatrix} V_S(s) \\ I_S(s) \end{pmatrix} + \begin{pmatrix} -D_T^*+sh_T \\ Q_T+sf_T \end{pmatrix}$$

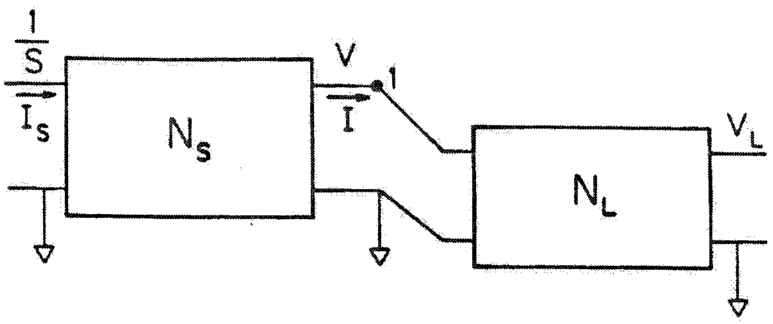


Figure 2.4. Delay calculation of a node in an RC network

with the set of parameters shown in (2.10).

5. input and output ports interchanged:

$$\begin{pmatrix} V_o(s) \\ I_o(s) \end{pmatrix} \approx \begin{pmatrix} 1+s(RC-D) & -R+sb \\ -sC & 1+sD \end{pmatrix} \begin{pmatrix} V_i(s) \\ I_i(s) \end{pmatrix} + \begin{pmatrix} -D^* + sh \\ Q+sf \end{pmatrix}$$

$$\begin{aligned} \begin{pmatrix} V_i(s) \\ I_i(s) \end{pmatrix} &\approx \begin{pmatrix} 1+s(RC-D) & -R+sb \\ -sC & 1+sD \end{pmatrix}^{-1} \begin{pmatrix} V_o(s)+D^* - se \\ I_o(s) - Q - sf \end{pmatrix} \\ &\approx \begin{pmatrix} 1+sD & R-sb \\ sC & 1+s(RC-D) \end{pmatrix} \begin{pmatrix} V_o(s) \\ I_o(s) \end{pmatrix} + \begin{pmatrix} D^* - RQ + sh_r \\ -Q - sf_r \end{pmatrix}. \end{aligned}$$

Finally,

$$\begin{pmatrix} V_i(s) \\ -I_i(s) \end{pmatrix} \approx \begin{pmatrix} 1+sD & -R+sb \\ -sC & 1+s(RC-D) \end{pmatrix} \begin{pmatrix} V_o(s) \\ -I_o(s) \end{pmatrix} + \begin{pmatrix} -(RQ-D^*)+sh_r \\ Q+sf_r \end{pmatrix}.$$

∎

Theorem 2.5. If a node connects to the source through a network N_S with parameters R_S, C_S, D_S, Q_S, and D_S^*, and is loaded by another network N_L with parameters R_L, C_L, D_L, Q_L, and D_L^*, then the delay T_D of this node is $(D_S + D_S^* - R_SQ_S) + R_S(C_L - Q_L)$ (Figure 2.4).

Proof:

From the hypothesis and from Theorem 2.2,

$$\begin{pmatrix} V_L(s) \\ 0 \end{pmatrix} \approx \begin{pmatrix} 1 + s(R_LC_L - D_L) & -R_L + sb_{_L} \\ -sC_L & 1 + sD_L \end{pmatrix} \begin{pmatrix} V(s) \\ I(s) \end{pmatrix} + \begin{pmatrix} -D_L^* + sh_{_L} \\ Q_L + sf_{_L} \end{pmatrix}$$

$$\begin{pmatrix} V(s) \\ I(s) \end{pmatrix} \approx \begin{pmatrix} 1 + s(R_SC_S - D_S) & -R_S + sb_S \\ -sC_S & 1 + sD_S \end{pmatrix} \begin{pmatrix} \frac{1}{s} \\ I_S \end{pmatrix} + \begin{pmatrix} -D_S^* + sh_s \\ Q_S + sf_s \end{pmatrix}.$$

After a few steps of simplification,

$$V_L(s) \approx \frac{1 + s(R_TC_T - R_S(C_T - Q_S - Q_L) - (D_S^* + R_LQ_S + D_L^*))}{s(1 + sD_T)}$$

where $R_T = R_S + R_L$, $C_T = C_S + C_L$, and $D_T = D_S + D_L + R_SC_L$. Reflecting back from node L to node 1,

$$\begin{aligned} \begin{pmatrix} V(s) \\ I(s) \end{pmatrix} &\approx \begin{pmatrix} 1 + s(R_LC_L - D_L) & -R_L + sb_L \\ -sC_L & 1 + sD_L \end{pmatrix}^{-1} \begin{pmatrix} V_L(s) + D_L^* - sh_{_L} \\ -Q_L - sf_{_L} \end{pmatrix} \\ &\approx \begin{pmatrix} 1 + sD_L & R_L - sb_{_L} \\ sC_L & 1 + s(R_LC_L - D_L) \end{pmatrix} \begin{pmatrix} V_L(s) + D_L^* - sh_{_L} \\ -Q_L - sf_{_L} \end{pmatrix}. \end{aligned}$$

Finally,

$$\begin{aligned} V(s) &\approx (1 + sD_L)V_L(s) + (1 + sD_L)(D_L^* - sh_{_L}) - (R_L - sb_{_L})(Q_L + sf_{_L}) \\ &\approx \frac{(1 + sD_L)(1 + s\Delta) + s(1 + sD_T)(D_L^* - R_LQ_L)}{s(1 + sD_T)} \\ &\approx \frac{1 + s(D_L + \Delta + D_L^* - R_LQ_L)}{s(1 + sD_T)} \end{aligned}$$

where $\Delta = R_TC_T - R_S(C_T - Q_S - Q_L) - (D_S^* + R_LQ_S + D_L^*)$. As a result, $T_D = D_T - (D_L + \Delta + D_L^* - R_LQ_L) = D_S + R_S(C_L - Q_L) + (D_S^* - R_SQ_S)$. ∎

Theorem 2.9. Suppose node N_1 and node N_2 are cascaded in an RC tree network. N_1 is nearer to the source and is connected to N_2 through a resistor of value r. The total capacitance and total charge of the loading network of N_2 are C_L and Q_L, respectively. If the delay of node N_1 is T_1, then the delay of node N_2 is $T_1 + r\overline{C}_L = T_1 + r(C_L - Q_L)$.

Proof:

Let subscripts $_{Li}$ and $_{Si}$ denote the loading and driving networks of node $N_i, i = 1, 2$, respectively.

From Theorem 2.2′,

$$\overline{D}_{S2} = \overline{D}_{S1} + \overline{R}_{S2}\overline{C}_2$$

$$\overline{R}_{S2} = \overline{R}_{S1} + r$$

$$\overline{C}_{L2} = \overline{C}_{L1} - \overline{C}_2.$$

From Theorem 2.5′,

$$T_1 = \overline{D}_{S1} + \overline{R}_{S1}\overline{C}_{L1}$$

$$T_2 = \overline{D}_{S2} + \overline{R}_{S2}\overline{C}_{L2}.$$

Thus, $T_2 - T_1 = (\overline{D}_{S2} - \overline{D}_{S1}) + (\overline{R}_{S2}\overline{C}_{L2} - \overline{R}_{S1}\overline{C}_{L1}) = r\overline{C}_{L1} = r(C_{L1} - Q_{L1})$. ∎

Chapter 3

Delay Calculation in General RC Networks

In section 2.4, a linear algorithm (TREE) was presented to calculate the delays of all nodes in an RC tree network. This algorithm cannot be applied to a non-tree network because the driving and loading networks of the nodes in such a network are not explicit. Parallel (and bridge) connections couple all nodes together, so that every node is driving and loading other nodes at the same time. As a result, the delay values must be determined collectively through some relaxation process.

In section 3.1, the problem of delay calculation is reformulated as a set of relations among neighboring nodes and branches. By taking advantage of the effectiveness of delay calculation in RC tree networks, this formulation leads to the concept of tree decomposition and load redistribution, which is the main subject of section 3.2. A relaxation algorithm for calculating delays in general RC networks is presented in section 3.3, and analyzed in section 3.6. This algorithm requires information from neighboring nodes and branches only during the relaxation process. Under certain conditions, this algorithm is equivalent to the block Gauss-Seidel method for solving a system of linear equations. The matrix associated with this system of linear equations is symmetric and positive-definite, which guarantees the convergence of the G-S method. Examples of delay calculations are given in section 3.4.

3.1 Distributive View of Delay Calculation

The problem of evaluating delays in an RC network can be reformulated as a set of relations among neighboring nodes and branches. Associate a global index with each node. Suppose there are a_i branches incident on a node N_i. Let $r_{(i,j)}$ denote the resistance of the jth branch, and $f(i,j)$ denote the global index of the neighboring node of N_i through this branch. The idea here is to partition $\overline{C}_i = C_i - Q_i$ into these a_i branches, each of value $\overline{C}_{(i,j)}$, such that $\sum_{j=1,\ldots,a_i} \overline{C}_{(i,j)} = \overline{C}_i$, and the delays evaluated from different branches are the same. $\overline{C}_{(i,j)}$ is the equivalent load on node N_i from the jth branch. By Theorem 2.9, $T_i = T_{f(i,j)} + r_{(i,j)}\overline{C}_{(i,j)}$. To summarize, we have the following set of relations

$$\begin{cases} T_i = T_{f(i,j)} + r_{(i,j)}\overline{C}_{(i,j)} & j = 1,\ldots,a_i,\ i = 1,\ldots,N \\ \sum_{j=1}^{a_i} \overline{C}_{(i,j)} = \overline{C}_i & i = 1,\ldots,N \end{cases} \tag{3.1}$$

where N is the number of nodes in the network. The formal derivation of (3.1) is as follows. By the definition of delay (2.1′), $T_i = \int_0^\infty (1 - v_i)dt$, and $T_{f(i,j)} = \int_0^\infty (1 - v_{f(i,j)})dt$.

$$\overline{C}_{(i,j)} \stackrel{\text{def}}{=} \frac{T_i - T_{f(i,j)}}{r_{(i,j)}} = \int_0^\infty \frac{v_{f(i,j)} - v_i}{r_{(i,j)}} dt \tag{3.2}$$

The first set of equations of (3.1) are established by this definition. The second set of equations are established as follows: By Ohm's law and Kirchoff's current law,

$$C_i \frac{dv_i}{dt} = \sum_j \frac{v_{f(i,j)} - v_i}{r_{(i,j)}}. \tag{3.3}$$

Combining (3.2) and (3.3),

$$\begin{aligned} \sum_j \overline{C}_{(i,j)} &= \sum_j \int_0^\infty \frac{v_{f(i,j)} - v_i}{r_{(i,j)}} dt \\ &= \int_0^\infty C_i \frac{dv_i}{dt} dt \\ &= C_i - Q_i \\ &= \overline{C}_i \end{aligned}$$

∎

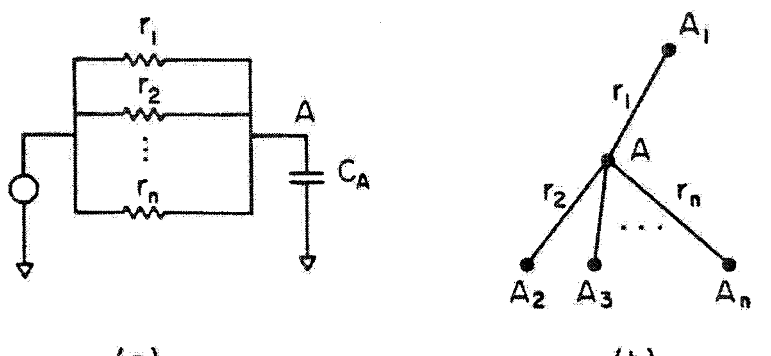


Figure 3.1. Two examples illustrating T_i's and $C_{(i,j)}$'s

Formula (3.1) represents a system of linear equations: $r_{(i,j)}$'s and $\overline{C}_i$'s are known, and T_i's and $\overline{C}_{(i,j)}$'s are to be determined. This system of equations is general enough to deal with all RC networks, including those with bridge connections. To simplify the discussion, zero initial charge is assumed throughout this chapter. The results obtained are directly applicable to general cases by replacing C's with $(C - Q)$'s.

Example 3.1. Consider the two simple circuits in Figure 3.1. Circuit 3.1.a consists of n transistors connecting the source to a node A. All transistors are ON, and are with resistances $r_{1,\ldots,n}$, respectively. From Theorem 2.2 and Corollary 2.7, $T_A = R_T C_A$, where $R_T = \frac{1}{\sum_{j=1}^{n} \frac{1}{r_j}}$, and C_A is the capacitance of node A. Another way to calculate the delay is that, instead of combining resistances, capacitance C_A is distributed into the n incident branches: $C_{(A,j)} = \frac{R_T}{r_j} C_A$, for $j = 1, \ldots, n$. This combination of $C_{(A,j)}$'s is the only possible partition of C_A such that $\sum_{j=1}^{n} C_{(A,j)} = C_A$, and the delays evaluated from all n branches are equal. This common value of delay equals $R_T C_A$. Both methods give the same result.

Consider an arbitrary node A inside a tree network such as circuit 3.1.b. Suppose there are n branches incident on node A. As the network is a tree, there are also n neighboring nodes of A. Among these n neighboring nodes, one node is nearer to the source than node A (call this node A_1), and all other nodes are farther away from the source (call these nodes $A_{2,\ldots,n}$, respectively). By Theorem 2.9, $T_{A_j} = T_A + r_j C_j^L$, for $j = 2,\ldots,n$, where r_j is the resistance between nodes A and A_j, and C_j^L is the total load capacitance of node A_j. From (3.2), $C_{(A,j)} = \frac{T_A - T_{(A,j)}}{r_j} = -C_j^L$, for $j = 2,\ldots,n$. Again by Theorem 2.9, $T_A = T_{A_1} + r_1 C_A^L$, and thus $C_{A,1} = C_A^L$. It is easy to check that $\sum_{j=1}^{n} C_{(A,j)} = C_A^L - \sum_{j=2}^{n} C_j^L = C_A$. Note that $C_{(A,j)}$ is negative for $j = 2,\ldots,n$, indicating that node A is driving, not loading node A_j. ∎

3.2 Tree Decomposition and Load Distribution

We do not intend to solve equation (3.1) directly because of the enormous number of variables involved: $\sum_{i=1}^{N}(a_i + 1)$. Note that the a_i branches incident on node N_i need not be decoupled completely as we did in the formulation of (3.1). These branches can be divided into any number $(b_i, 1 \le b_i \le a_i)$ of groups. Rather than fully decouple the network into nodes and transistors, it is decomposed into a smaller number of subnetworks. Delays are calculated directly and independently inside each subnetwork using the techniques discussed in Chapter 2. The consistency of the delay of a common node shared by more than one subnetwork is checked and corrected by a procedure similar to the formulation of (3.1). As delays can be calculated very efficiently for a tree network, we require that all decomposed subnetworks be trees. The root of every tree must be the source of the network. For convenience, the following terminology is introduced. As node capacitance C_i is partitioned into b_i parts, each of these partitioned capacitances is considered as separated nodes. Such nodes are referred to as "secondary nodes," while the original nodes of the network are referred to as "primary nodes." If there is no ambiguity in the context, the term "node" refers to either a primary node or a secondary node. Those primary nodes with $b_i > 1$ are also called "split primary nodes." Suppose that there are P split primary nodes $(N_{1,\ldots,P})$ and $N - P$ nonsplit ones $(N_{P+1,\ldots,N})$ in the network. With every secondary node is associated an index pair (i, j), indicating the jth secondary node generated from the ith primary node of the network. The term "equivalent secondary nodes" refers to the set of secondary nodes that correspond to the same primary node. By considering equivalent secondary nodes as disjoint, the decomposition of a network is achieved. The original network is also called the "primary network," and the decomposed network is called the "secondary network." The transformation from a primary network to a secondary network is a two-step process. The first step is purely topological, while the second step concerns the distribution of node capacitances, as well. The first step is referred to as "topological decomposition," and the second step is referred to as "load distribution." For a given RC network, a topological decomposition can always be found that separates the network into a collection of tree subnetworks. This collection of trees can be considered either as disjoint trees or as branches of one single tree that is rooted at the source of the network. Based upon the concept of dominant path [4], one such decomposition scheme is presented in section 4.1. The discussion in this chapter applies to any tree decomposition of RC networks.

As the secondary network is a collection of independent tree subnetworks, the delay of each secondary node can be calculated directly. The question arises as to how the delays of these secondary nodes are related to those of the primary nodes. It is quite possible that equivalent secondary nodes have different delay values. Note that the delays of secondary nodes depend on the values of $C_{(i,j)}$'s. If the capacitances C_i's are distributed incorrectly among these secondary nodes, the delay values will be different. However, if the C_i's are somehow distributed so that equivalent secondary nodes give the same delay, then it makes no difference whether these nodes are connected or not. If connected together, the secondary network reduces to the primary network, and the delays of the primary nodes are equal to the common delays of the corresponding set of equivalent secondary nodes. In what follows, we show that for any given tree (topological) decomposition of an RC network, such a load distribution always exists and is unique. Via this distribution, we also present an algorithm to find the delays of all nodes of an arbitrary network.

From Corollary 2.10, the delay $T_{(i,j)}$ of node $N_{(i,j)}$ is equal to $\sum_{u=1}^{N}\sum_{v=1}^{b_u} R_{(i,j)}^{(u,v)} C_{(u,v)}$, where $R_{(i,j)}^{(u,v)}$ is the resistance of the (unique) path between the source and node $N_{(i,j)}$, that is in common with the (unique) path between the source and node $N_{(u,v)}$. If node $N_{(u,v)}$ and node $N_{(i,j)}$ are not in the same tree subnetwork, then $R_{(i,j)}^{(u,v)} = 0$. Equating $T_{(i,j)}$'s for equivalent secondary nodes, $\sum_{u=1}^{N}\sum_{v=1}^{b_u} R_{(i,1)}^{(u,v)} C_{(u,v)} = \sum_{u=1}^{N}\sum_{v=1}^{b_u} R_{(i,2)}^{(u,v)} C_{(u,v)} = \cdots = \sum_{u=1}^{N}\sum_{v=1}^{b_u} R_{(i,b_i)}^{(u,v)} C_{(u,v)}$, or $\sum_{u=1}^{N}\sum_{v=1}^{b_u}(R_{(i,j)}^{(u,v)} - R_{(i,b_i)}^{(u,v)}) C_{(u,v)} = 0$, for $j = 1, \ldots, b_i - 1$, and $i = 1, \ldots, P$. Since $\sum_{v=1}^{b_u} C_{(u,v)} = C_u$, or $C_{(u,b_u)} = C_u - \sum_{v=1}^{b_u-1} C_{(u,v)}$, the above set of equations can be reduced to

$$\sum_{u=1}^{P}\sum_{v=1}^{b_u-1}\left(R_{(i,j)}^{(u,v)} - R_{(i,b_i)}^{(u,v)} - R_{(i,j)}^{(u,b_u)} + R_{(i,b_i)}^{(u,b_u)}\right) C_{(u,v)} = \sum_{u=1}^{N}\left(R_{(i,b_i)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)}\right) C_u. \tag{3.4}$$

Formula (3.4) represents a system of linear equations with $\sum_{u=1}^{P}(b_i - 1)$ variables: $C_{(1,1)}, \cdot, C_{(1,b_1-1)}, C_{(2,1)}, \cdot, C_{(2,b_2-1)}, \ldots, C_{(p,1)}, \cdot, C_{(p,b_p-1)}$. Equation (3.4) can also be written in a matrix form $Ax = b$, where A is a $\sum_{i=1}^{P}(b_i - 1) \times \sum_{u=i}^{P}(b_i - 1)$ matrix with element $a_{(i,j),(u,v)} = R_{(i,j)}^{(u,v)} - R_{(i,b_i)}^{(u,v)} - R_{(i,j)}^{(u,b_u)} + R_{(i,b_i)}^{(u,b_u)}$. Both b and x are $\sum_{i=1}^{P}(b_i - 1)$-vectors with element $b_{(i,j)} = \sum_{u=1}^{N}(R_{(i,b_i)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)})C_u$, and $x_{(i,j)} = C_{(i,j)}$. Given a tree decomposition, all $a_{(i,j),(u,v)}$'s and $b_{(i,j)}$'s are fixed. This matrix equation can also be expressed in the following block form:

$$\begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,p} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,p} \\ \vdots & \vdots & & \vdots \\ A_{p,1} & A_{1,2} & \cdots & A_{p,p} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_p \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_p \end{pmatrix} \tag{3.5}$$

where $A_{i,u} = \begin{pmatrix} a_{(i,1),(u,1)} & \cdots & a_{(i,1),(u,b_u-1)} \\ \vdots & & \vdots \\ a_{(i,b_i-1),(u,1)} & \cdots & a_{(i,b_i-1)(u,b_u-1)} \end{pmatrix}$, $C_u = \begin{pmatrix} x_{(u,1)} \\ \vdots \\ x_{(u,b_u-1)} \end{pmatrix}$, and $B_i = \begin{pmatrix} b_{(i,1)} \\ \vdots \\ b_{(i,b_i-1)} \end{pmatrix}$.

The block Gauss-Seidel method [48] can be used to solve (3.5); i.e.,

$$C_i^{(m+1)} = A_{i,i}^{-1} \left(B_i - \sum_{j=1}^{i-1} A_{i,j} C_j^{(m+1)} - \sum_{j=i+1}^{P} A_{i,j} C_j^{(m)} \right) \qquad (3.6)$$
$$i = 1, \ldots, P, \; m \geq 0$$

where superscript $^{(m)}$ indicates the mth step of relaxation. Starting from any initial guess of $C_i^{(0)}$, this method always converges, as indicated in the following theorems. The proofs of these theorems are given in section 3.7.

Theorem 3.2. The $\sum_{i=1}^{N} b_i \times \sum_{i=1}^{N} b_i$ matrix R with elements $R_{(i,j),(u,v)} = R_{(i,j)}^{(u,v)}$ is symmetric and positive-definite [41]. ∎

Theorem 3.3. The matrix A in (3.5) is symmetric and positive-definite. ∎

Corollary 3.4. Matrix A is nonsingular, so the solution of (3.5) exists and is unique. ∎

Theorem 3.5. Let A be an $n \times n$ real symmetric matrix. Then the block Gauss-Seidel Method is convergent for all initial $x_i^{(0)}$'s if and only if A is positive-definite. [48] ∎

Corollary 3.6. The scheme (3.6) converges for all initial $C_i^{(0)}$'s. ∎

3.3 The LRD Algorithm

The system of linear equations (3.5) can be solved by another algorithm which only uses local information during the relaxation process. Given an initial load distribution for a tree decomposition of an RC network, the delays of the secondary nodes are calculated using algorithm TREE. The relaxation process starts by scanning through the split primary nodes $N_{1,\ldots,p}$, and checking if the corresponding secondary nodes give the same values of delay. If they do not, node capacitances are distributed improperly somewhere in the network. Although nothing is known as to where this improper distribution happens, we can always adjust the local distribution of $C_{(i,j)}$'s so that the delays of equivalent secondary nodes are equal for the primary node presently under investigation. The adjustment is done as follows. Suppose N_i is the current node under investigation, and $T_{(i,1)}, \ldots, T_{(i,b_i)}$ are not all equal. Based upon case 3 of Theorem 2.2, the delay of node N_i at this relaxation step is given by

$$T_i = \frac{\sum_{j=1}^{b_i} \frac{T_{(i,j)}}{R_{(i,j)}}}{\sum_{j=1}^{b_i} \frac{1}{R_{(i,j)}}} \tag{3.7}$$

where $R_{(i,j)}$ is the source resistances of node $N_{(i,j)}$, and remains fixed during the relaxation process. For the dominant-path decomposition scheme to be described in Chapter 4, the $R_{(i,j)}$ values are determined at the time when the network is decomposed. Let $\Delta_{C_{(i,j)}}$ be the amount of load adjustment for the secondary node $N_{(i,j)}$. Then

$$\Delta_{C_{(i,j)}} = \frac{T_i - T_{(i,j)}}{R_{(i,j)}}. \tag{3.8}$$

The constraint $\sum_{j=1}^{b_i} C_{(i,j)} = C_i$ is satisfied automatically since $\sum_{j=1}^{b_i} \Delta_{C_{(i,j)}} = \sum_{j=1}^{b_i} \frac{T_i - T_{(i,j)}}{R_{(i,j)}} = \left(\sum_{j=1}^{b_i} \frac{1}{R_{(i,j)}}\right) T_i - \sum_{j=1}^{b_i} \frac{T_{(i,j)}}{R_{(i,j)}} = 0$. To maintain consistency, this adjustment of $C_{(i,j)}$'s must be propagated to other nodes in the same tree so that their delays may be updated accordingly $(\Delta_{T_{(u,v)}}|_{(i,j)} \overset{\text{def}}{=} \Delta_{T_{(u,v)}}|_{\Delta_{C_{(m,n)}}=0, \forall m \neq i, \forall n \neq j} = R_{(u,v)}^{(i,j)} \Delta_{C_{(i,j)}})$. Consider the following two conditions:

1. Before a node is combined with other nodes using (3.7), the delay of the node is fully updated.

2. No two equivalent secondary nodes lie in the same tree, i.e., $R_{(i,j)}^{(i,v)} = 0$

if $j \neq v$, for $j, v = 1, \ldots, b_i$, and $i = 1, \ldots, P$. (3.9)

Theorem 3.7. If both conditions of (3.9) are satisfied, then the relaxation process based upon (3.7) and (3.8) is equivalent to the block Gauss-Seidel method of (3.6), the convergence of which is guaranteed. ∎

The proof of this theorem is also given in section 3.7. Condition 1 of (3.9) can always be satisfied if, whenever there is a change in $C_{(i,j)}$, this information is propagated to all the nodes in the same tree. However, this is a very time-consuming process. A more efficient approach is to accumulate the changes as the scan process goes along. The delay of a node is not updated until it is scanned. Instead of scanning through split primary nodes, the corresponding secondary nodes are visited in a depth-first manner [2] for each tree subnetwork. This algorithm, called LRD (Load ReDistribution), is described in the following pseudo-code.

```
procedure LRD;
var source:secondary_node; "source of the network"
begin
    function scan(A:secondary_node; T0:delay) =capacitance;
    var $\sum_T$:delay; $\sum_C$,c1:capacitance; S:secondary_node;
    begin
        "A.primary: corresponding primary node"
        "A.sons: succeeding nodes"
        "A.R: source resistance $R_A$"
        "A.$\Delta_C$: capacitance adjustment $\Delta_{C_A}$, (3.8) "
        "A.T: delay $T_A$"
        A.T:=A.T+T0;                                    ...... (a)
        combine(A.primary);       " (3.7) & (3.8) "
        if A.sons = nil then scan:=A.$\Delta_C$
        else begin
            $\sum_T$:=T0+A.$\Delta_C$*A.R;                ...... (b)
            $\sum_C$:=0.0;
            for S ∈ A.sons do begin
                c1:=scan(S,$\sum_T$+A.R*S.c2);
```

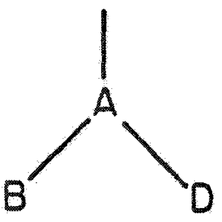


Figure 3.2. Illustration of the idea behind procedure SCAN

```
                $\sum_C$:= $\sum_C$+c1;
                S.c2:= $\sum_C$;                          ...... (c)
                $\sum_T$:= $\sum_T$+c1*A.R;               ...... (d)
                A.T:=A.T+c1*A.R;                          ...... (e)
            end;
            for S ∈ A.sons do S.c2:= $\sum_C$ −S.c2;      ...... (f)
            scan:=A.$\Delta_C$+$\sum_C$;
        end;
    end; "scan"
begin "LRD"
    while not converge do
    for S ∈ source.sons do scan(S,0.0);
end; "LRD"
```

Procedure COMBINE implements (3.7) and (3.8). The idea behind procedure SCAN is indicated by the following relationship among the three nodes A, B and D of Figure 3.2:

- $\Delta_{T_B}|_{A,D}(\overset{\text{def}}{=}\Delta_{T_B}|_{\Delta_{C_i}=0, i\neq A,D}) = R_{B,A}\Delta_{C_A}+\Delta_{B,D}\Delta_{C_D} = R_A(\Delta_{C_A}+\Delta_{C_D}) = \Delta_{T_A}|_{A,D}$.
- $\Delta_{T_A}|_{B,D} = R_{A,B}\Delta_{C_B} + R_{A,D}\Delta_{C_D} = R_A(\Delta_{C_B}+\Delta_{C_D})$.

The first equation above suggests the accumulation and propagation of Δ_T (parameter $T0$ of procedure SCAN) from the driving end towards the loading ends as the nodes of a tree are scanned in a depth-first manner. The second equation suggests the accumulation of Δ_C (returned by procedure SCAN) from the loading ends towards the driving end. If branch B is scanned before branch D, then $\Delta_{T_D}|_B$ is in effect at the present scan. On

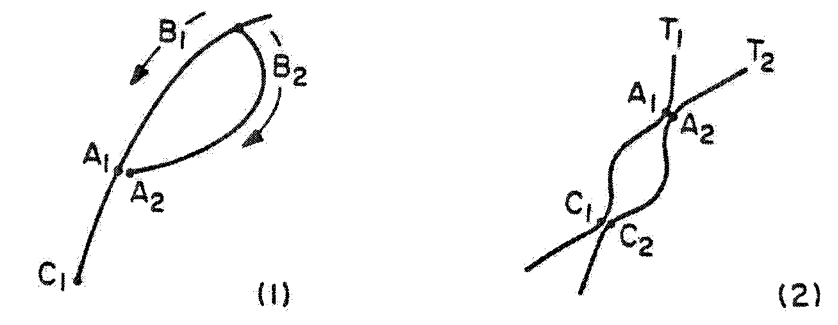


Figure 3.3. Two situations in which condition 1 of (3.9) is not satisfied

the other hand, the value of Δ_{C_D} is stored at variable B.c2 to update T_B when node B is scanned at the next relaxation step.

Condition 1 of (3.9) is satisfied in most cases, except the two situations shown in Figure 3.3. In case 1 (Figure 3.3.a), nodes A_1 and A_2 are equivalent, and they are in the same tree as node C_1. Suppose branch B_1 is scanned before branch B_2. Then, at the time when node C_1 is scanned, $T_{(C,1)}$ is only partially updated (Since $A_1 \prec_s C_1 \prec_s A_2$, the adjustment of $T^{(m)}_{(C,1)}$ due to $\Delta^{(m)}_{C_{(A,1)}}$ is in effect, but that due to $\Delta^{(m)}_{C_{(A,2)}}$ is not. Note that $A_1 \prec_s C_1$ means that node A_1 is scanned before node C_1). In case 2 (Figure 3.3.b), nodes A_1 and C_1 are in a same tree, while A_2 and C_2 are in another. Suppose tree T_1 is scanned before tree T_2. Then at the time when C_1 is scanned, node C_2 is not fully updated yet ($A_1 \prec_s C_1 \prec_s A_2 \prec_s C_2$, so the adjustment of $T_{(C,2)}$ due to $\Delta_{C_{(A,2)}}$ is not in effect).

Algorithm LRD is applied regardless of whether the two conditions of (3.9) are satisfied or not. Convergence of this algorithm has always been observed, although the convergence rate is slower when condition 2 is not satisfied. Note that only secondary nodes that correspond to split primary nodes need to be considered in the relaxation process. The delays of other nodes can be calculated after the relaxation process terminates.

Theorem 3.8. The time complexity of algorithm LRD is $O(l \cdot q)$, where l is the number of relaxation steps used, and $q = \sum_{i=1,b_i \neq 1}^{N} b_i = \sum_{i=1}^{P} b_i$ is the number of secondary nodes

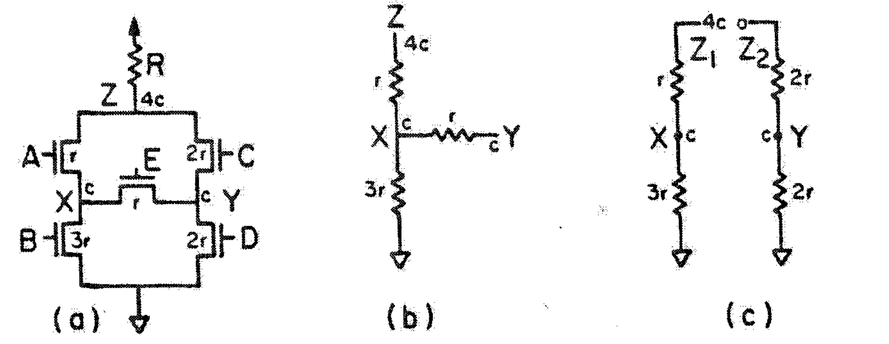


Figure 3.4. An nMOS circuit to illustrate algorithms TREE and LRD

corresponding to split primary nodes.

Proof: COMBINE (p) is of time complexity $O(|S_p|)$ and, at each relaxation step, COMBINE is called exactly once for each split primary node. The other part of the code SCAN($A, T0$) takes time $O(|D_A|)$, which can be easily checked by induction. The theorem follows from the fact that $\sum_p(|S_p|) = |D_{source}| = q$. ∎

The number of relaxation steps required depends on the accuracy aimed at. Usually four or five steps are enough to bring the error down to within 10%. The correctness of algorithm LRD is proved in section 3.6.

3.4 Examples of Delay Calculation

Example 3.9. Consider the nMOS circuit shown in Figure 3.4.a. Let A, B, C, D, E indicate the input nodes (also transistors) of the circuit, and X, Y, Z indicate the internal nodes. The values marked by the transistors are their ON-resistances and those by the nodes their capacitances. In order for this circuit to function properly, the condition $R \gg 6r$ must be satisfied. Initially, transistors D and B are OFF, and transistors A and C are ON, so all the internal nodes are at voltage level VDD. At time 0^+, the topology of the network changes, and various internal nodes are pulled down towards a voltage level very close to GND. The delays of these nodes are determined using algorithms TREE and

LRD. As a shorthand, let $(b_1, b_2, b_3, b_4, b_5)$ denote the logic level of the five input nodes A, B, C, D, E, where $b_{1,\dots,5} \in \{0,1\}$.

1. $(1,1,0,0,1)$: The corresponding RC network in this case is a tree (Figure 3.4.b). Therefore, the delays can be calculated directly using algorithm TREE, and no relaxation process is required.

   - (load): $C_y^L = c$, $C_z^L = 4c$, and $C_x^L = C_y^L + C_z^L + c = 6c$.
   - (delay): $T_x = 3r \cdot 6c = 18rc$, $T_y = T_x + r \cdot c = 19rc$, and $T_z = T_x + r \cdot 4c = 22rc$.

2. $(1,1,1,1,0)$: The network in this case is not a tree, so some decomposition is necessary. Say, node capacitance C_z is split into two parts: $C_{(z,1)}$, and $C_{(z,2)}$ (Figure 3.4.c). The source resistances of various nodes are $R_x = 3r$, $R_y = 2r$, $R_{(z,1)} = R_x + r = 4r$, and $R_{(z,2)} = R_y + 2r = 4r$. Assume that $C_{(z,1)} = 4c$, and $C_{(z,2)} = 0$ initially.

   a. Initialization of delays using TREE:

   - $C_{(z,1)}^{L(0)} = 4c$, $C_x^{L(0)} = 5c$ (tree 1), $C_{(z,2)}^{L(0)} = 0$, and $C_y^{L(0)} = c$ (tree 2).
   - $T_x^{(0)} = 15rc$, $T_{(z,1)}^{(0)} = 19rc$ (tree 1), $T_y^{(0)} = 2rc$, and $T_{(z,2)}^{(0)} = 2rc$ (tree 2).

   b. LRD, step 1:

   - Check node Z (forward): $T_z^{(1)} = \dfrac{\dfrac{T_{(z,1)}^{(0)}}{R_{(z,1)}} + \dfrac{T_{(z,2)}^{(0)}}{R_{(z,2)}}}{\dfrac{1}{R_{(z,1)}} + \dfrac{1}{R_{(z,2)}}} = \dfrac{19+2}{2} rc = 10.5rc$. Then, $\Delta_{C_{(z,1)}}^{(1)} = \dfrac{10.5-19}{4} c = -2.125c$, and $\Delta_{C_{(z,2)}}^{(1)} = \dfrac{10.5-2}{4} c = 2.125c$.
   - Correct T_x and T_y due to the changes of $C_{(z,1)}$ and $C_{(z,2)}$ (backward): $T_x^{(1)} = T_x^{(0)} + R_x \Delta_{C_{(z,1)}} = (15 - 3 \cdot 2.125)rc = 8.625rc$, and $T_y^{(1)} = T_y^{(0)} + R_y \Delta_{C_{(z,2)}} = (2 + 2 \cdot 2.125)rc = 6.25rc$.

One relaxation step gives us the exact delays of all the nodes. This is in general true if there is only one node to split, and condition 2 of (3.9) is satisfied. As a comparison, the delays of node X, Y, and Z are calculated again using the techniques presented in Chapter 2.

The triple (R, C, D) is used to represent a two-port RC network, where R, C, D are the parameters described in Theorem 2.2. Let $-$ and $\|$ denote series and parallel connections, respectively, and $-$ has precedence over $\|$. The RC networks between nodes X, Y, Z and GND can be individually represented and reduced as follows:

$$\begin{aligned}
(Z):\quad & (3r, c, 3rc) - (r, 2c, 2rc) \parallel (2r, c, 2rc) - (2r, 2c, 4rc) \\
&= (4r, 3c, 11rc) \parallel (4r, 3c, 10rc) \\
&= (2r, 6c, 10.5rc) \\
(X):\quad & (3r, c, 3rc) \parallel (2r, c, 2rc) - (2r, 4c, 8rc) - (r, 0, 0) \\
&= (3r, c, 3rc) \parallel (5r, 5c, 18rc) \\
&= (1.875r, 6c, 8.625rc) \\
(Y):\quad & (2r, c, 2rc) \parallel (3r, c, 3rc) - (r, 4c, 4rc) - (2r, 0, 0) \\
&= (2r, c, 2rc) \parallel (6r, 5c, 19rc) \\
&= (1.5r, 6c, 6.25rc)
\end{aligned}$$

By corollary 2.7, $T_z = 10.5rc$, $T_x = 8.625rc$, and $T_y = 6.25rc$.

3. $(1,1,1,1,1)$: Suppose the tree decomposition of Figure 3.4.d is selected. Both node X and Y are split, resulting in two chains: GND - Y_1 - Z - X_2 - Y_2, and GND - X_1, respectively. The source resistances of various nodes are $R_{(y,1)} = 2r$, $R_z = 4r$, $R_{(x,2)} = 5r$, $R_{(y,2)} = 6r$, and $R_{(x,1)} = 3r$. Assume that $C_{(x,1)} = 0$, $C_{(x,2)} = c$, $C_{(y,1)} = c$, and $C_{(y,2)} = 0$ initially.

a. Initialization of delays using TREE:

- $C^{L(0)}_{y,2} = 0$, $C^{L(0)}_{x,2} = c$, $C^{L(0)}_{z} = 5c$, $C^{L(0)}_{(y,1)} = 6c$ (tree 1), and $C^{L(0)}_{(x,1)} = 0$ (tree 2).
- $T^{(0)}_{(y,1)} = 12rc$, $T^{(0)}_{z} = 22rc$, $T^{(0)}_{(x,2)} = 23rc$, $T^{(0)}_{(y,2)} = 23rc$ (tree 1), and $T^{(0)}_{(x,1)} = 0$ (tree 2).

b. LRD, step 1: Suppose tree 1 is scanned before tree 2.

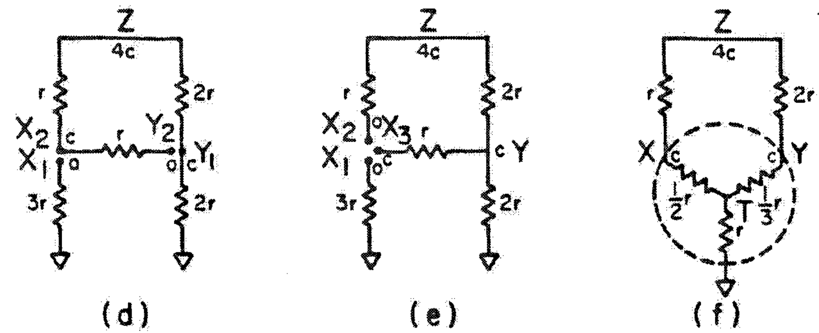


Figure 3.4. (continued) An nMOS circuit to illustrate algorithms TREE and LRD

- Check node Y_1: $T_y^{(1)} = \dfrac{\frac{12}{2}+\frac{23}{6}}{\frac{1}{2}+\frac{1}{6}} rc = 14.75rc$. Then $\Delta^{(1)}_{C_{(y,1)}} = \dfrac{14.35-12}{2}c = 1.375rc$, $\Delta_{C_{(y,2)}} = \dfrac{14.75-23}{6}c = -1.375c$. Also $\sum_T = 14.75rc - 12rc = 2.75rc$.

- Check node X_2 (forward): $T_{(x,2)}$ is first updated to $23rc + \sum_T = 25.75rc$ (the effect of $\Delta^{(1)}_{C_{(y,1)}}$ on $T_{(x,2)}$). Then $T_x^{(1)} = \dfrac{\frac{25.75}{5}+0}{\frac{1}{5}+\frac{1}{3}} rc = 9.66rc$. Thus, $\Delta^{(1)}_{C_{(x,1)}} = \dfrac{9.66-0}{3}c = 3.22c$, and $\Delta^{(1)}_{C_{(x,2)}} = \dfrac{9.66-25.75}{5}c = -3.22c$. Also $\sum_T = (2.75 + (9.66-25.75))rc = -13.34rc$ (or $(9.66-23)rc$).

- Update $T_{(y,2)}$ (forward): $T_{(y,2)}$ is first updated to $14.75rc + \sum_T = 1.41rc$. As node Y is already checked, nothing needs to be done further. On the other hand, an end of the tree is reached, so the backtrack phase starts with $\sum_C = \Delta_{C_{(y,2)}} = -1.375c$.

- Update $T_{(x,2)}$ (backward): $T^{(1)}_{(x,2)}$ is updated to $9.66rc + R_{(x,2)} \sum_C = (9.66 - 5 \cdot 1.375)rc = 2.78rc$. Also $\sum_C$ is accumulated up to $-1.375c + \Delta^{(1)}_{C_{(x,2)}} = -4.595c$.

- Update $T_{(y,1)}$ (backward): $T^{(1)}_{(y,1)}$ is updated to $14.75rc + R_{(y,1)} \sum_C = (14.75 - 2 \cdot 4.595)rc = 5.56rc$.

- Nothing is done for tree 2 since node X has already been scanned.

In summary, after the first step of relaxation, $C_{(x,1)} = 3.22c$, $C_{(x,2)} = -2.22c$, $C_{(y,1)} = 2.38c$, $C_{(y,2)} = -1.38c$, $T_{(x,1)} = 9.66rc$, $T_{(x,2)} = 2.78rc$, $T_{(y,1)} = 5.56rc$, and $T_{(y,2)} = 1.41rc$. The delay of every node is fully updated.

step	0	1	2	3	4	5	10	≥ 12
$C_{(x,1)}$	0	3.22	2.23	2.58	2.49	2.54	2.56	2.56
$C_{(x,2)}$	1	-2.22	-1.23	-1.58	-1.49	-1.54	-1.56	-1.56
$T_{(x,1)}$	0	9.66	6.69	7.75	7.48	7.62	7.68	7.69
$T_{(x,2)}$	23	2.78	9.29	7.19	7.86	7.65	7.69	7.69
$C_{(y,1)}$	1	2.38	1.86	1.97	1.89	1.88	1.83	1.83
$C_{(y,2)}$	0	-1.38	-0.86	-0.97	-0.89	-0.88	-0.83	-0.83
$T_{(y,1)}$	12	5.56	7.54	6.84	7.01	6.92	6.88	6.87
$T_{(y,2)}$	23	1.41	8.43	6.22	6.97	6.76	6.86	6.87

Table 3.1. Values of C's and T's at the end of some relaxation steps

The results at the end of some relaxation steps are summarized at Table 3.1. Note that node Z is not split, and thus is not involved in the relaxation process. The value of T_z is determined after the process terminates. In this case, T_y is $6.87rc$, and $C_z^L = C_{y,2} + C_{x,2} + C_z = (-1.56 - 0.83 + 4)c = 1.61c$, so $T_z = (6.87 + 2 \cdot 1.61)rc = 10.09rc$. Consider another tree decomposition shown in Figure 3.4.e: node X is split into three parts, and tree 1 is not a simple chain. The source resistances of various nodes are $R_{(x,1)} = 3r$, $R_{(x,2)} = 5r$, $R_{(x,3)} = 3r$, and $R_y = 2r$. Assume that $C_{(x,1)} = 0$, $C_{(x,2)} = 0$, and $C_{(x,3)} = c$ initially.

a. Initialization of delays using TREE: $T_y^{(0)} = 12rc$, $T_{(x,2)}^{(0)} = 20rc$, $T_{(x,3)}^{(0)} = 13rc$ (tree 1), and $T_{(x,1)}^{(0)} = 0$ (tree 2).

b. LRD, step 1: Assume that tree 1 is scanned before tree 2, and branch 1 of tree 1 is scanned before branch 2.

- Check node X_2: $T_x^{(1)} = \dfrac{0 + \frac{20}{5} + \frac{13}{3}}{\frac{1}{3} + \frac{1}{5} + \frac{1}{3}} = 9.61rc$. Then $\Delta_{C_{(x,2)}}^{(1)} = \dfrac{9.61 - 20}{5}c =$

$-2.08c$, $\Delta^{(1)}_{C_{(x,3)}} = -1.13c$, and $\Delta^{(1)}_{C_{(x,1)}} = 3.21c$.

- Back to node Y: $\sum_C = \Delta^{(1)}_{C_{(x,2)}} = -2.08c$, and $\sum_T = R_y \cdot \sum_C = -4.16rc$.
- Update node X_3: $T^{(1)}_{(x,3)}$ is updated to $9.61rc + \sum_T = 5.45rc$. Nothing is done further, except that $\Delta^{(1)}_{C_{(x,3)}}$ is recorded in branch 1 of tree 1 to update $T_{(x,2)}$ at step 2.

After step 1, $C_{(x,1)} = 3.21c$, $C_{(x,2)} = -2.08c$, $C_{(x,3)} = -0.13c$, $T_{(x,1)} = 9.61rc$, $T_{(x,2)} = 9.61rc$, and $T_{(x,3)} = 5.45rc$. $T_{(x,2)}$ is only partially updated (the actual value is $9.61rc - R_y \cdot \Delta_{C_{(x,3)}} = 7.35rc$).

c. LRD, step 2:

- Check node X_2: $T_{(x,2)}$ is updated to $7.35rc$. Then $T_x^{(2)} = \dfrac{\frac{9.61}{3} + \frac{7.35}{5} + \frac{5.45}{3}}{\frac{13}{15}} rc = 7.49rc$. Thus $\Delta^{(2)}_{C_{(x,2)}} = 0.03c$, and $\Delta^{(2)}_{C_{(x,3)}} = 0.68c$.
- Update node X_3: $T_{(x,3)}$ is updated to $7.49rc + R_y \cdot \Delta^{(2)}_{C_{(x,2)}} = 7.55rc$. $T_{(x,2)}$ is not fully updated, and $\Delta^{(2)}_{C_{(x,3)}}$ is recorded for step 3.

step	0	1	2	3	4	5	6	≥ 10
$C_{(x,1)}$	0	3.21	2.50	2.61	2.57	2.57	2.57	2.56
$C_{(x,2)}$	0	-2.08	-2.05	-2.25	-2.31	-2.35	-2.37	-2.38
$C_{(x,3)}$	1	-0.13	0.55	0.64	0.74	0.78	0.80	0.82
$T_{(x,1)}$	0	9.61	7.50	7.83	7.72	7.72	7.70	7.69
$T_{(x,2)}$	20	9.61	7.50	7.83	7.72	7.72	7.70	7.69
$T_{(x,3)}$	13	5.46	7.55	7.42	7.60	7.64	7.67	7.69
$T'_{(x,2)}{}^*$	20	7.36	8.85	8.02	7.91	7.79	7.75	7.69

Table 3.2. Values of C's and T's at the end of some relaxation steps

Table 3.2 summarizes the results at the end of some relaxation steps. Note that item $^*T'_{(x,2)}$ in the table indicates the fully updated value of $T_{(x,2)}$. T_y and T_x are calculated after the process terminates: $T_y = T_{(x,3)} - r \cdot C_{(x,3)} = 6.87rc$, and $T_x = T_{(x,2)} - r \cdot C_{(x,2)} =$

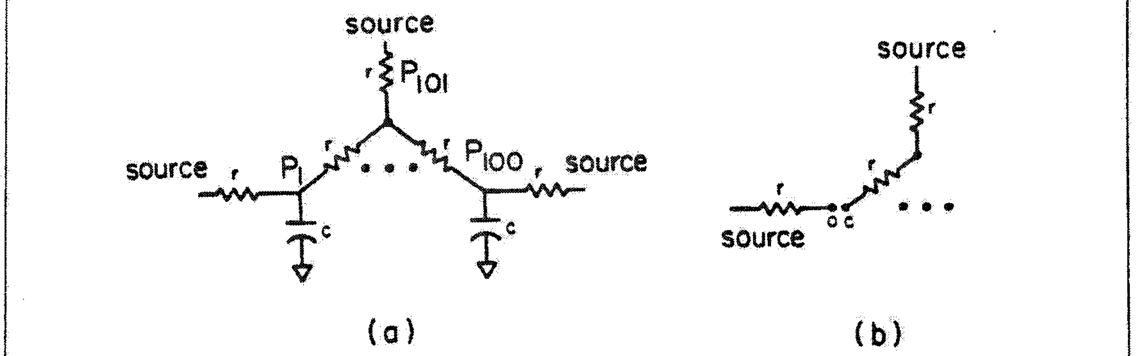


Figure 3.5. Example for which the Jacobi method diverges

10.09rc. Note that (3.1) is applied for the above calculation. To calculate the delay of node Z directly, a $\Delta - Y$ transformation among the three nodes GND, X, Y is necessary. One phantom node T is generated by this transformation, and the resulting network is shown in Figure 3.4.f. Note that this transformation is not necessary for node X and Y. The network between nodes X, Y, Z and GND are as follows:

$$(Z): \quad (r,0,0) - \{(0.5r,c,0.5rc) - (r,2c,2rc) \parallel (0.33r,c,0.33rc) - (2r,2c,4rc)\}$$

$$(X): \quad (3r,c,3rc) \parallel \{(2r,c,2rc) - ((2r,4c,8rc) - (r,0,0) \parallel (r,0,0))\}$$

$$(Y): \quad (2r,c,2rc) \parallel \{(3r,c,3rc) - ((r,4c,4rc) - (2r,0,0)) \parallel (r,0,0)\}$$

It can be easily checked that $T_x = 7.69rc$, $T_y = 6.87rc$, and $T_z = 10.09rc$. ∎

Before finishing this section, we give an example to indicate that a Jacobi-like method [44] (update the delays simultaneously after all the nodes are scanned) may lead to divergence of the relaxation scheme.

Example 3.10. Consider the circuit shown in Figure 3.5.a. There are 101 branches incident on node N. Paths $P_{1,\cdots,100}$ are all identical: P_i connects to node N_i, which in turn connects the source. Path P_{101} connects to the source directly. Suppose that all nodes $N_{1,\ldots,100}$ are split into two parts (Figure 3.5.b), and the network is decomposed into 101 tree subnetworks. Suppose the initial guess of the load distribution is such that

$C_{(N_i,1)} = c$, and $C_{(N_i,2)} = 0$ for $i = 1, \cdots, 100$. Then $C_N^{L(0)} = 101c$, $T_N^{(0)} = 101rc$, and $T_{N_i,1}^{(0)} = 102rc$, and $T_{(N_i,2)} = 0$. At the first step of relaxation, $T_{N_i}^{(1)} = \frac{102}{2}rc = 51rc$, and $\Delta_{C_{(N_i,1)}}^{(1)} = -25.5c$. Reflecting all these changes of capacitances back to node N, $T_N^{(1)}$ becomes $101rc - 100 \cdot r \cdot 25.5c = -2449rc$. At the second step of relaxation, $T_{(N_i,1)}^{(2)} = -2473.5rc$, $T_{N_i}^{(2)} = \frac{-2473.5 + 102}{2}rc = -1185.75rc$. Then $\Delta_{C_{(N_i,1)}}^{(2)} = 643.875c$. Reflecting these changes back to node N, $T_N^{(2)}$ becomes $-2449rc + 100 \cdot r \cdot 643.875c = 61938.5rc$. It can be easily seen that the absolute values of $\Delta_{C_{N_i}}$'s, $T_{N_i,1}$'s and T_N's will grow indefinitely, and the process diverges. On the other hand, both conditions of (3.9) are satisfied by this example, so algorithm LRD converges. ∎

3.5 Summary

Complementary to the transfer-equation formulation of Chapter 2, "delay" is reformulated as the fixed point of a collective process among all nodes in an RC network. The technique of tree decomposition and load redistribution combines these two formulations in a nice way. The proof of positive-definiteness which leads to the convergence of the LRD algorithm indicates the robustness of this technique.

Our transistor-level timing model is mathematically well-founded. To apply this model to digital MOS circuits, the following four assumptions are made:

1. MOS circuits are approximated by RC networks.
2. Transistors are approximated by linear resistors, and inputs to transistor gates are approximated by step waveforms.
3. Each approximating RC network is assumed to be driven by one and only one source.
4. The area criterion of (1.1′) is used as the definition of delay.

These four assumptions are justified in various sections of Chapter 2. The application of the TREE and LRD algorithms to timing simulation of digital MOS circuits is presented in the next chapter.

3.6 Appendix: Correctness of the LRD Algorithm

To discuss the LRD algorithm of section 3.3 in more detail, the following terminology is used. Referring to Figure 3.6, let $C(B)$ be the set of nodes along the path from the source to node B, excluding node B itself. If a node X is in $C(B)$, then X is called an "ancestor" of B, and this relation is denoted by $X \prec B$. A is called the "parent" of B if A is adjacent to B, and $A \prec B$. Every node N except the source has a unique parent which is denoted by $p(N)$. Let $\text{com}(B,C)$ denote the common ancestor of B and C such that every other common ancestor of B and C is also an ancestor of $\text{com}(B,C)$. Given two nodes B and C, $\text{com}(B,C)$ always exists, and is unique. Also $R_{B,C} = R_{\text{com}(B,C)}$. Let $S(A) = \{X | p(X) = A\}$, and $D(A) = \{X | A \preceq X\}$. With $S(A)$ is associated an ordering $O_A : S(A) \rightarrow \{1, \ldots, |S(A)|\}$, where $|S(A)|$ is the size of $S(A)$. A.SONS in the above code corresponds to the set $S(A)$, and O_A indicates the ordering of the nodes in A.SONS. Let $E(B) = \{X | X \in S(p(B)),\ O_{p(B)}(X) < O_{p(B)}(B)\}$, and $F(B) = \{X | X \in S(p(B)),\ O_{p(B)}(B) < O_{p(B)}(X)\}$. Define a total ordering $\prec_s$ among the nodes in a tree network as follows. For any two distinct nodes B and C, let $A = \text{com}(B,C)$,

1. If $B \prec C$, i.e. $A = B$, then $B \prec_s C$.
2. If $C \prec B$, then $C \prec_s B$.
3. If $B, C \in S(A)$, then $B \prec_s C$ if $O_A(B) < O_A(C)$, and vice versa.
4. Otherwise, let B_1 and C_1 be the two nodes such that $B \preceq B_1$, $C \preceq C_1$, and $B_1, C_1 \in S(A)$. Then $B \prec_s C$ if $B_1 \prec_s C_1$, and vice versa.

That is to say, if node B is scanned before node C in the pseudo-code of LRD (section 3.3), then $B \prec_s C$, and vice versa. Let $Q(A) = \{X | X \prec_s A\}$ be the set of (secondary) nodes that is scanned before A. The following relations among various sets defined above are noted.

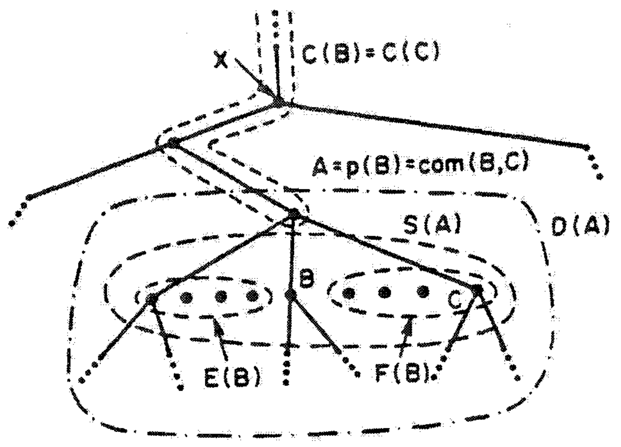


Figure 3.6. Terminology defined for a tree network

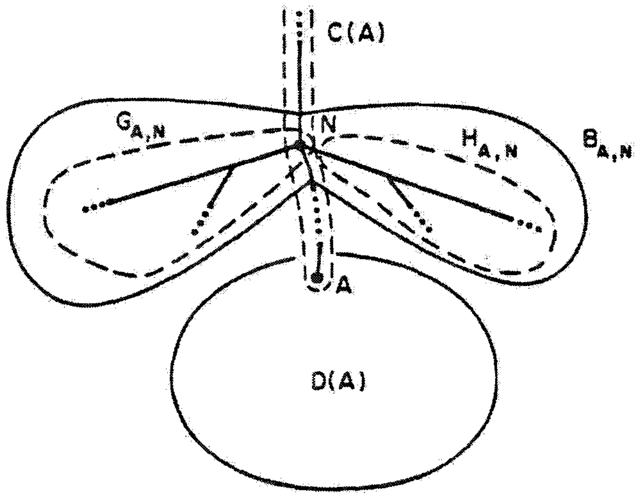


Figure 3.7. Classification of nodes in a tree network (wrt. node A)

Lemma 3.11. $D(A) = \{A\} \bigcup \left(\bigcup_{x \in S(A)} D(x) \right)$. ∎

Lemma 3.12. If A is the parent of S, i.e., $A = p(S)$, then $Q(S) = Q(A) \bigcup \{A\} \bigcup \left(\bigcup_{x \in E(S)} D(x) \right)$. ∎

Let A be a node in a tree network, and node N be any ancestor of node A (refer to Figure 3.7). Let $B_{A,N} = \{X | \text{com}(A,X) = N\}$, $G_{A,N} = B_{A,N} \cap Q(A)$, and $H_{A,N} = B_{A,N} - G_{A,N}$. With respect to node A, all nodes in the tree are classified into three sets: $\bigcup_{N \in C(A)} G_{A,N}$, $D(A)$, and $\bigcup_{N \in C(A)} H_{A,N}$. Accordingly, Δ_{T_A} is split into three terms as follows. As $T_A = \sum_x R_{A,x} C_x = \sum_x R_{\text{com}(A,x)} C_x$,

$$\begin{aligned}
\Delta T_A &= \Delta \left(\sum_x R_{\text{com}(A,x)} C_x \right) \\
&= \sum_x R_{\text{com}(A,x)} \Delta C_x \\
&= \sum_{N \in C(A)} R_N \left(\sum_{x \in B_{A,N}} \Delta C_x \right) + R_A \sum_{x \in D(A)} \Delta C_x \\
&= \sum_{N \in C(A)} R_N \left(\sum_{x \in G_{A,N}} \Delta C_x + \sum_{x \in H_{A,N}} \Delta C_x \right) + R_A \sum_{x \in D(A)} \Delta C_x \\
&= \sum_{N \in C(A)} R_N \left(\sum_{x \in G_{A,N}} \Delta C_x \right) + R_A \sum_{x \in D(A)} \Delta C_x + \sum_{N \in C(A)} R_N \left(\sum_{x \in H_{A,N}} \Delta C_x \right)
\end{aligned}$$

The three terms split from Δ_{T_A} are as follows:

$$\begin{aligned}
\Delta_1 T_A &= \sum_{N \in C(A)} R_N \left(\sum_{x \in G_{A,N}} \Delta C_x \right) \\
\Delta_2 T_A &= R_A \left(\sum_{x \in D(A)} \Delta C_x \right) \\
\Delta_3 T_A &= \sum_{N \in C(A)} R_N \left(\sum_{x \in H_{A,N}} \Delta C_x \right).
\end{aligned}$$

Lemma 3.13. $Q(S) = \bigcup_{N \in C(S)} G_{S,N}$. Moreover, if $A = p(S)$, then

$$\bigcup_{N\in C(S)} G_{S,N} = \left(\bigcup_{N\in C(A)} G_{A,N}\right)\bigcup\{A\}\bigcup\left(\bigcup_{x\in E(S)} D(x)\right)$$

$$\bigcup_{N\in C(S)} H_{S,N} = \left(\bigcup_{N\in C(A)} H_{S,N}\right)\bigcup\left(\bigcup_{x\in F(S)} D(x)\right).$$

∎

Theorem 3.14. Refer to the pseudo-code of LRD in section 3.3. As procedure SCAN(A,T0) is called and executed at the mth relaxation step, the following statements are true.

(1) The value of $\sum_{x\in D(A)} \Delta C_x^{(m)}$ is returned by the procedure.

(2) When the procedure returns, variable $A.c2$ contains the value of $\sum_{x\in F(A)} \sum_{i\in D(x)} \Delta C_i^{(m)}$.

(3) The value of the parameter $T0$ equals $\Delta_1 T_A^{(m)} + \Delta_3 T_A^{(m-1)}$.

(4) Before entering procedure COMBINE, T_A is updated to be consistent with all the changes of $C_i^{(k)}$ for $k < m, \forall i$, and for $k = m$, $i \prec_s A$.

Proof:

1. (1) is proved by induction. If node A is a leaf, i.e., $S(A) = \phi$, then $D(A) = \{A\}$. In this case, SCAN returns $\Delta C_A^{(m)}$ ($A.\Delta_C$ in the code). Suppose $S(A) \neq \phi$, and (1) is true for all nodes $X \in D(A)$. In this case, SCAN returns $A.\Delta_C + \sum_C = \Delta C_A^{(m)} + \sum_{S\in S(A)}\text{SCAN}(S,\cdot) = \Delta C_A^{(m)} + \sum_{S\in S(A)} \sum_{i\in D(S)} \Delta C_i^{(m)}$, which equals $\sum_{i\in D(A)} \Delta C_i^{(m)}$, by Lemma 3.11.

2. By (1), SCAN($S1,\cdot$) returns $\sum_{i\in D(S1)} \Delta C_i^{(m)}$. When instruction (c) marked in the code is executed, S.c2= $\sum_{S1\in E(S)\cup\{S\}}\text{SCAN}(S1,\cdot)$. Then at the time when instruction (f) is executed, $\sum_C = \sum_{S1\in S(A)}\text{SCAN}(S1,\cdot)$, where $A = p(S)$. Thus S.c2= $\sum_{S1\in F(S)}\text{SCAN}(S1,\cdot) = \sum_{S1\in F(S)} \sum_{i\in D(S1)} \Delta C_i^{(m)}$.

3. (3) is proved by induction. If node S is connected to the source directly, then $C(S) = \phi$, and $\Delta_1 T_S^{(m)} = \Delta_3 T_S^{(m-1)} = 0$. Suppose node S is not connected di-

rectly to the source, and (3) is true for $p(S) = A$. Then by Lemma 3.13, $\Delta_1 T_S^{(m)} = \Delta_1 T_A^{(m)} + R_A(\Delta C_A^{(m)} + \sum_{x \in E(S)} \sum_{i \in D(x)} \Delta C_i^{(m)})$, and $\Delta_3 T_S^{(m-1)} = \Delta_3 T_A^{(m-1)} + R_A(\sum_{x \in F(S)} \sum_{i \in D(x)} \Delta C_i^{(m-1)})$. Thus

$$\Delta_1 T_S^{(m)} + \Delta_3 T_S^{(m-1)} = \Delta_1 T_A^{(m)} + \Delta_3 T_A^{(m-1)} + \\ R_A \Delta C_A^{(m)} + R_A \sum_{x \in E(S)} \sum_{i \in D(x)} \Delta C_i^{(m)} + R_A \sum_{x \in F(S)} \sum_{i \in D(x)} \Delta C_i^{(m-1)}.$$

By (1) and (2), $\Delta_1 T_S^{(m)} + \Delta_3 T_S^{(m-1)} = \Delta_1 T_A^{(m)} + \Delta_3 T_A^{(m-1)} + R_A \Delta C_A^{(m)}$ $+ R_A \sum_{x \in E(S)} \mathrm{SCAN}(x, \cdot) + R_A(\mathrm{S.c2})$. $\Delta_1 T_A^{(m)} + \Delta_3 T_A^{(m-1)}$ is passed as the parameter $T0$. $T0$ and $R_A \Delta C_A^{(m)}$ are added up to $\sum_T$ by instruction (b). $R_A \sum_{x \in E(S)} \mathrm{SCAN}(S, \cdot)$ is added to $\sum_T$ by instruction (d). Finally, when SCAN(S,T) is called, the parameter T equals $\sum_T + R_A(\mathrm{S.c2}) = \Delta_1 T_S^{(m)} + \Delta_3 T_S^{(m-1)}$.

4. (4) is proved by induction on m. When SCAN(A,T0) is called at the first relaxation step, $\mathrm{T0} = \Delta_1 T_A^{(1)}$ is added to T_A by instruction (a) before calling procedure COMBINE. Suppose (4) is true for $k = 1$ up to $m - 1$. In procedure COMBINE ($(m-1)$th step), $R_A \Delta C_A^{(m-1)}$ is added to T_A. Then $R_A \cdot \mathrm{SCAN}(S, \cdot) = R_A \sum_{i \in D(S)} \Delta C_i^{(m-1)}$ is added to T_A one by one by instruction (e) for $\forall S \in S(A)$. Thus all terms of $\Delta_2 T_A^{(m-1)}$ are added to T_A before the end of the $(m-1)$th relaxation step. As procedure SCAN(A,T0) is called at the mth relaxation step, $T0 = \Delta_1 T_A^{(m)} + \Delta_3 T_A^{(m-1)}$ is added to T_A (instruction (a)) before procedure COMBINE is called. ∎

3.7 Appendix: Proofs of Theorems

The following lemmas are used in the proofs of Theorem 3.2 and Theorem 3.3.

Lemma A1. Suppose A and D are two real symmetric $n \times n$ matrices, and X is an $n \times n$ nonsingular matrix. If $A = XDX^T$, then A is positive-definite if and only if D is positive-definite [44]. ∎

Lemma A2. The principal minors of a positive-definite matrix are also positive-definite [44]. ∎

Theorem 3.2′. Suppose there are N nodes in a collection of tree networks. Let R be an $N \times N$ matrix with elements $R_{i,j}$ equal to the resistance of the path between the source of node i, that is in common with the path between the source and node j. Then the matrix R is symmetric and postive-definite.

Proof:

It suffices to consider a tree network since the R matrix associated with a collection of tree networks is just the direct sum of the R matrices associated with individual trees. It is easy seen that $R_{i,j} = R_{j,i}$, so matrix R is symmetric. As the network is a tree, there are same number of branches as there are nodes (the source, or the root, is not considered as a node in this case). With each node i is associated a branch $b(i)$ connecting the node to its parent node $p(i)$. This node-to-branch mapping is one-to-one and onto. Let matrix X be defined as follows: If the path from the source to node i passes through branch $b(j)$, then $x_{i,j} = 1$; otherwise $x_{i,j} = 0$. Note that X can be obtained from $I_{N\times N}$ by a sequence of row operations $op(i)$: adding row i to all rows j such that $p(j) = i$. Starting from the source, this operation proceeds in a top down manner until the leaves of the tree are met. $op(i), \forall i$ preserves the determinant of the matrix, so $\det(X) = \det(I) = 1$, and X is nonsingular. Let D denote the diagonal matrix with diagonal element $d_{i,i} = r_{b(i)} > 0$, $r_{b(i)}$ being the resistance of branch $b(i)$. It is obvious that D is a positive-definite matrix. Check that $XDX^T = R$. Finally, by Lemma A1, R is positive-definite. ∎

Theorem 3.2. The $\sum_{i=1}^{N} b_i \times \sum_{i=1}^{N} b_i$ matrix R with elements $R_{(i,j),(u,v)} = R_{(i,j)}^{(u,v)}$ is symmetric and positive-definite.

Proof: immediate from Theorem 3.2′. ∎

Theorem 3.3. The matrix A in (3.5) is symmetric and positive-definite.

Proof:

Let E be an $Q \times N$ ($Q = \sum_{i=1}^{P}(b_i - 1)$) matrix with elements

$$e_{(i,j),u} = \begin{cases} -1 & \text{if } i = u, \text{ for } j = 1, \ldots, b_i - 1,\ i = 1, \ldots, P \\ 0 & \text{otherwise} \end{cases}$$

Consider the $\sum_{i=1}^{N} b_i \times \sum_{i=1}^{N} b_i$ matrix $X = \begin{pmatrix} I_{Q\times Q} & E \\ E^T & I_{N\times N} + E^T E \end{pmatrix}$. As $\det(X) = \det \begin{pmatrix} I & 0 \\ E^T & I \end{pmatrix} = 1$, X is nonsingular. Let $A' = XRX^T$, where R is the matrix of Theorem 3.2. Then by Lemma A1, A' is positive definite. Check that matrix A is the Qth principal minor of A'. By Lemma A2, A is positive-definite. ∎

The following lemmas are used in the proof of Theorem 3.7.

Lemma A3. Let $A_{i,i}$ be the ith diagonal block of the matrix A in (3.5). If $R_{(i,j)}^{(i,v)} = 0$ for $j \neq v$, $j, v = 1, \ldots, b_i$, then the determinant of $A_{i,i}$ equals

$$\left(\prod_{k=1}^{b_i} R_{(i,k)}\right)\left(\sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}\right) \tag{3.10}$$

where $R_{(i,k)}$ is the source resistance of node $N_{(i,k)}$.

Proof:

As $R_{(i,j)}^{(i,v)} = 0$ for $j \neq v$, the (j,v)-element of $A_{i,i}$, $a_{(i,j),(i,v)}$, equals $R_{(i,b_i)} + \delta_{j,v} R_{(i,j)}$. This lemma is proved by induction on the order of the matrix $A_{i,i}$: $n = b_i - 1$. For $n = 1$, $A_{i,i}$=$[R_{(i,1)} + R_{(i,2)}]$= $[R_{(i,1)}R_{(i,2)}(\frac{1}{R_{(i,1)}} + \frac{1}{R_{(i,2)}})]$

Suppose (3.10) is true for $n = m - 1$. Then for $n = m(= b_i - 1)$,

$$A_{i,i} = \begin{pmatrix} R_{(i,1)} + R_{(i,b_i)} & R_{(i,b_i)} & \cdots & R_{(i,b_i)} \\ R_{(i,b_i)} & R_{(i,2)} + R_{(i,b_i)} & \cdots & R_{(i,b_i)} \\ \vdots & \vdots & & \vdots \\ R_{(i,b_i)} & R_{(i,b_i)} & \cdots & R_{(i,b_i-1)} + R_{(i,b_i)} \end{pmatrix}$$
$$= \begin{pmatrix} R_{(i,1)} & R_{(i,b_i)} & \cdots & R_{(i,b_i)} \\ -R_{(i,2)} & R_{(i,2)} + R_{(i,b_i)} & \cdots & R_{(i,b_i)} \\ 0 & R_{(i,b_i)} & \cdots & R_{(i,b_i)} \\ \vdots & \vdots & & \vdots \\ 0 & R_{(i,b_i)} & \cdots & R_{(i,b_i-1)} + R_{(i,b_i)} \end{pmatrix} \tag{3.11}$$

Expanded along the first column, the determinant of $A_{i,i}$ equals $R_{(i,1)} \left(\prod_{k=2}^{b_i} R_{(i,k)}\right) \left(\sum_{k=2}^{b_i} \frac{1}{R_{(i,k)}}\right) + R_{(i,2)} \left(\prod_{k=3}^{b_i} R_{(i,k)}\right) = \left(\prod_{k=1}^{b_i} R_{(i,k)}\right) \left(\sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}\right)$. ∎

Lemma A4. Let $A_{i,i}$ be the ith diagonal block of the matrix A in (3.5), and $D_{i,i}$ be the inverse of $A_{i,i}$. If $R_{(i,j)}^{(i,v)} = 0$ for $j \neq v$, $j, v = 1, \ldots, b_i$, then

$$d_{(i,j),(i,v)} = \begin{cases} \Delta_{(i,j)} \left(\sum_{k=1,k\neq j}^{b_i} \frac{1}{R_{(i,k)}}\right) & \text{if } j = v \\ \frac{-\Delta_{(i,j)}}{R_{(i,v)}} & \text{otherwise} \end{cases} \tag{3.12}$$

where $d_{(i,j),(i,v)}$ is the (j,v)-element of $D_{i,i}$, and $\Delta_{(i,j)} = \frac{1}{R_{(i,j)} \sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}}$.

Proof: Immediate from inverting (3.11) using cofactors. ∎

Theorem 3.7. If both conditions of (3.9) are satisfied, then algorithm LRD is equivalent to the block Gauss-Seidel method of (3.6).

Proof:

Algorithm LRD, plus condition 1 of (3.9), implies that $T_{(i,j)}^{(m+1)} = \sum_{u=1}^{i-1} \sum_{v=1}^{b_u} R_{(i,j)}^{(u,v)} C_{(u,v)}^{(m+1)} + \sum_{u=i}^{N} \sum_{v=1}^{b_u} R_{(i,j)}^{(u,v)} C_{(u,v)}^{(m)}$, and $T_i^{(m+1)} = \frac{\sum_{k=1}^{b_i} \frac{T_{(i,k)}^{(m+1)}}{R_{(i,k)}}}{\sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}}$. Thus

$$\begin{aligned}
C_{(i,j)}^{(m+1)} &= C_{(i,j)}^{(m)} + \Delta_{C_{(i,j)}}^{(m+1)} \\
&= C_{(i,j)}^{(m)} + \frac{T_i^{(m+1)} - T_{(i,j)}^{(m+1)}}{R_{(i,j)}} \\
&= C_{(i,j)}^{(m)} + \frac{1}{R_{(i,j)}} \left(\frac{\sum_{k=1}^{b_i} \frac{T_{(i,k)}^{(m+1)} - T_{(i,j)}^{(m+1)}}{R_{(i,k)}}}{\sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}} \right) \\
&= C_{(i,j)}^{(m)} + \Delta_{(i,j)} \sum_{k=1}^{b_i} \left(\sum_{u=1}^{i-1} \sum_{v=1}^{b_u} \frac{R_{(i,k)}^{(u,v)} - R_{(i,j)}^{(u,v)}}{R_{(i,k)}} C_{(u,v)}^{(m+1)} + \sum_{u=i}^{N} \sum_{v=1}^{b_u} \frac{R_{(i,k)}^{(u,v)} - R_{(i,j)}^{(u,v)}}{R_{(i,k)}} C_{(u,v)}^{(m)} \right) \\
&= C_{(i,j)}^{(m)} + \Delta_{(i,j)} \sum_{k=1}^{b_i} \left(\sum_{u=1}^{i-1} \sum_{v=1}^{b_u-1} \frac{R_{(i,k)}^{(u,v)} - R_{(i,j)}^{(u,v)} - R_{(i,k)}^{(u,b_u)} + R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}} C_{(u,v)}^{(m+1)} + \right. \\
&\quad \left. \sum_{u=i}^{P} \sum_{v=1}^{b_u-1} \frac{R_{(i,k)}^{(u,v)} - R_{(i,j)}^{(u,v)} - R_{(i,k)}^{(u,b_u)} + R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}} C_{(u,v)}^{(m)} + \sum_{u=1}^{N} \frac{R_{(i,k)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}} C_u \right)
\end{aligned} \tag{3.13}$$

where $\Delta_{(i,j)} = \frac{1}{R_{(i,j)} \sum_{k=1}^{b_i} \frac{1}{R_{(i,k)}}}$. On the other hand, the block Gauss-Seidel method of (3.6), plus condition 2 of (3.9), implies

$$\begin{aligned}
C_{(i,j)}^{(m+1)} &= \sum_{k=1}^{b_i-1} d_{(i,j),(i,k)} \left(b_{(i,k)} - \sum_{u=1}^{i-1} \sum_{v=1}^{b_u-1} a_{(i,k),(u,v)} C_{(u,v)}^{(m+1)} - \sum_{u=i+1}^{P} \sum_{v=1}^{b_u-1} a_{(i,k),(u,v)} C_{(u,v)}^{(m)} \right) \\
&= \Delta_{(i,j)} \left(\sum_{k=1,k\neq j}^{b_i} \frac{1}{R_{(i,k)}} \right) \left(\sum_{u=1}^{N} \left(R_{(i,b_i)}^{(u,b_u)} - R_{(i,k)}^{(u,b_u)} \right) C_u - \sum_{u=1}^{i-1} \sum_{v=1}^{b_u-1} a_{(i,j),(u,v)} C_{(u,v)}^{(m+1)} - \right. \\
&\quad \left. \sum_{u=i+1}^{P} \sum_{v=1}^{b_u-1} a_{(i,j),(u,v)} C_{(u,v)}^{(m)} \right) + \sum_{l=1,l\neq j}^{b_i} -\frac{\Delta_{(i,j)}}{R_{(i,l)}} \left(\sum_{u=1}^{N} \left(R_{(i,b_i)}^{(u,b_u)} - R_{(i,l)}^{(u,b_u)} \right) C_u - \right. \\
&\quad \left. \sum_{u=1}^{i-1} \sum_{v=1}^{b_u-1} a_{(i,l),(u,v)} C_{(u,v)}^{(m+1)} - \sum_{u=i+1}^{P} \sum_{v=1}^{b_u-1} a_{(i,l),(u,v)} C_{(u,v)}^{(m)} \right)
\end{aligned} \tag{3.14}$$

where $a_{(i,k),(u,v)}$ is the $(i,k),(u,v)$-element of the matrix A in (3.5). The coefficients of some terms of (3.13) and (3.14) are reduced as follows:

- Term $C_{(i,v)}$ in (3.13), for $v = 1, \ldots, b_i,\ v \neq j$:

$$
\begin{aligned}
&\Delta_{(i,j)} \sum_{k=1}^{b_i} \frac{R^{(i,v)}_{(i,k)} - R^{(i,v)}_{(i,j)} - R^{(i,b_i)}_{(i,k)} + R^{(i,b_i)}_{(i,j)}}{R_{(i,k)}} \\
&= \Delta_{(i,j)} \left(\sum_{k=1}^{b_i} \frac{R^{(i,v)}_{(i,k)}}{R_{(i,k)}} - \sum_{k=1}^{b_i} \frac{R^{(i,b_i)}_{(i,k)}}{R_{(i,k)}} \right) \\
&= \Delta_{(i,j)} \left(\frac{R^{(i,v)}_{(i,v)}}{R_{(i,v)}} - \frac{R^{(i,b_i)}_{(i,b_i)}}{R_{(i,b_i)}} \right) \\
&= 0
\end{aligned}
$$

- Term $C_{(i,j)}$ in (3.13):

$$
\begin{aligned}
&1 + \Delta_{(i,j)} \sum_{k=1}^{b_i} \frac{R^{(i,j)}_{(i,k)} - R^{(i,j)}_{(i,j)} - R^{(i,b_i)}_{(i,k)} + R^{(i,b_i)}_{(i,j)}}{R_{(i,k)}} \\
&= 1 - \Delta_{(i,j)} \left(\sum_{k=1}^{b_i} \frac{R^{(i,j)}_{(i,j)}}{R_{(i,k)}} \right) + \Delta_{(i,j)} \left(\sum_{k=1}^{b_i} \frac{R^{(i,j)}_{(i,k)}}{R_{(i,k)}} - \sum_{k=1}^{b_i} \frac{R^{(i,b_i)}_{(i,k)}}{R_{(i,k)}} \right) \\
&= 1 - 1 + 0 \\
&= 0
\end{aligned}
$$

- Term C_u in (3.14), for $u = 1, \ldots, N$:

$$\Delta_{(i,j)} \sum_{k=1,k\neq j}^{b_i} \sum_{u=1}^{N} \frac{R_{(i,b_i)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}} - \Delta_{(i,j)} \sum_{l=1,l\neq j}^{b_i} \sum_{u=1}^{N} \frac{R_{(i,b_i)}^{(u,b_u)} - R_{(i,l)}^{(u,b_u)}}{R_{(i,l)}}$$

$$= \Delta_{(i,j)} \sum_{k=1,k\neq j}^{b_i} \sum_{u=1}^{N} \frac{R_{(i,b_i)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)} - R_{(i,b_i)}^{(u,b_u)} + R_{(i,k)}^{(u,b_u)}}{R_{(i,k)}}$$

$$= \Delta_{(i,j)} \sum_{k=1,k\neq j}^{b_i} \sum_{u=1}^{N} \frac{R_{(i,k)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}}$$

$$= \Delta_{(i,j)} \sum_{k=1}^{b_i} \sum_{u=1}^{N} \frac{R_{(i,k)}^{(u,b_u)} - R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}}$$

- Term $C_{(u,v)}$ in (3.14), for $v = 1, \ldots, b_u - 1,\ u = 1, \ldots, P$, and $u \neq i$:

$$\Delta_{(i,j)} \sum_{k=1,k\neq j}^{b_i} -\frac{a_{(i,j),(u,v)}}{R_{(i,k)}} - \Delta_{(i,j)} \sum_{l=1,l\neq j}^{b_i} -\frac{a_{(i,l),(u,v)}}{R_{(i,l)}}$$

$$= \Delta_{(i,j)} \sum_{k=1,k\neq j}^{b_i} \frac{a_{(i,k),(u,v)} - a_{(i,j),(u,v)}}{R_{(i,k)}}$$

$$= \Delta_{(i,j)} \sum_{k=1}^{b_i} \frac{R_{(i,k)}^{(u,v)} - R_{(i,b_i)}^{(u,v)} - R_{(i,k)}^{(u,b_u)} + R_{(i,b_i)}^{(u,b_u)} - R_{(i,j)}^{(u,v)} + R_{(i,b_i)}^{(u,v)} + R_{(i,j)}^{(u,b_u)} - R_{(i,b_i)}^{(u,b_u)}}{R_{(i,k)}}$$

$$= \Delta_{(i,j)} \sum_{k=1}^{b_i} \frac{R_{(i,k)}^{(u,v)} - R_{(i,j)}^{(u,v)} - R_{(i,k)}^{(u,b_u)} + R_{(i,j)}^{(u,b_u)}}{R_{(i,k)}}$$

All the corresponding coefficients of (3.13) and (3.14) are equal, so the two methods are equivalent.

Chapter 4

Timing Simulation of Transistor Networks

In section 2.1, a timing model for MOS transistor networks was presented. RC networks are used to approximate transistor networks for estimating delays between logic events. Two algorithms were presented in Chapter 2 and Chapter 3 for calculating the delays of RC tree networks and general RC networks, respectively. In this chapter, these algorithms are applied to perform timing simulation of transistor networks.

The concept of dominant path has been successfully used in logic simulation of digital MOS circuits [5,6]. This concept also leads to a natural tree decomposition of RC networks, as explained in section 5.1. In most cases, the delays estimated from individual tree subnetworks are already very accurate. For other cases, the technique of load redistribution can be used to calculate the exact delay of every node. In section 4.2, logic simulation and delay estimation are combined into one unified process. The mechanism for scheduling logic events is similar to that presented by Terman [46]. As each new event is invoked, the delay values as well as the logic states of the nodes in the network are updated incrementally. Simulation results are discussed in section 4.3.

4.1 Dominant Path Decomposition

Consider a transistor network driven by both VDD and GND. There are a certain number of paths that lead a node in this network to either one of the sources. Among all these paths, the ones with the smallest series resistance (the source resistance) are called the dominant paths of the node (in general, there is more than one such path). In most practical cases, all the dominant paths of a node lead to the same source, and the logic state of the node is set to that of the source, 1 for VDD and 0 for GND. Similar to the formulation of (3.1), the source resistances of the nodes and the series resistances of the paths are related as

$$\begin{cases} R_i^j = R_j + r_{i,j} & j = 1, \ldots, a_i,\ i = 1, \ldots, N \\ R_i = \min_{j=1,\ldots,a_i} R_i^j & i = 1, \ldots, N \end{cases} \tag{4.1}$$

where R_i is the source resistance of node N_i, $r_{i,j}$ is the resistance between node N_i and its jth neighbor, and R_i^j is the series resistance of the jth path of node N_i. The determination of the R_i and R_i^j values are equivalent to the single-source shortest path problem with positive costs. There exist quite a few algorithms for solving this problem, for example, Dijkstra's algorithm [2]. The solution of (4.1) exists and is unique, and the time complexity of the algorithm is $O(n^2)$, where n is the number of nodes in the network. Equation (4.1) needs to be solved only once and, as the transistor network evolves, the dominant paths and source resistances of the nodes and the series resistances of the paths can be updated incrementally.

The dominant paths of the nodes in a network lead to a very efficient scheme for estimating delays. Recall, in section 2.1, that any (connected) RC network is driven by one and only one source. If a network N is driven by both VDD and GND, then this network is decomposed into two (or more) groups: N_{VDD} and N_{GND} which consist of the nodes of N whose dominant paths lead to VDD and GND, respectively. A resistor between two nodes in the same group (N_{VDD} or N_{GND}) is considered to be a resistor of the group. A resistor between one node in N_{VDD} and another node in N_{GND} is considered as if it were not present. Such resistors are called the "bridges" between N_{VDD} and N_{GND}. The above scheme can be generalized to decompose an RC network even if the nodes are driven by the same source. That is to approximate the load distribution such that a node loads only those nodes that are along its dominant paths, and has no effect on the nodes along other paths, whether they lead to the same source or not. This scheme, referred to as the

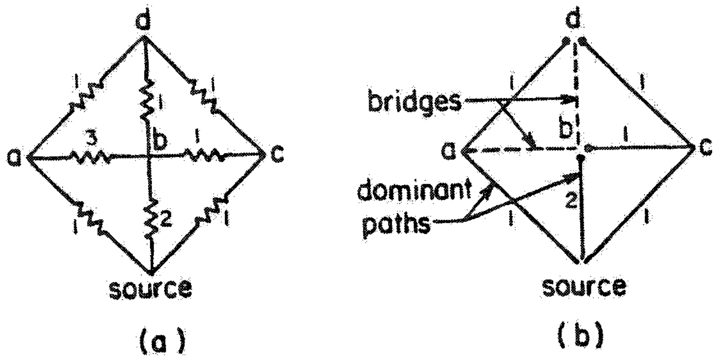


Figure 4.1. Illustration of the DP (dominant path) decomposition scheme

dominant path (DP) scheme, decomposes a network into a collection of trees since a node A cannot be in the dominant path of another node B if node B is in the dominant path of A. If a node has more than one dominant path, then the load is equally distributed among these paths.

Example 4.1. Consider the network of Figure 4.1.a. which consists of four nodes: a, b, c and d. There is only one source driving the network. The dominant paths of these nodes decompose the network into three trees, which are indicated in Figure 4.1.b. Branches $b-d$ and $a-b$ do not belong to any of these trees, and are called the bridges among these trees. The delays of various nodes are calculated independently for each tree. There are two dominant paths incident on node b, so the node capacitance is split into two equal parts. The delay of this node is approximated by the average of the delays evaluated from the two dominant paths. The same goes for node d. If the difference between the delays of every pair of nodes that are connected by a bridge is within certain error bound, then no relaxation process is necessary. Otherwise, every bridge needs to be bound to a tree to start the relaxation process. Note that the load redistribution technique applies only to a collection of trees that are driven by the same source. Two trees that are driven by different sources are not allowed to be connected by bridges. ∎

As an aid to detect conflict conditions in a network, the source resistance of a node to

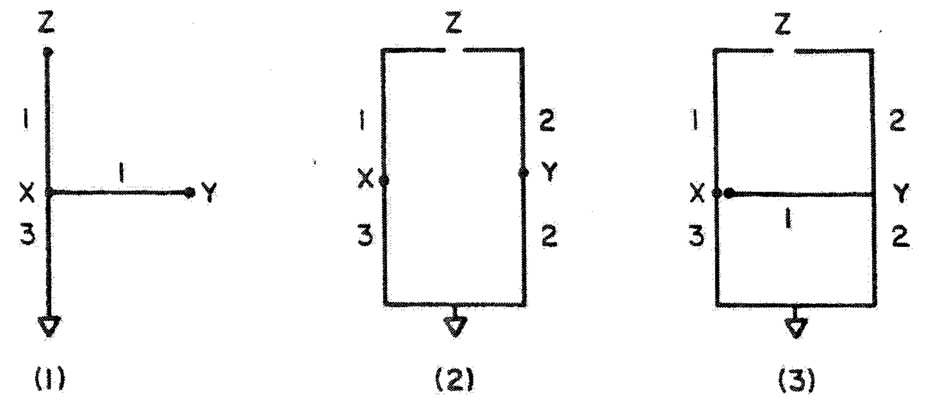


Figure 4.2. The DP decomposition for Example 3.9

VDD (the smallest resistance among the paths that lead to VDD) is compared with that to GND. A conflict condition is detected when the difference between the two resistances is smaller than a threshold value.

Example 4.2. The nMOS circuit presented in Example 3.9 is considered again. The dominant paths of the nodes in each of the three cases are shown in Figure 4.2. The delays evaluated by the DP scheme (T') and the exact values (T) are listed in Table 4.2.

case	1	2			3		
	T_x, T_y, T_z	T_x	T_y	T_z	T_x	T_y	T_z
T'	–	9	6	10.5	9	7.5	11.25
T	–	8.625	6.25	10.5	7.69	6.87	10.09
$\frac{T'-T}{T}$	0%	4.2%	−4.2%	0%	−2.5%	31%	11%

Table 4.1. Error Percentage of the delays calculated by the DP scheme

In case 1, the network is a tree, so the delays evaluated by the DP scheme are all exact. In case 2, both paths incident on node Z are dominant, and the delay calculated is also exact. Independent of the size of the network, the relative errors of case 3 and case 2 are typical for nontree networks with and without bridge connections, respectively. Such error percentages are quite acceptable for most practical applications. ▮

To a node, there are two kinds of nondominant paths: one kind leads to a source (nondominant driving paths), and the other kind leads to an open end (loading paths). A pair of nondominant paths may also result in an phantom path, i.e. a loop with no side branch that connects to a source. Such paths are no different from those which lead to an open end. Note that all dominant paths must lead to the source. The presence of nondominant driving paths causes an error in evaluating delays, while that of loading paths does not. On the other hand, the following two conditions are sufficient for the exact calculation of the delay of a node N using the DP scheme:

- All the driving paths of node N are dominant.
- For those nodes that are along a driving path of N, all their driving paths but the one that passes through N are dominant.

Another factor that affects the accuracy of the DP scheme is the relative magnitude of the node capacitances. Note that only the values of resistances participate in the determination of dominant paths, not the values of capacitances. Roughly speaking, the more uniformly the capacitances are distributed in the network, the more accurate the DP scheme is. The following example illustrates the effect of the capacitance ratio on the accuracy of the DP scheme.

Example 4.3. Consider the simple circuit of Figure 4.3.a which is driven by a source from both ends. The exact delay of node 1 of the circuit is $T = \frac{r_1(r+r_2)}{r_1+(r+r_2)}c_1 + \frac{r_1 r_2}{r_1+(r+r_2)}c_2$. On the other hand, the delay T' estimated by the DP scheme depends on the relative magnitude of the three resistors: r, r_1, and r_2. There are five possible cases, of which the values of T' and the relative errors E_r are listed in Table 4.2. The term E_r is $\frac{T'-T}{T}$ if $T' > T$, and $\frac{T'-T}{T'}$ otherwise. Shown in Figure 4.3.b is E_r plotted as a function of $\log r_\rho$, with c_ρ as a parameter, where $r_\rho = \frac{r_1}{r_2}$, and $c_\rho = \frac{c_1}{c_2}$. The following remarks refer to Table 4.2 and Figure 4.3.b.

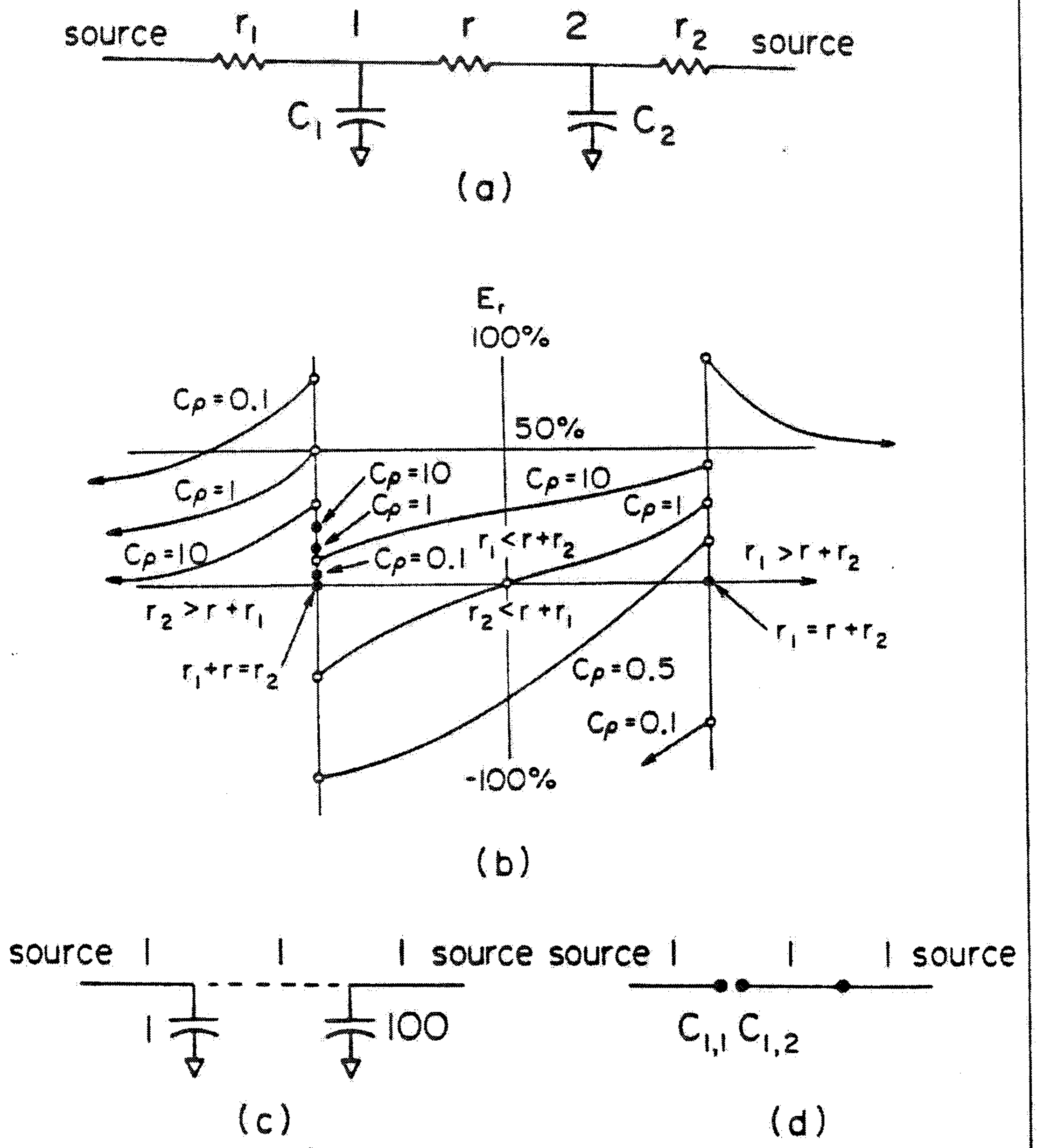


Figure 4.3. Effect of capacitance ratios on the accuracy of the DP scheme

		T'	$\frac{T'-T}{T}$
1	$r_1 + r < r_2$	$r_1(c_1 + c_2)$	$\frac{r_1c_1+(r+r_1)c_2}{(r+r_2)c_1+r_2c_2}$
2	$r_2 = r_1 + r$	$r_1(c_1 + \frac{c_2}{2})$	$\frac{r_1c_1}{(r+r_2)c_1+r_2c_2}$
3	$r_1 + r > r_2, r_2 + r > r_1$	r_1c_1	$\frac{r_1c_1-r_2c_2}{c_1(r_1+r+r_2)}$
4	$r_1 = r + r_2$	$\frac{r_1c_1+r_2c_2}{2}$	0
5	$r_1 > r + r_2$	$(r + r_2)c_1 + r_2c_2$	$\frac{r+r_2}{r_1}$

Table 4.2. T' and E_r for different ratios of r_1, r_2, and r

- If $r_2 > r_1 + r$, then the approximation is made as if the resistor r_2 were not present. As a result, T' is larger than T. E_r increases as r_p gets larger, or c_p gets smaller.
- If $r_1 > r + r_2$, then the approximation is made as if r_1 were not present. In this case, E_r is independent of the value of c_p. With r_1 and r_2 interchanged, this error is always larger than that in the previous case. As $c_p \to 0$, the two errors become identical, and the two curves become symmetric.
- If $r_1 < r + r_2$, and $r_2 < r + r_1$, then the approximation is made as if the resistor r did not exist. Node 1 is fully loaded (overloaded) by c_1, and is not loaded at all (underloaded) by c_2. Although these two errors are of opposite sign, they do not always cancel out exactly. As the c_p varies, the error E_r changes tremendously. As two extremes, $E_r \to 50\%$ as $c_p \to \infty$, and $E_r \to -\infty$ as $c_p \to 0$. Fortunately, in a practical circuit, such highly asymmetric situations do not occur very often.
- If $r_1 = r + r_2$, then $E_r = 0$; if $r_2 = r + r_1$, then $E_r = \frac{c_p}{5c_p+3}$. Compared with the errors of the other cases, these two values are considerably smaller. Moreover, a small perturbation of the values of r's at these points will result in a drastic increase of the error. In fact, the smaller the perturbation, the larger the increase of the error. This phenomenon, due to the discontinuous (with respect to r) approximation of load distribution, is common to various networks. One possible modification of the DP scheme is to consider paths with slightly larger resistance to be dominant as well.
- Take $r_1 = r = r_2 = 1$, $c_1 = 1$, and $c_2 = 100$ to show how algorithm LRD corrects the

errors caused by the DP scheme. The exact delays of nodes 1 and 2 are: $T_1 = 34$ and $T_2 = 67$, respectively. The dominant paths of these two nodes are shown in Figure 4.3.c. and, under the DP scheme, their delays are $T'_1 = 1$ and $T'_2 = 100$, respectively. To start the relaxation process, bridge $1-2$ is linked to either node 1 or node 2. If it is linked to node 2, then the network is decomposed as indicated in Figure 4.3.d and node 1 is split into two parts. At the first relaxation step, $T_1^{(1)} = \frac{\frac{T'_1}{r_1} + \frac{T'_2}{(r+r_2)}}{\frac{1}{r_1} + \frac{1}{r+r_2}} = 34$, $\Delta_{C_{1,2}} = \frac{T_1^{(1)} - T'_2}{r + r_2} = -33$, and $T_2^{(1)} = T'_2 + r_2 \cdot \Delta_{C_{1,2}} = 67$. One relaxation step is enough to correct the values of the delays. ▮

4.2 Simulation Algorithms

The evolution of a transistor network is represented by a sequence of logic events, sorted by time. Each event corresponds to a change of the logic state of a node, from 1 to 0, or vice versa. At the time when an event is scheduled, say to change node A from L to $\overline{L}$, the logic state of node A is set to X first, indicating that it is in transition. Not until the event is activated after a certain delay does the state of A switch to $\overline{L}$. The activation of this event will affect other nodes in the network through the transistors whose gates are controlled by node A. Consider the circuit of Figure 4.4.a. Initially, transistor M is turned off, and networks $N1$ and $N2$ are independent of each other. Let R_D and R_S denote the source resistances of the two lateral nodes D and S (drain and source) of transistor M, respectively. At time T_c, an event is invoked to switch the gate node of M and turn the transistor on. As a result, the series resistance of path $P1$ of node D changes from ∞ to $R_S + R_M$, where R_M is the ON-resistance of M. Likewise, the series resistance of path $P2$ becomes $R_D + R_M$. One and only one of the following five conditions is satisfied.

1. $R_S + R_M > R_D$ and $R_D + R_M > R_S$: $P1$ and $P2$ are both nondominant paths of node D and S, respectively. Under the DP scheme, neither the logic state nor the

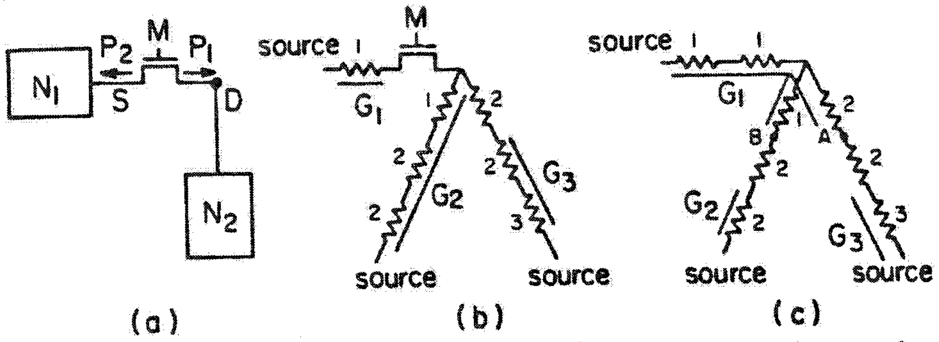


Figure 4.4. Network configuration before and after transistor M is turned on

delay value of any node of the network is changed.

2. $R_S + R_M < R_D$: $P1$ becomes the dominant path of node D. Take the network of Figure 4.4.b as an example. The dominant paths of the nodes before and after transistor M switches are indicated in Figure 4.4.b and Figure 4.4.c, respectively. With M turned off, node A belongs to tree G_1, so its logic state equals, or is scheduled to be, the state of the source driving G_1. As M is turned on, A becomes a part of tree G_3, and its logic state will be driven towards the source of G_3 which may be the same or different from that of G_1. The above argument also applies to node B. Trees G_1, G_2 and G_3 are perturbed by transistor M's turning on. The logic states of the other nodes in these perturbed trees do not change; however, their delay values may be affected. The most primitive approach to updating the delays is to evaluate the stored charge of each node of these trees and use algorithm TREE to recalculate the delays. If there is no event scheduled for a node, then the stored charge of the node equals 0 or the node capacitance, depending on whether its logic state is the same as or opposite to that of the source driving it. The simplest way to estimate the stored charge of a node when an event is being scheduled is by way of linear interpolation: $Q = Q_0 + \frac{T_1+T_0-T_c}{T_1}(C - Q_0)$, where Q_0 and T_1 are respectively the charge and delay calculated at time T_0 when the event is scheduled. More accurate estimation by using an exponential function is also possible, but unwarranted considering the level

of approximation being used.

The other three cases ($R_D + R_M < R_S$, $R_S + R_M = R_D$, and $R_D + R_M = R_S$) can be dealt with in a similar way.

The following pseudo-code describes a simulation cycle:

```
procedure sim_cycle;
begin
    put input events into the event queue;
    while (event queue not empty) do begin
        1. fetch an event from the top of the event queue;
        2. update the node state corresponding to the event.
        3. propagate the effect of this update,
              and put all affected nodes in the event queue.
    end;
end;
```

It was pointed out, in section 2.2, that the definition of delay (2.1) is equal to the Elmore's delay in the case of zero initial charge. In what follows, the consistency between these two definitions is discussed for the cases of nonzero initial charge. This discussion also suggests another simulation algorithm that is very efficient, and gives the exact delay value for the end node of a chain of transistors. Consider the chain of transistors in the tree network shown in Figure 4.5. Initially, all the transistors in the chain are turned off, and all the transistors in the side branches are turned on. All internal nodes are without initial charge. Compare the following two cases:

1. All transistors in the chain are turned on at the same time. This is a case in which Elmore's definition can be applied.
2. The transistors in the chain are turned on one after another, starting from M_1, then $M_2, \ldots, M_n$, successively. M_i is not turned on until the nodes $N_{1,\ldots,i-1}$ all settle

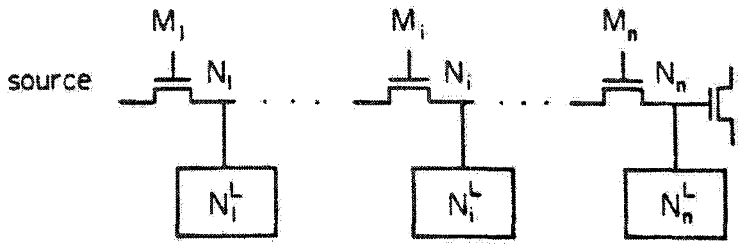


Figure 4.5. A chain of transistors in an RC tree network

down. This is a case where Elmore's definition cannot be applied.

In case 1, T_D, the delay for the end node N_n, equals $\sum_{i=1}^{n}(r_i \sum_{k=i}^{n} C_k^L)$, where r_i is the ON-resistance of transistor T_i, and C_k^L is the total load capacitance of node N_k, including the loads from side branches. In case 2, there are n time intervals to be considered. The ith time interval starts when transistor M_i is turned on, and ends when the nodes $N_{1,\ldots,i}$ all settle down. T_i, the length of the ith time interval, is equal to $(\sum_{k=1}^{i} r_k)C_i^L$. Note that the stored charge in the nodes $N_{1,\ldots,i-1}$ has been taken into consideration. It is easy to check that $\sum_{i=1}^{n} T_i$ is equal to the T_D in case 1. ∎

In the case of nonzero initial conditions, T_D is still consistent with the Elmore's delay, as the above example indicates. In fact, $\sum_{i=1}^{n} T_i = T_D$ even if transistor M_i is turned on before the nodes $N_{1,\ldots,i}$ settle down. As theorem 2.2′ and theorem 2.5′ indicate, the value of delay depends only upon the amount of charge yet to be supplied for each node. Regardless of how the charge is actually supplied, the overall delay should always be the same. Another thing to be noted is that, when the network topology changes, a nonzero delay may be associated with a node which has been settled previously. This delay corresponds to the settling of the glitches produced by the dynamic charge sharing effect. Recall that T_D is equal to the area between 1 and the response curve. The larger value of T_D always implies a bigger glitch. In practical circuits, small glitches do not tend to produce transitions at the next stages. We set a threshold value and ignore all glitches that are smaller than this value. This filtering action prevents circuit events from over-propagation, and makes our algorithm more efficient. On the other hand, the occurrence of a sizable glitch is very useful information for the designer, and our algorithm is capable of detecting these glitches without any extra cost.

The result of the above example suggests a modification of the DP simulation scheme. That is to consider a node not directly driven by the source, but by one of its neighboring nodes through a dominant path. Take case 1 of the above example. Instead of driving all nodes $N_{1,\ldots,n}$ at the same time, the source only drives N_1 because only N_1 is adjacent to the source. After delay time $T_1 = r_1 c_1$, node N_1 will change its logic state, and be able to drive next node N_2, which takes time $T_2 = (r_1 + r_2)c_2 = R_2 c_2$, where $R_2 = r_1 + r_2$ is the source resistance of N_2. Any other nodes that are also adjacent to N_1 are driven by it, as well. However, each node is driven independently (an additional approximation), and the delay is not affected by the presence of other nodes. Note that c_i is the node capacitance, as opposed to C_i^L which is the total load capacitance of node i. In general, node N_i is driven by node N_{i-1}, which takes time $T_i = R_i c_i = (r_1 + \ldots + r_i)c_i$.

The advantages of this modified scheme over the original one are as follows:

- Among the three parameters R, C and D for evaluating delay values, only R (source resistance) needs to be calculated. The determination of the other two parameters does not require any computation at all. Under this modified scheme, the total load capacitance is replaced by the node capacitance for parameter C. On the other hand, parameter D is implicit in the overall delay before the activation of the event corresponding to its driving node. Note that the values of resistances are also essential for the determination of logic levels, and this modified scheme is almost as efficient as a pure logic simulator.
- Under the original scheme, all nodes $N_{1,\ldots,n}$ are inserted into the event queue at the same time. Under the modified scheme, however, a node is not inserted until the event

corresponding to its driving node is evoked and removed. The average length of the queue is in general much shorter for the modified scheme.

The disadvantages of the modified scheme are as follows:

- This modified scheme only calculates correct delay for a node that is at the open end of a chain of transistors. An error occurs if some intermediate node is also of concern, or the network is more complicated than just a chain.

In summary, the modified scheme is more efficient than the original one; however, it is less accurate. Depending on individual applications, this simulator can be easily tuned to fulfill the requirement.

Based on algorithms TREE and LRD, an experimental simulator called SDS (Signal Delay Simulator) has been developed. Some simulation results are presented in the next section.

4.3 Simulation Results

Before any simulation can be done, the effective sheet resistances of various transistor types must be determined first. Note that, in this section, the term "effective resistance" is used interchangeably with the term "ON-resistance." The high-field effect of MOS transistors is ignored in our model, and the effective resistance of a transistor is inversely proportional to its W/L ratio. The capacitance values can be obtained directly from the fabrication data. To calibrate the effective resistances of depletion and enhancement transistors in nMOS circuits, the inverter chain in Figure 4.6.a is considered. The W/L ratios of the pullup and pulldown transistors are 8/2 and 2/2 (unit: $\lambda=2\mu m$), respectively. The SPICE simulation result is shown in Figure 4.6.b. The time taken for two consecutive inverters to switch, one up and one down, is $\frac{2.7}{3}$ =0.9ns. The time taken for an up transition is about 4 times longer than that for a down transition. The capacitance of the output node is estimated to be 0.015pf. As a result, the effective sheet resistances of enhancement and depletion transistors are the same, both equal to $\frac{0.9}{5}/0.015 = 12\text{Kohm}/_{\square}$. The same analysis has been performed on multi-input nand/nor gates. The variations among the resistance values calibrated by different gate configurations are only minor, as predicted from our theory (Theorem 2.2). The SDS simulation result on the inverter chain is indicated in Figure 4.6.c.

The schematic diagram of MOS circuits being tested and their input waveforms are shown in Figure 4.7.a–4.11.a. Comparisons of output waveforms generated by SPICE and SDS are shown in Figure 4.7.b-4.11.b. A comparison of the analysis time in CPU seconds spent by each program is given in Table 4.3. Both programs run on a DEC-2060 computer. The tabulated figures do not include the time spent in the read-in, setup, and read-out phases of each program.

Circuit of	Fig. 4.7	Fig. 4.8	Fig. 4.9	Fig. 4.10	Fig. 4.11
# of unknown nodes	4	8	9	11	48
# of transistors	9	13	6	22	104
# of R's and C's	0	0	12	0	0
CPU-SPICE (sec)	10.3	12.3	12.9	40.3	198
CPU-SDS (sec)	0.07	0.05	0.09	0.12	0.31
CPU-SPICE/CPU-SDS	147	246	143	336	639

Table 4.3. Comparisons of analysis time between SPICE and SDS

Remarks:

- Figure 4.7.a shows an nMOS XOR circuit. Note that when input A goes high and input B goes low, both the gate and source voltages of transistor M and transistor N are changing, an effect that is not considered in our model. As a result, the output delay estimated by SDS (2.0ns) is shorter than that predicted by SPICE (2.2ns); the error is about 10%. Note that the value 2.2ns is obtained by cascading an inverter chain to the output node, and using the same technique as we did for the inverter chain (Figure 4.6.b).

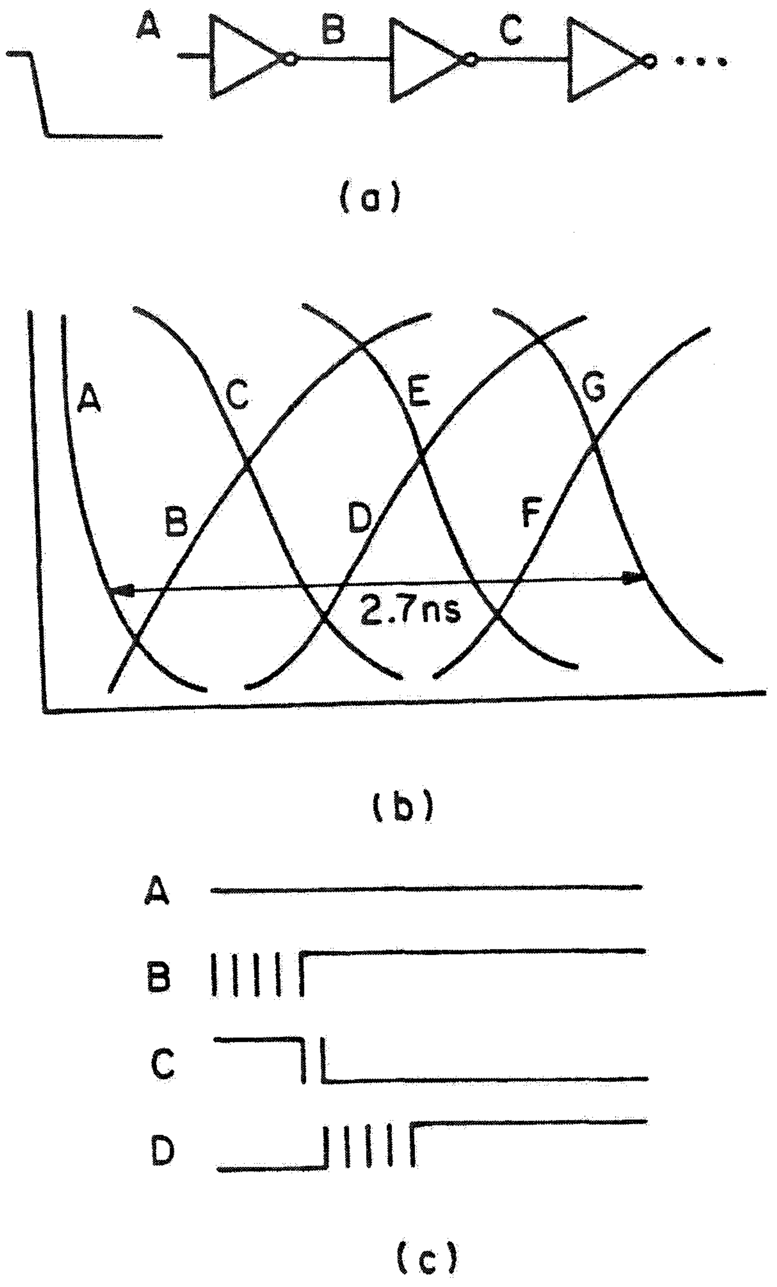
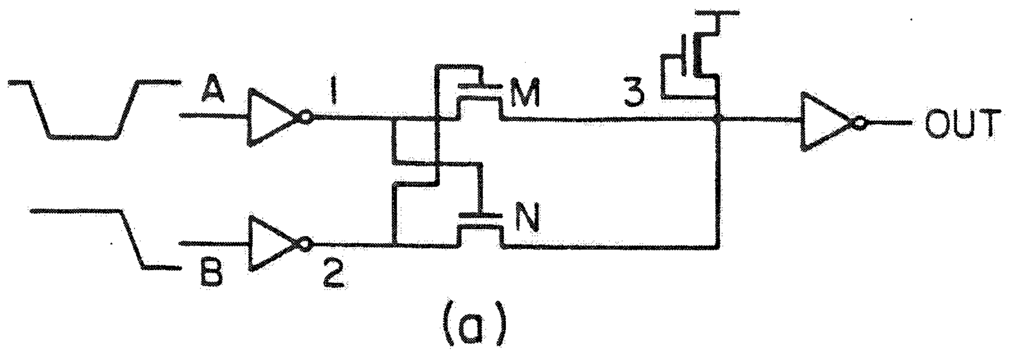


Figure 4.6. Calibration of effective resistances by SPICE



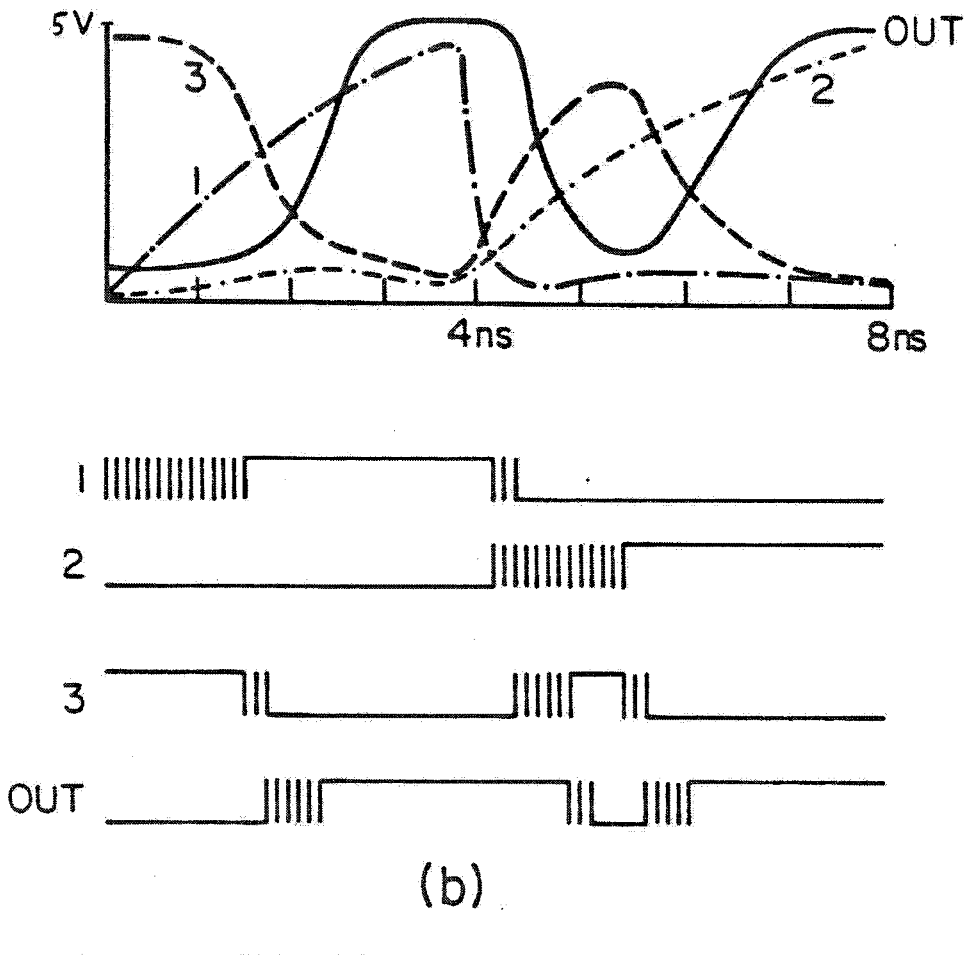


Figure 4.7. SDS and SPICE simulations of an nMOS XOR gate

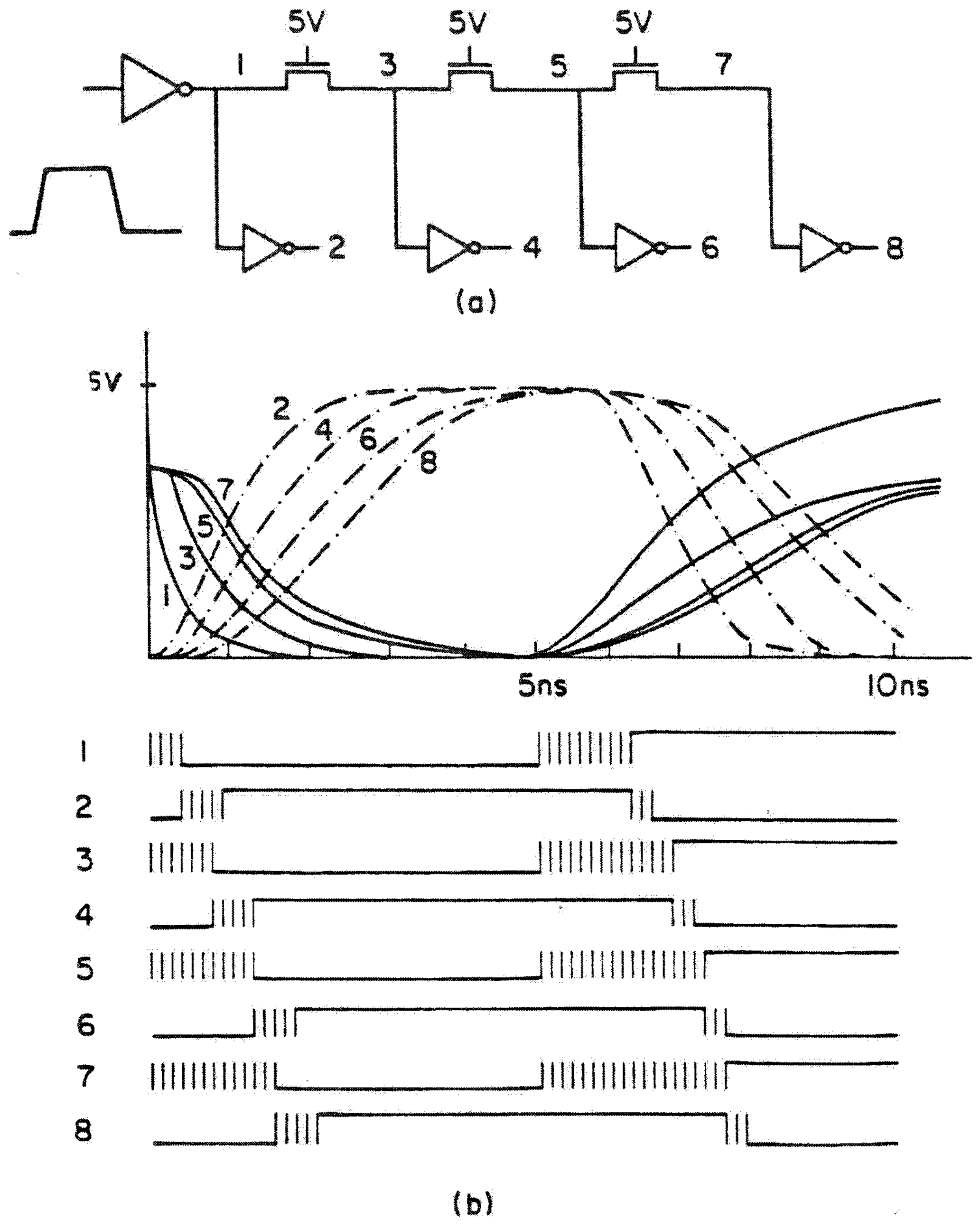
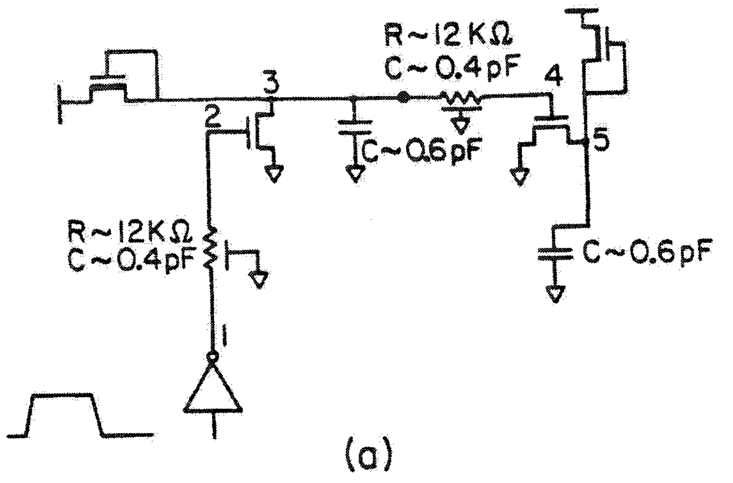


Figure 4.8. SDS and SPICE simulation of an nMOS carry chain



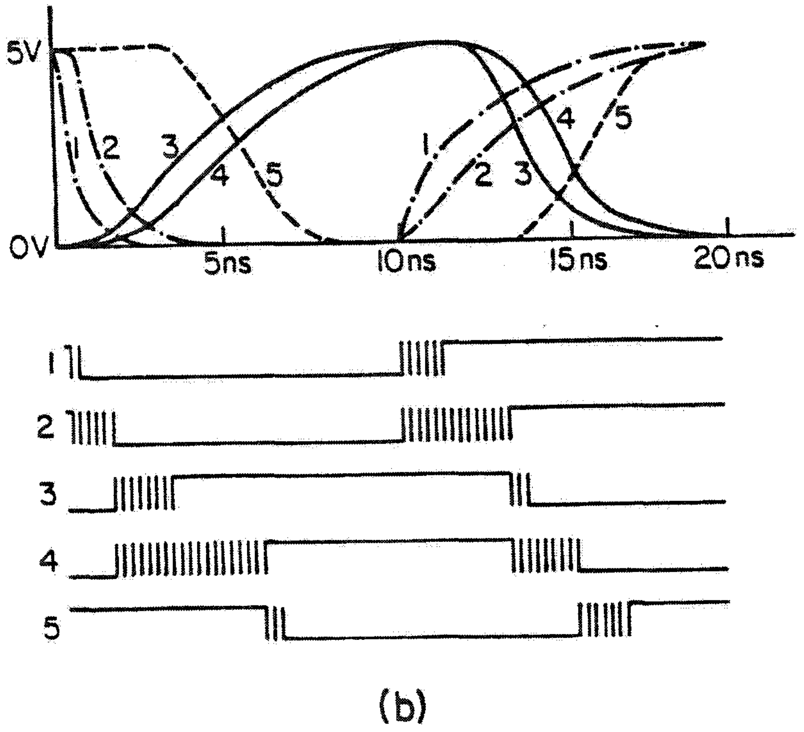
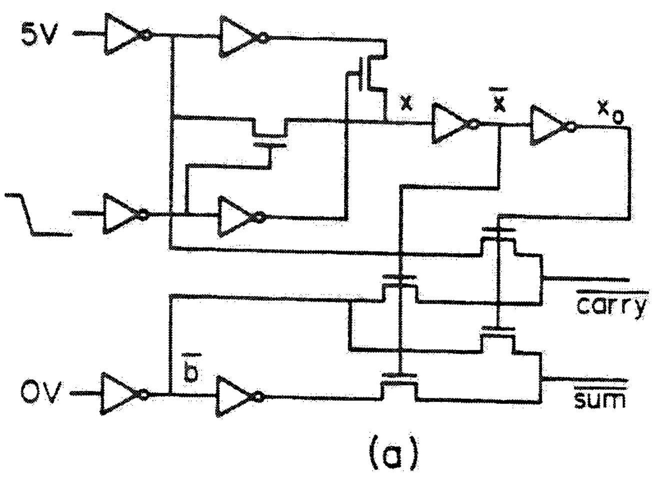


Figure 4.9. SDS and SPICE simulation of an nMOS PLA



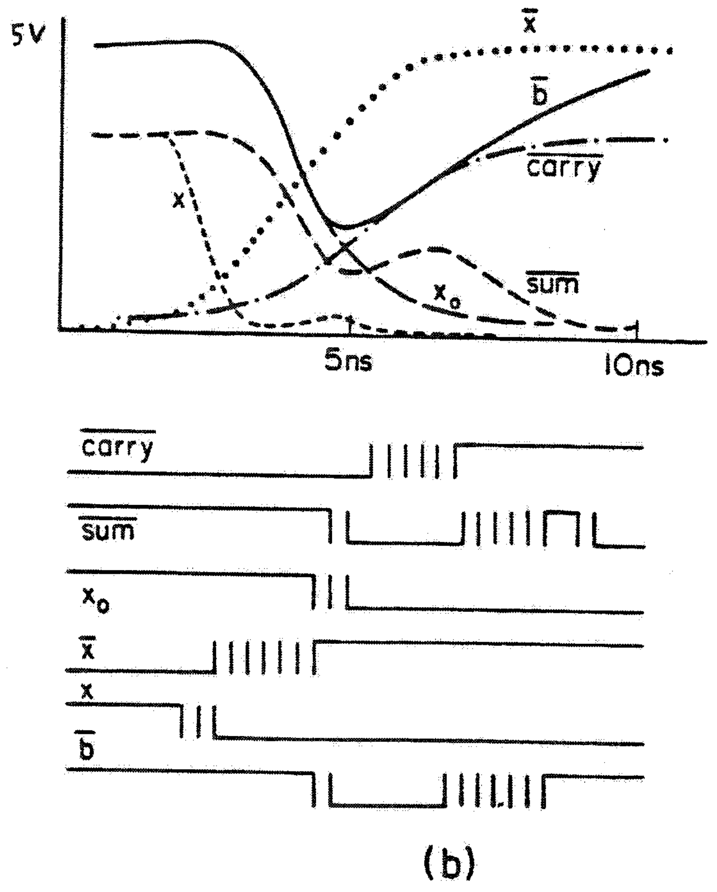
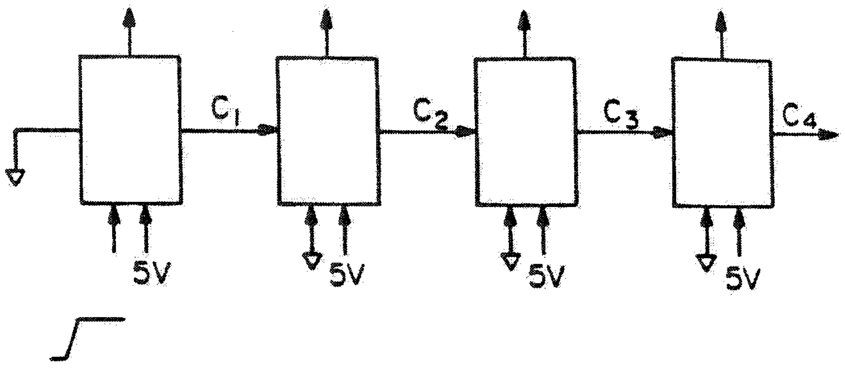
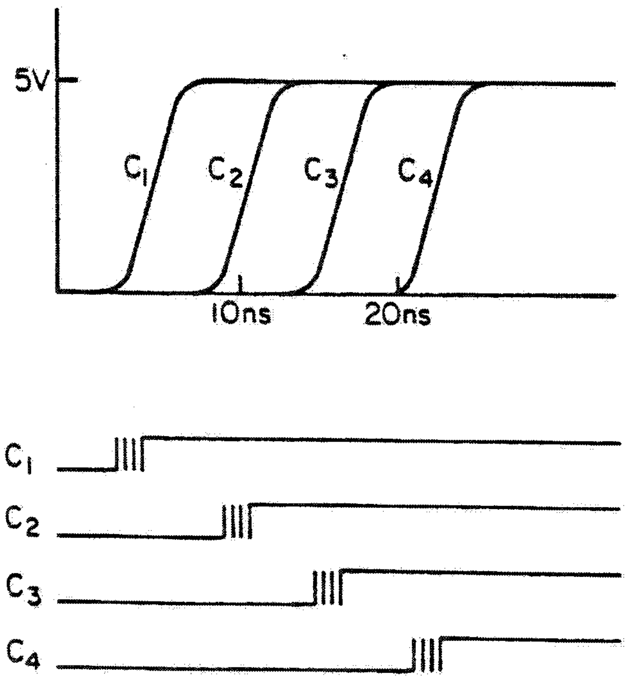


Figure 4.10. SDS and SPICE simulation of an nMOS one-bit adder



(a)



(b)

Figure 4.11. SDS and SPICE simulation of an nMOS four-bit adder

- Figure 4.8.a shows an nMOS carry-chain circuit. From the 5ns–10ns section of the SPICE output (Figure 4.8.b), high-going signals degrade significantly when they pass through the carry chain. The present implementation of SDS does not include the effect of slow inputs; thus the estimated delay between node 1 and node 2 is the same as that between node 7 and node 8. Furthermore, the body effect is only dealt with by doubling the effective resistances of those transistors that are gated by a degraded signal. The output delays estimated by SPICE and SDS are listed and compared in Table 4.4. Note that the $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions indicated in the table refer to signals along the carry chain. The direction is reversed for the output signals. While it predicts the correct qualitative behavior, SDS underestimates the absolute delays of almost all the output nodes because the slope of the signals that drive the output transistors is smaller than that in the inverter chain (the calibration reference). The main problem is that the MOS pass transistor is not well approximated as a linear resistor, since its conductance is voltage-dependent. To improve the accuracy, different effective resistances may be associated with transistors of different usages (such as pass transistors) [23] or different types (rising or falling) of transitions [20,21]. Note that a static pre-analysis is required to determine the usage of transistors. For the present implementation of SDS, no static pre-analysis is performed, and the effective resistance was established by a single calibration, and is only a function of the type of the transistor (enhancement or depletion). Simple as it is, SDS works reasonably well for circuits without long carry chains.

	$1 \rightarrow 0$ transitions					$0 \rightarrow 1$ transitions			
Node	2	4	6	8		2	4	6	8
SPICE	0.83	1.17	1.87	2.15		1.95	2.72	3.80	4.30
SDS	0.84	1.15	1.70	1.85		1.65	2.10	2.40	2.55
error	2.3%	-1.9%	-9.1%	-14%		-15%	-22%	-36%	-41%

Table 4.4. Delay estimation of the output nodes of Figure 4.8.a

- Figure 4.9.a represents the critical path of an nMOS PLA which contains 60 minterms. The metal wires in the PLA are approximated by pure capacitances, and the polysilicon wires are approximated by six-step π ladder network. The estimated stray capacitances and resistances of these wires are indicated in Figure 4.9.a. Both the AND plane and the OR plane are driven by a strong buffer. The output delay estimated by SDS (5.2ns) is about 15% shorter than that predicted by SPICE (6.1ns).
- Figure 4.10.a shows an nMOS one-bit adder. To exaggerate the effect of feedback and multiplexing to test the capabilities of SDS, the W/L ratio of the pass transistors is deliberately changed to make their resistances unreasonably small. The resistance ratio among pass-transistors, pulldown transistors and pullup transistors is 1:30:120. Compared with the SPICE simulation result of Figure 4.10.b, SDS detects all glitches and transitions at approximately the correct time. Note that, although the length of the glitch of $\overline{\text{SUM}}$ is not estimated very accurately, the time when it settles is.
- From Table 4.4, SDS runs two to three orders of magnitude faster than SPICE for circuits consisting of fewer than one hundred transistors. Note that, in SDS, delays are calculated independently for different transistor groups. The simulation time grows linearly with the number of logic events, and does not depend directly on the size of the circuit. The CPU-SPICE to CPU-SDS ratio grows drastically as the size of the network increases.
- Recently, a new circuit simulation technique called "waveform relaxation" (WR) has been reported in the literature [26]. This technique is claimed to have nice numerical properties, and can speed up circuit analysis by at least an order of magnitude over SPICE. One possible application of SDS is to provide initial waveforms for WR-based circuit simulators. Note that a good initial guess of waveforms is crucial to the performance of this type of circuit simulator.

4.4 Summary

Delay estimation and logic simulation are combined into one unified process through the concept of dominant path. The usage of (2.1′) as the definition of delay is also justified for the case of nonzero initial charge. An experimental simulator, called SDS, has been developed and applied to a number of test circuits. Comparison of the simulation results with SPICE indicates that SDS is capable of analyzing the timing of digital MOS circuits with reasonable accuracy, and much faster simulation speed. The circuit information required for running SDS can either be extracted from masks [3] or be generated from layout synthesis systems [22,53].

Like SPICE and other timing and circuit simulators, SDS deals only with designs at the transistor level. To run SDS, the entire design must be flattened into transistors and nodes. However, due to the composition capabilities of our model, simulation can be done in a hierarchical manner. The generalization of the R, C, D parameters of two-port RC networks to functional blocks, and the application of our timing model to hierarchical timing simulation are discussed in Chapter 5. A timing simulator based on these principles is described in Chapter 6.

Chapter 5

Hierarchical Timing Simulation

It was shown in section 2.3 that the timing behavior of a two-port RC network can be characterized by three parameters: $\overline{R}$: series resistance, $\overline{C}$: effective capacitance, and $\overline{D}$: internal delay. These three parameters can be calculated in a hierarchical manner as the corresponding networks are composed in various ways. In this chapter, this characterization is generalized to semantic cells at any level of representation. Note that the characterization of a "semantic cell" is such that the cell can be abstracted by its steady-state behavior to interface with other cells. For any given timing discipline, there exists a way to partition a system such that every subsystem is a semantic cell. In this chapter, two-phase synchronous systems are used as an example to illustrate the principles of our hierarchical timing model. These principles apply equally well to other timing disciplines [10,43], and can be extended as necessary.

Two kinds of cells are distinguished in our hierarchical timing model: leaf cells and composition cells. Leaf cells are the primitive components that have no sub-components. A composition cell is a legal composition of leaf cells and other composition cells. With each leaf cell is associated a logic and timing description, which is valid for all possible input patterns and driving and loading conditions of the cell (as long as the composition preserves the semantics of individual component cells). To obtain such a description, SDS or other circuit or timing simulators can be used.

With each composition cell are associated a number of subcells which may be either leaf cells or composition cells, and a set of nets indicating how these subcells are connected. There is no explicit logic and timing description for a composition cell; however, it is possible to derive such a description from the descriptions of its subcells either analytically or by simulation. Once the logic and timing description of a composition cell is obtained, the composition cell is reduced to a leaf cell, and the details of its implementation can be totally eliminated.

In section 5.1, semantic cells of two-phase synchronous systems are characterized. Composition of semantic cells is discussed in section 5.2. The TREE algorithm developed in section 2.4 is extended for this purpose. Based on a one-input, one-output model, the timing of a static nMOS PLA is analyzed in section 5.3. Using the results of section 5.2, the timing behavior of the PLA is derived analytically from the behaviors of its subcomponents. These subcomponents are merely buffers or multi-input NOR gates, whose delays can be easily obtained using SDS. The effect of multiple inputs is considered in section 5.4. Two propositions are given for deriving the output timing of a series or parallel connection of transistors, the input timings of which are dynamically changing. Based on these two propositions, the timing of a bit-serial multiplier and interpolator is analyzed in section 5.5. The implementation of our hierarchical timing simulator is discussed in the next chapter.

5.1 Semantic Cells in Two-phase Synchronous Systems

In a two-phase synchronous system, all operations are initiated by global clocks. The period of a clock phase (the period from the rising edge of one clock to that of the other clock) is greater than the maximum amount of time necessary to complete any computations that occur during that phase; the results are then ready to be latched by the clock of the other phase. If the system is partitioned according to the phase relationships, then every partition of the network is a semantic cell. The reason is as follows: When clock ϕ_1 goes high, the inputs of all ϕ_1 cells switch to the results of the previous ϕ_2 computations, and remain stable during the rest of the clock phase. Although the outputs of these cells may switch several times during this period, the intermediate results are stoped by clock ϕ_2, and have no effect on the system behavior. Only the steady-state value of an output

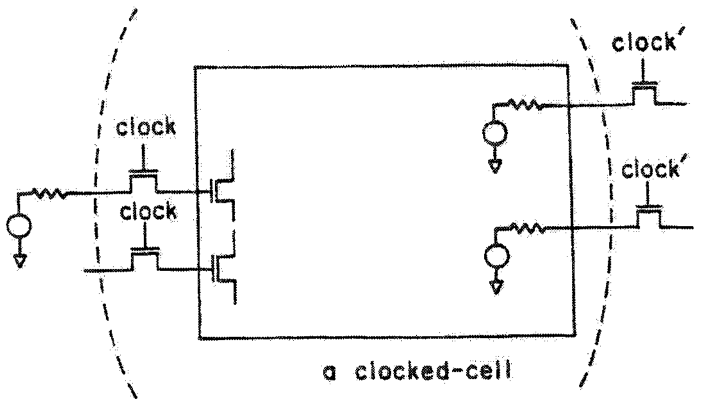


Figure 5.1. Clocked-cell: building block for synchronous systems

is of importance, and all internal nodes are stabilized at the end of this period. A ϕ_1 cell can be abstracted by its steady-state behavior to interface with the rest of the system, and thus is a semantic cell. Similarly for ϕ_2 cells.

A cell thus partitioned can be represented by the structure shown in Figure 5.1. All inputs of the cell (except clocks) are controlled by pass transistors gated by a clock signal. All outputs are static with no pass transistor blocking the way. Such a cell, called a "clocked-cell" by Chen [8], is the primitive building block of any synchronous system [1].

A semantic cell in a two-phase synchronous system is recursively defined: it is either a clocked-cell or a legal composition of semantic cells. A phase attribute is associated with each input to, and each output from a semantic cell indicating the active phases of the input and output ports of the cell. A legal composition of semantic cells is such that the following two conditions are satisfied:

a. ϕ_1 inputs connect to ϕ_2 outputs, and vice versa.

b. The period for both phases is sufficient for all circuits active during that phase to

[1] Pass transistors are the most common clocking primitive in MOS designs. The same comments apply to any clocked signal gating discipline.

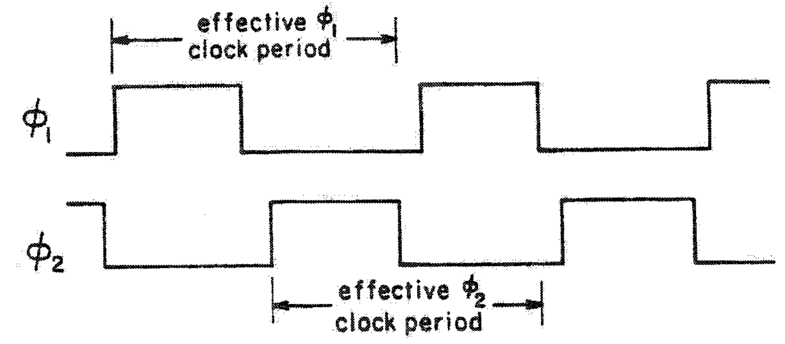

effective ϕ_1 clock period
ϕ_1
ϕ_2
effective ϕ_2 clock period

Figure 5.2. Effective ϕ_1 and ϕ_2 clock periods

reach their steady states. (5.1)

The checking of the first condition of (5.1) is purely syntactic. To assure the second condition, the minimum clock period of both phases can be determined using the timing model presented in this paper. The second condition is a strong one, and can be checked for every cell without regard to how it will be interconnected. It is often desirable, however, to relax the second condition to allow the borrowing of time between ϕ_1 and ϕ_2 [1], as often implemented in practical designs. A ϕ_1 cell is not required to reach its steady state before the rising edge of ϕ_2. As long as all inputs to ϕ_2 cells are stabilized by the falling edge of ϕ_2, a circuit can be made to function correctly. The "effective clock period" of a ϕ_1 cell starts from the rising edge of ϕ_1, through the rising edge of ϕ_2, till the falling edge of ϕ_2 (Figure 5.2). Similarly for a ϕ_2 cell. To allow borrowing time between ϕ_1 and ϕ_2, condition (5.1.b) is replaced by the following three weaker conditions. Note that the term "period" refers to the effective clock period of a cell.

a. The network activities of two consecutive periods of the cell are loosely coupled, so that each period can be considered independently.

b. The response of the cell at any period can be described analytically with reasonable complexity.

c. The period of each phase is sufficient for all inputs of that phase to stabilize before the falling edge of that phase. (5.2)

One possible interpretation of conditions (5.2.a) and (5.2.b) are as follows.

- When the cell is excited at any period, all nodes in the cell stabilize to a fully charged or discharged state before the next period starts. (5.2.a′)

If (5.2.a′) is satisfied, then we need not keep track of the stored charge of the internal nodes any more. Although it is necessary to record the logic states of storage nodes of a sequential circuit, the number of such nodes is much smaller than the total number of internal nodes in the network.

During one period of the cell, an input state may switch once, more than once, or not switch at all. In general, there are infinite number of input patterns that need to be considered. To make the situation tractable, the following requirement is imposed.

- During any given period, the state of every input port switches at most once.(5.2.b′)

In summary, conditions (5.2.a′) and (5.2.b′) determine whether or not a clocked-cell is a semantic cell. Note that these two conditions refer to interactions among cells. Therefore, whether a cell is a semantic cell depends not only on its content, but also on how it is interconnected with other cells.

Depending on individual applications, conditions (5.2.a′) and (5.2.b′) can be further relaxed to assure that every clocked-cell is a semantic cell. For instance, certain input states are allowed to switch more than once during a given period, or certain internal nodes are allowed not to stabilize before the next period starts. However, more complicated expressions are required to describe the timing behavior of such a semantic cell. In this paper, conditions (5.2.a′) and (5.2.b′) are used as an example to illustrate the general idea of our timing model. These two conditions, or the stronger conditions (5.1.a) and (5.1.b) are believed to be satisfied by clocked-cells of most digital circuits. The set of timing parameters presented at the end of section 5.2 are based on these two conditions.

Syntactically, a clocked-cell can be further decomposed into gate-level cells. According to the argument at the beginning of this section, clocked-cells were the smallest possible semantic cells in synchronous systems. With the relaxation of condition (5.1.b), however, it is possible to treat gate-level cells as semantic objects. In particular, if both conditions (5.2.a′) and (5.2.b′) are satisfied by the gate-level cells decomposed from a clocked-cell, then the timing behavior of the clocked-cell can be derived from the behaviors of these gate-level cells analytically. The PLA example of section 5.3 and the multiplier example of section 5.5 are treated this way.

MOS transistors are, in general, bidirectional devices; the signal may flow in either direction. For a semantic cell, however, the direction of every connection port must be determined. Note that this restriction does not exclude the possibility of an I/O port. Although the direction of such a port changes dynamically, at any given clock period, it is either an input or an output. One illegal situation is that two input ports of a cell are shorted (even temporarily) by a conducting path of pass transistors within the input network of the cell. We assume that some discipline has been applied in the input network to assure "no fighting" between driven signals [39].

5.2 Composition of Semantic Cells

Consider a semantic cell with n input ports ($I_{1,\ldots,n}$) and m output ports ($O_{1,\ldots,m}$). From the previous discussions, every input or output state of the cell switches at most once during any given clock period. Suppose, during the current clock period, the input states of the cell switch to $VI_{1,\ldots,n}$ at time $TI_{1,\ldots,n}$, respectively. Note that all TI values of a semantic cell under condition (5.1) are equal to 0, because new input values enter the cell on the rising edge of the clock (the reference time). In general, the TI's may admit any non-negative values. Suppose the output states are updated to $VO_{1,\ldots,m}$, and stabilized at time $TO_{1,\ldots,m}$, respectively. Note that, in general, the values of the TI's depend on the driving resistances at the input ports, and the TO's depend on the loading capacitances at the output ports. Consider the general situation of interconnections among cells indicated in Figure 5.3.a. An "interconnection" (or "net") is always of a tree structure. There are several loading nodes (referred to as nodes $N_{1,\ldots,s}$), and only one driving node (node N_o). Although there may be more than one driving node in the case of a bus, we assume a discipline in which only one driving node is active at any given time. All nodes in the net are logically equivalent because there are no transistors separating them. However, due to the stray resistances and capacitances of the interconnection wires, these nodes are not electrically equivalent, and their delay values are different. Every driving or loading node of the net is contained in a "partial" transistor group of the cell that contains the node. The term "partial" is used here to mean that the boundary of the transistor group is not defined unless the net is defined. According to the transistor level model presented in section 2.1, every such transistor group can be approximated by a two-port RC network for estimating delays. Note that the input port of the two-port RC network that contains the driving node N_o of the net is connected directly to the signal source (this network is denoted by M_o). The output port of the two-port RC network that contains the loading node N_i of the net is open (this network is denoted by M_i, $i = 1, \ldots, s$). Let N_i' denote the node at the output port of M_i. Referring to Figure 5.3.a, the net combines all these two-port RC networks into one RC network through which these cells interact (this resulting network is referred to as M_{NET}).

In section 2.4, a linear two-step algorithm (TREE) was presented for calculating the delays of all nodes in an RC tree network where every branch is a pure resistor. The two steps of this algorithm are repeated as follows.

1. The load capacitance C_i^L of every node i is accumulated and propagated from the loading ends towards the driving end of the tree. If node i is a leaf node, then $C_i^L = C_i$; otherwise, $C_i^L = C_i + \sum_j C_j^L$, where index j ranges over all succeeding

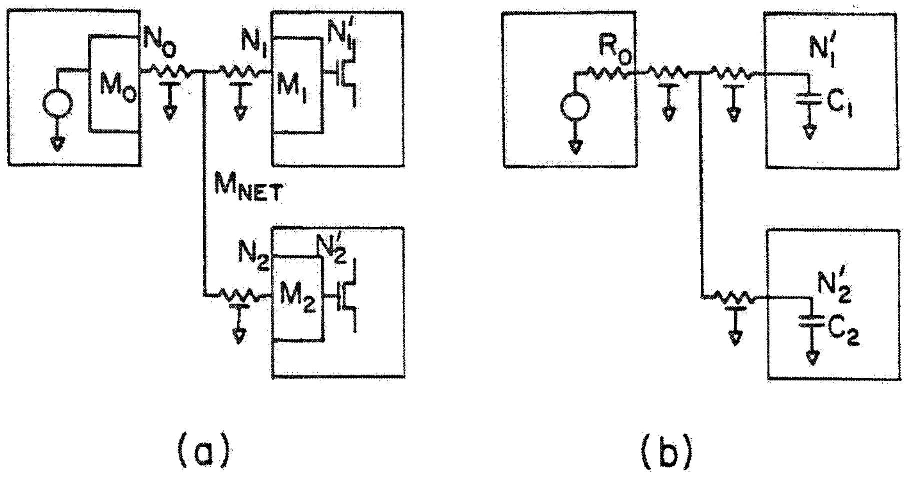


Figure 5.3. Cells and interconnections

nodes of node i, and C_i is the node capacitance of node i.

2. The delay of every node is calculated incrementally from the driving end towards the loading ends: $T_i = T_{p(i)} + r_i C_i^L$, where $p(i)$ is the parent node of node i, $T_{p(i)}$ is the delay of node $p(i)$, and r_i is the resistance between node $p(i)$ and node i.

This algorithm can be extended to deal with networks where every branch is a two-port RC network. The two steps of the modified calculation are as follows (the differences are underlined).

1′. load capacitance C_i^L: if node i is a leaf node, then $C_i^L = C_i$; otherwise $C_i^L = C_i + \sum_j \left(C_j^L + \underline{C_{i,j}}\right)$, where $C_{i,j}$ is the C parameter of the two-port RC network between node i and node j.

2′. delay value T_i: $T_i = T_{p(i)} + \underline{R_{p(i),i}}C_i^L + \underline{D_{p(i),i}}$, where $R_{p(i),i}$ and $D_{p(i),i}$ are the R and D parameters of the two-port RC network between node $p(i)$ and node i.

The delays of all nodes in network M_{NET} can be calculated using the extended TREE algorithm.

Consider the RC network derived from M_{NET} by the following operations:

- Replace the two-port RC network M_o by R_o, where R_o is the R parameter of M_o.
- Replace the two-port RC network M_i by C_i, where C_i is the C parameter of M_i, $i = 1, \ldots, s$.

This derived RC network is indicated in Figure 5.3.b. Every branch of this network is a resistor so that the original TREE algorithm can be applied. Except for the R_o (driving resistance) and C_i's (loading capacitances), all capacitances and resistances in this derived network come from interconnection wires. The delay properties of a composition of cells are based on the following theorem:

Theorem 5.1. Let T_i^* and T_i denote the delays of node N_i' in the original and derived M_{NET}, respectively, $i = 1, \ldots, s$. Then $T_i^* = T_i + D_o + D_i$, where D_o and D_i are the D parameters of M_o and M_i, respectively.

Proof:

First note that the load capacitance of every node in the net is the same for both the original and the derived M_{NET}. Let t_o^*, t_o, t_i^* and t_i be the delays of node N_o and N_i in the original and the derived M_{NET}, respectively. The proof of the theorem proceeds from the driving end towards the loading ends of M_{NET}.

- Node N_o: The signal source is the parent node of node N_o in both the original and the derived M_{NET}. In the original network, $t_o^* = R_o C_o^L + D_o$, where C_o^L is the load capacitance of node N_o. In the derived network $t_o = R_o C_o^L$. Therefore, $t_o^* = t_0 + D_o$.
- Node N_i: Note that all the branches in the net between node N_o and N_i are pure resistors. Therefore, the two algorithms add the same amount of delay to both t_o^* and t_o to obtain the values of t_i^* and t_i, respectively. Thus $t_i^* - t_i = t_o^* - t_o = D_o$.
- Node N_i': Node N_i is the parent node of N_i' in the original N_{NET}; thus $T_i^* = t_i^* + D_i$. In the derived M_{NET}, T_i and t_i refer to the same node, thus $T_i = t_i$.

Combining the results of the above three items, $T_i^* = T_i + D_o + D_i$. ∎

Note that D_o and D_i's in the above theorem are only functions of individual cells, and are independent of the interconnection. On the other hand, T_i is only a function of the interconnection, and is independent of the internal behavior of any of these cells. The ability to derive T_i^* from these three terms analytically makes it possible to abstract and compose timing behaviors of cells.

Timing Parameters

In summary, the timing behavior of a cell with n input ports $(I_{1,\ldots,n})$, m output ports $(O_{1,\ldots,m})$ and t internal states $(S_{1,\ldots,t})$ can be characterized by the following set of parameters:

1. VO_i for $i = 1, \ldots, m$: the logic state of output O_i after the network has stabilized.
2. TO_i for $i = 1, \ldots, m$: the time when output O_i is stabilized (all the output ports are open and input ports directly driven by signal sources).
3. VS_i for $i = 1, \ldots, t$: the internal state S_i after the network has stabilized.
4. CI_i for $i = 1, \ldots, n$: the load capacitance of input I_i.
5. RO_i for $i = 1, \ldots, m$: the driving resistance of output O_i. (5.3)

These parameters are evaluated each time any input of the cell switches during a clock period. In general, these parameters are functions of

- VI_i for $i = 1, \ldots, n$: the state of input I_i that excites the cell.
- TI_i for $i = 1, \ldots, n$: the time when the state of I_i switches to VI_i.
- $VS_i^{(0)}$ for $i = 1, \ldots, t$: the internal state S_i before the cell is excited.
- $VO_i^{(0)}$ for $i = 1, \ldots, m$: the state of output O_i before the cell is excited.

Among the five items of (5.3), RO's, CI's and TO's are generalizations of the R, C and D parameters of a two-port RC network. TO's and VS's describe the logical behavior of the transistor network. These two items are not necessary in a two-port RC network because the state of the output port simply follows that of the input port.

Note that, in this section, the input capacitances and output resistances are assumed to be fixed when the corresponding port is active at a given clock period. While this assumption is valid for output ports, it does not always hold for those input ports that connect laterally to pass transistors in the input network. The input capacitance of such a port depends on whether the pass transistors are on or off. The relative timing between the driving transistors and the pass transistors affect the timing of the input port in a significant way. A generalization of Theorem 5.1 is presented in section 5.4 when the effect of multiple inputs is discussed.

5.3 Example: nMOS Static PLA

Consider the structure of a static nMOS PLA shown in Figure 5.4.a. There are n non-feedback inputs, m non-feedback outputs, l feedback terms, and t product terms. According to the phase relationships, this circuit is partitioned into two clocked-cells: B_1 is active during ϕ_1, and B_2 is active during ϕ_2. The structure of B_2 is very simple: every feedback or output term corresponds to either an inverting or non-inverting buffer. Each buffer contains two inputs (I and ϕ_2), one output, and no internal states. Take an inverting buffer as an example. Its schematic diagram with associated circuit parameters is shown in Figure 5.4.b. The set of parameters for describing the timing behavior of this buffer is indicated in Table 5.1. VI_I is the state of input I, and TI_{ϕ_2} is the time when clock ϕ_2 rises. We assume that input I is stabilized before TI_{ϕ_2}; therefore TI_I is zero, and not shown in the table. The general situation is considered in the next section. A subscript $_2$ is associated with every parameter in the table indicating that they belong to clocked-cell B_2. The TO_2 values in the table are based on the assumption that the output state is switched; otherwise, TO_2 is 0.

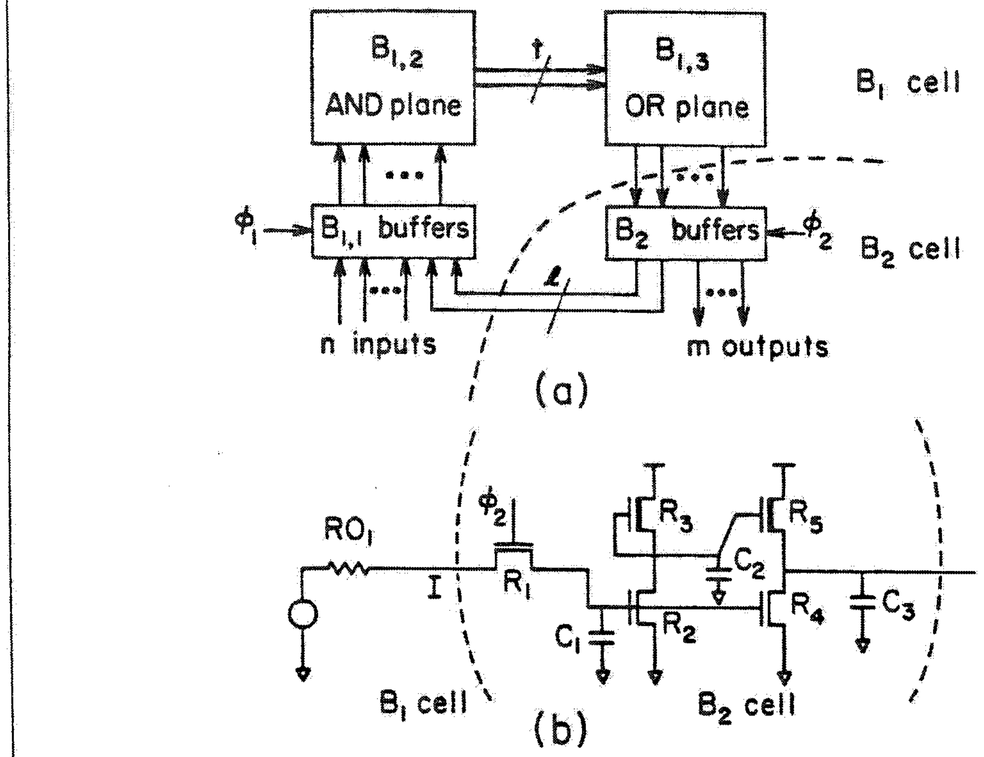


Figure 5.4. Structure of a static nMOS PLA

	$VI_I = 1$	$VI_I = 0$
VO_2	0	1
TO_2	$TI_{\phi_2} + R_1C_1 + R_2C_2 + R_4C_3$	$TI_{\phi_2} + R_1C_1 + R_3C_2 + R_5C_3$
CI_2	C_1	C_1
RO_2	R_4	R_5

Table 5.1. Timing parameters of an inverting buffer

Clocked-cell B_1 can be divided into three gate-level subcells: $B_{1,1}$ contains the input buffers; $B_{1,2}$ is the AND-plane; $B_{1,3}$ is the OR-plane. Both conditions (5.2.a′) and (5.2.b′) are satisfied by these three subcells; therefore the timing parameters of B_1 can be derived from those of the three subcells. $B_{1,1}$ can be treated in the same way as cell B_2. $B_{1,2}$ and $B_{1,3}$ are both (a collection of) multi-input NOR gates. Refer to Figure 5.4.c for the transistor circuit of an r-input NOR gate. The timing parameters of a single NOR gate

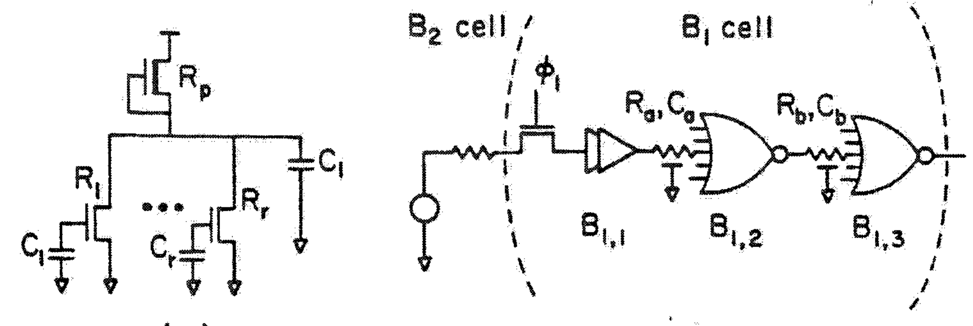


Figure 5.4. (continued) Structure of a static nMOS PLA

are considered for the following two cases:

1. All the pulldown transistors are turned off.
2. Only one, say the jth, pulldown transistor is turned on.

case	1 ($0 \rightarrow 1$)	2 ($1 \rightarrow 0$)
TO	$TO = (\max_{i=1}^{r} TI_i) + R_pC_L$	$TO = TI_j + R_jC_L$
$CI_{1,\ldots,r}$	$C_{1,\ldots,r}$	$C_{1,\ldots,r}$
RO	R_p	R_j

Table 5.2. Timing parameters of an r-input NOR gate

The timing parameters in the above two cases are indicated in Table 5.2[2]. The values in case 1 are used for $0 \rightarrow 1$ transitions, and those in case 2 are used for $1 \rightarrow 0$ transitions. The delay values estimated for $1 \rightarrow 0$ transitions are always conservative since only one pull-down transistor is assumed and are very useful in pattern-independent timing analysis [22,33]. The general situation when more than one pulldown transistors are turned on is discussed in Example 5.4 and Proposition 5.5 of the next section. Consider one special case in which all such transistors are turned on at approximately the same time; the two

[2] The composition of delays may be done either at the input side or at the output side. The TO values in the tables are composed at the input side.

R_j's in Table 5.2 change to $\frac{1}{\sum_i \frac{1}{R_i}}$, where the summation ranges over all transistors that are turned on. This approximation is useful when the delays of a NOR gate under specific input patterns are of interest. For the PLA, the output delay of an AND plane under these conditions will be dominated by the pullup of the following OR Plane, and hence the issue is largely academic.

Let $TO_{1,i}$, $RO_{1,i}$, and $CI_{1,i}$ be the timing parameters of cell $B_{1,i}$, for $i = 1, 2, 3$. These values can be determined from Table 5.1 and Table 5.2, respectively. Cell B_1 is composed from these three subcells, and the stray capacitances and resistances of the (long poly) interconnection wires are indicated in Figure 5.4.d. The lumped approximation of wires presented in [12] and the TREE algorithm are used in calculating the timing parameters of B_1. The result is as follows:

- $TO_1 = TO_{1,1} + RO_{1,1}(C_a + CI_{1,2}) + \frac{1}{2}R_aC_a + TO_{1,2} +$
  $RO_{1,2}(C_b + CI_{1,3}) + \frac{1}{2}R_bC_b + TO_{1,3}$.
- $RO_1 = RO_{1,3}$.
- $CI_1 = CI_{1,1}$.

Up to this point, B_1 and B_2 are analyzed independently. Again, TREE algorithm is used to combine these two cells together. During a ϕ_1 period, cell B_2 drives cell B_1, and the output timing of B_1 is equal to $TO_1 + RO_2 \cdot CI_1$. Likewise, the output timing of B_2, during a ϕ_2 period, is equal to $TO_2 + RO_1 \cdot CI_2$.

5.4 Effect of Multiple Inputs

Abstraction and composition of cells are discussed in the last section. The effect of one input on the output timing of a cell is illustrated by the PLA example. Note that the assumption that all pulldown transistors are turned on at approximatedly the same time reduces an r-input NOR gate to an equivalent one-input NOR gate (inverter). If the PLA had used a NAND structure, however, the reduction to a single wider transistor would not

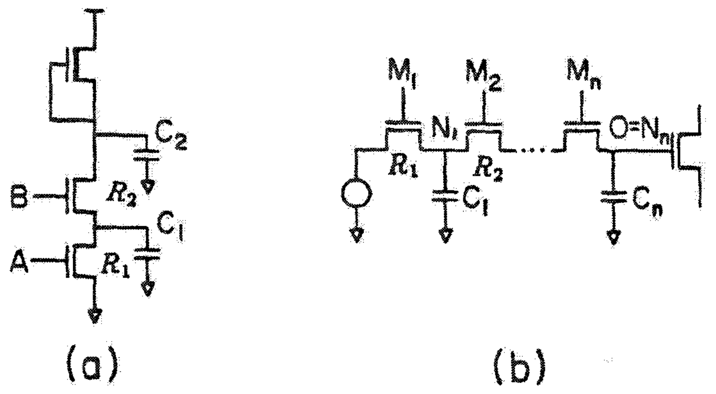


Figure 5.5. Series connection of transistors

have been possible. In this section, the effect of multiple inputs are considered for both series and parallel transistors connections.

Delay due to a series connection of transistors is probably the most important consideration in any delay model. In the following example, the output delay of a two-input NAND gate is analyzed. The result leads to a general expression (abstraction) for the output delay of any transistor chain. Without this abstraction, at least $2n$ constants (resistances and capacitances) need to be stored to calculate the delay, where n is the number of transistors in the chain. With this abstraction, only n constants are required. These n constants are denoted as $T_{1,\dots,n}$ in the example (also in the Proposition that follows). Without abstraction, the computation time is of order $O(n^2)$. With abstraction, the order reduces to $O(n)$. There is no approximation made during the abstraction process, and the delay calculated is as accurate as that done without abstraction.

Example 5.2. Refer to the two-input NAND gate in Figure 5.5.a. Consider the following three cases that switch the output state of the gate from 1 to 0. The transistor-level model presented in Chapter 2 is used for the delay calculation.

1. $B = 1$ initially, and A switches from 0 to 1: The delay of the output after A switches is $T_1 = R_1(C_1 + C_2) + R_2C_2$.

2. $A = 1$ initially, and B switches from 0 to 1: Capacitance C_1 is discharged to GND already. The delay of the output after B switches is $T_2 = (R_1 + R_2)C_2$.

3. Both A and B switch from 0 to 1: Let TI_A and TI_B denote the time when A and B switch, respectively.

   - $TI_A < TI_B$: Let $T_1^* = R_1C_1$, the time it takes to fully discharge capacitance C_1 with transistor M turning off. If $TI_B \geq TI_A + T_1^*$, then this case reduces to case 2, and the output timing $TO = TI_B + T_2$. On the other hand, if $TI_B < TI_A + T_1^*$, then capacitance C_1 is only partially discharged when B switches. Define $\rho = \frac{TI_B - TI_A}{T_1^*}$. By using linear interpolation to approximate the stored charge of C_1, the output delay after B switches is $T_1' = R_1((1-\rho)C_1 + C_2) + R_2C_2$. As a result, the output timing $TO = TI_B + T_1' = TI_A + \rho T_1^* + T_1' = TI_A + T_1$, which is the same as the result of case 1. Note that the dynamic charge sharing effect is directly modelled by our definition of delay [27].

   - $TI_B \leq TI_A$: This reduces to case 1.

In summary, when both inputs switch, $TO = \max(TI_A + T_1, TI_B + T_2)$. The output timing of a series of transistors can in general be determined in the same way. ▮

Proposition 5.3. Let $M_1, \ldots, M_n$ be a series of transistors connecting a signal source (VDD or GND) to an open output O (Figure 5.5.b). Consider the case when all these transistors are turned on, and the output switches to the logic state of the source after certain delay. Let TI_i denote the time when transistor M_i is turned on, $i = 1, \ldots, n$. If M_i is on initially, then $TI_i = 0$. Let N_i be the node between transistors M_i and M_{i+1}, $i = 1, \ldots, n-1$. Let D_i and C_i be the D and C parameters of the two-port RC network between node N_i and output O with $M_{i-1}, \ldots, M_n$ turning on. In particular, C_n is the node capacitance of output O, and $D_n = 0$. Let R_i be the source resistance of node N_i with transistors $M_1 \ldots, M_i$ turning on, $i = 1, \ldots, n$. Define $\underline{T_i = R_i \cdot C_i + D_i}$, $i = 1, \ldots, n$. This value is $\underline{\text{the output delay after transistor } M_i \text{ is turned on, under the condition that all the other}}$

transistors in the chain are turned on initially. Then the output timing $TO = \max_{i=1, TI_i>0}^{n}(TI_i + T_i)$.

Proposition 5.2 is very useful in determining the critical path of an MOS circuit. Consider a chain consisting of three transistors. Depending on the relative time each transistor is turned on, the critical path to the output is different. For instance, let T_1 =8ns, T_2 =6ns, and T_3 =3ns. The critical path goes through M_1, M_2 or M_3, depending on the input timings, as indicated in Table 5.6.

TI_1	TI_2	TI_3	TI_1+T_1	TI_2+T_2	TI_3+T_3	critical path
1	1	1	9	7	4	M_1
1	5	4	9	11	7	M_2
1	1	8	9	7	11	M_3

Table 5.3. Timing parameters and critical paths (unit: ns)

In many situations, a series of transistors spans over two cells, and is thus split into two different phases. For instance, transistors M and N of Figure 5.6 are in two different cells, and are active at phase 1 and phase 2, respectively. According to Proposition 5.3, the timing of node B when it switches low is $\max(TI_A + T_A, TI_{\phi_2} + T_{\phi_2})$, where TI_A is the time when node A rises, and $T_A = r_1(C_1 + C_2) + r_2C_2$; TI_{ϕ_2} is the time when clock ϕ_2 rises, and $T_{\phi_2} = (r_1 + r_2)C_2$. Note that T_A includes delays of both phases, and can be naturally split into two terms: $T_{A,1} = r_1C_1$ and $T_{A,2} = (r_1 + r_2)C_2 = T_{\phi_2}$. When ϕ_2 is low and transistor N is off, $T_{A,1}$ is the time required to discharge capacitance C_1. Usually, capacitance C_1 is fully discharged before ϕ_2 rises, and the timing of node B is $TI_{\phi_2} + T_{\phi_2}$. In general, if the borrowing-time technique is employed, and C_1 is or is not settled before ϕ_2 goes high, then the timing of B is $\max(TI_A + T_{A,1}, TI_{\phi_2}) + T_{\phi_2}$. Although this formula is equivalent to the one in Proposition 5.3, it distinguishes the delays contributed from individual phases, and is thus more convenient to use. The following corollary generalizes the above arguments.

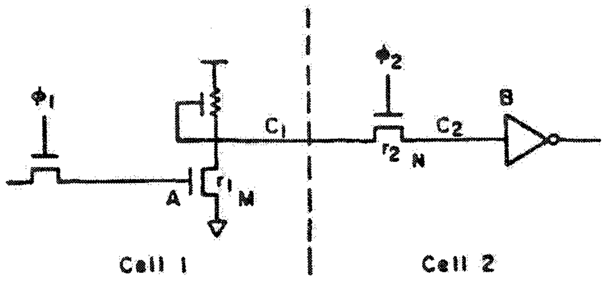


Figure 5.6. Series connection of transistors across cell boundary

Corollary 5.4. Refer to Proposition 5.2 and Figure 5.5. Suppose M_ϕ is a distinguished transistor along the transistor chain $M_1, \cdots, M_n$. With respect to this transistor, every T_i term, with $1 \leq i \leq \phi$, can be split into two terms: $T_{i,1} = T_i - T_\phi$, and $T_{i,2} = T_\phi$. In terms of these new parameters, the output timing is expressed as $TO = \max_{i=1}^{\phi}(TI_i + T_{i,1}) + \max_{i=\phi}^{n}(TI_i + T_i)$. ∎

For RC tree networks, no simple abstraction can be made that is as general and simple as Proposition 5.3 (and Corollary 5.4) for RC chains. Based upon the functionality of individual circuits, however, the above proposition can be extended to deal with tree networks. For instance, in the common case where two branches of a tree network are mutually exclusive, then the tree reduces to two chains which can be analyzed separatedly. If the load on a node in a tree network is dynamically changing due to the turn-on and turn-off of side branches, the maximum possible load may be used statically. Again, the tree network reduces to a chain. The estimated delay, in this case, is always conservative.

In theorem 5.1, the input capacitance of an input port is assumed to be fixed when this port is active during a given clock period. Consider the net of Figure 5.7. Suppose the driving node of this net is active during ϕ_1, and all the input ports driven by this net are active during ϕ_2. Refer to Figure 5.1 for the structure of a clocked-cell. Every input port of this net is controlled by a pass transistor gated by a clock signal active during ϕ_2 (this

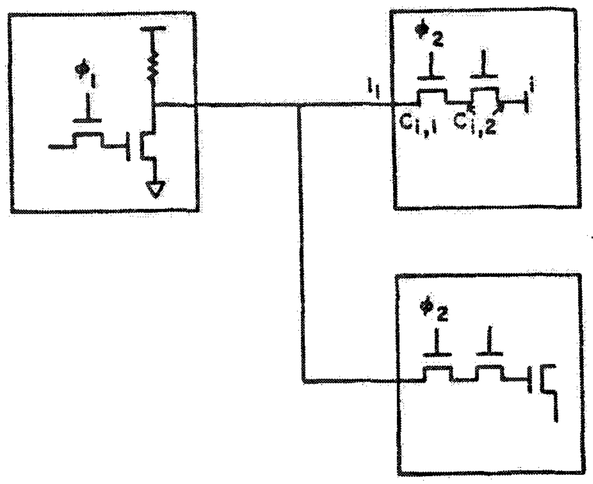


Figure 5.7. Cell interconnections with pass transistors in the input ports

transistor is referred to as the "clocked pass transistor" of the input). Given an input port I_i, the "total input capacitance" C_i is defined to be the input capacitance when all the pass transistors are turned on. With respect to the clocked pass transistor, C_i can be split into two parts: $C_{i,1}$: the input capacitance when the clocked pass transistor is turned off, and $C_{i,2}$: $C_i - C_{i,1}$. Note that, although the input signal does not have to be settled when the clock goes high (borrowing time technique of section 5.1), the controlling signals of the other pass transistors are required to be stable. This requirement reduces the number of possible input capacitances to only two. More discussions on input capacitances are given in Chapter 6.

As a generalization of Corollary 5.4, two delays are associated with input port I_i: $D_{i,1}$ for phase 1, and $D_{i,2}$ for phase 2. The value of these two delays can be calculated by applying the TREE algorithm to some "reduced networks" of the net (refer to section 5.2): the output port is replaced by its driving resistance; the input ports I_j's are replaced by corresponding $C_{j,1}$'s and $C_{j,2}$'s, respectively.

Let T_o be the TO parameters of the output port of the net. Note that this value does

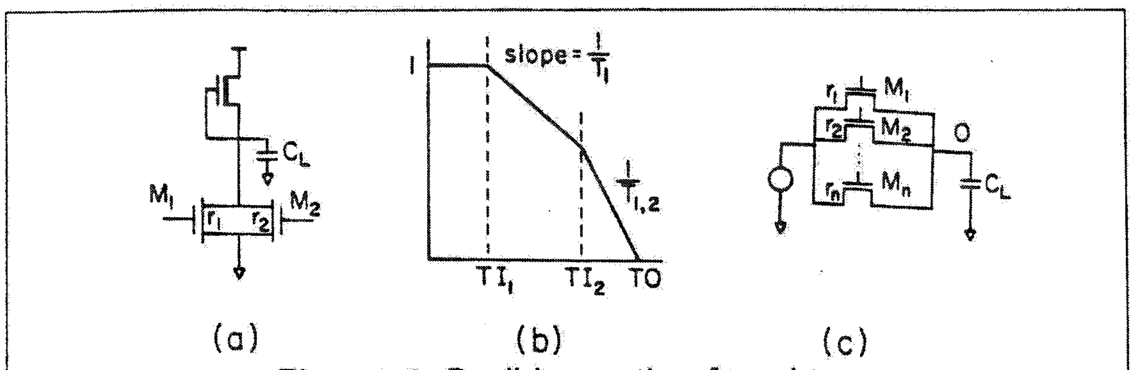


Figure 5.8. Parallel connection of transistors

not include the effect of loading from the net. Suppose the "clocked pass transistor" of input port I_i turns on at time T_ϕ; then the effective input timing of I_i (T_i) is

$$T_i = \max(T_o + \underline{D_{i,1}}, T_\phi) + \underline{D_{i,2}}. \tag{5.5}$$

Formula (5.5) is always conservative, and gives correct results when the "clocked pass transistors" of all the input ports of a given net are turned on at approximately the same time.

The following example deals with parallel connections of transistors. Similar results as example 5.2 and proposition 5.3 are derived.

Example 5.5. Consider the 2-input NMOS NOR gate of Figure 5.8.a. The ON-resistances of the two input transistors (M_1 and M_2) are r_1 and r_2, respectively. The loading capacitance at the output is C_L. Initially, both transistors M_1 and M_2 are off. During the current clock period, either or both of the transistors are turned on.

1. Only one transistor is turned on: If it is M_1 that is turned on, then the output delay is $T_1 = r_1 C_L$. If it is transistor M_2 that is turned on, then the delay is $T_2 = r_2 C_L$.
2. Both transistors are turned on: Let TI_1 and TI_2 denote the time when transistors M_1 and M_2 are turned on, respectively. Assume that $TI_2 \geq TI_1$, and let $\rho_1 = \frac{TI_2 - TI_1}{T_1}$. If $\rho_1 > 1$, then this case is no different from case 1 (when M_2 is turned on, C_L is

fully discharged). If $0 \le \rho_1 < 1$, then the delay of the output after M_2 turns on is $(1-\rho_1)T_{1,2}$, where $T_{1,2} = \frac{1}{\frac{1}{T_1}+\frac{1}{T_2}}$ (Figure 5.8.b). Similarly for the case that $TI_2 < TI_1$.

Parallel connections of transistors in general can be dealt with in the same way. ∎

Proposition 5.6. Let $M_1, \ldots, M_n$ be n parallel transistor connecting a source to an open output O (Figure 5.8.c). Initially, all M_i's are turned off, and during the current clock period, at least one transistor is turned on. Let TI_i denote the time when transistor M_i is turned on. If M_i remains off, then $TI_i = 0$. Define $T_i = r_i C_L$, where r_i is the ON-resistance of M_i and C_L is the node capacitance of O. This value is the delay of the output when it is only driven by transistor M_i. Let $p(1), \ldots, p(n)$ be a permutation of $1, \ldots, n$ such that $TI_{p(1)} \le \cdots \le TI_{p(n)}$. Then the output timing $TO = H_n$, where H_i, $i = 1, \ldots, n$ are recursively defined as follows:

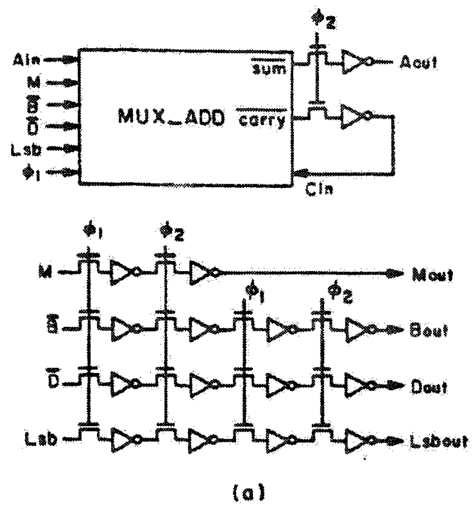
$$\begin{aligned} H_1 = {} & \text{if } TI_{p(1)} = 0 \text{ then } 0 \text{ else } TI_{p(1)} + T_{p(1)} \\ H_i = {} & \text{if } TI_{p(i)} = 0 \text{ then } H_{i-1} \quad i = 2, \ldots, n \\ & \text{else if } TI_{p(i)} > H_{i-1} \text{ then } H_{i-1} \\ & \text{else } TI_{p(i)} + (1-\rho_1)\cdots(1-\rho_n)T_{p(1),\cdots,p(i)} \end{aligned}$$

where $\rho_i = \dfrac{TI_{p(i+1)} - TI_{p(i)}}{H_i - TI_{p(i)}}$, and $T_{p(1),\ldots,p(i)} = \dfrac{1}{\frac{1}{T_{p(1)}} + \cdots + \frac{1}{T_{p(n)}}}$.

The above equation is much more complicated than that for series connections of transistors. Approximations of this expression under special conditions were presented in the PLA example of the last section. These approximations are adequate for most practical applications. ∎

5.5 Example: UPE, a Pipelined Multiplier and Interpolator

The result of section 5.4 is applied to analyze the timing of a bit-serial multiplier and interpolator called UPE (Universal Processing Element) [50,51], one stage of which is



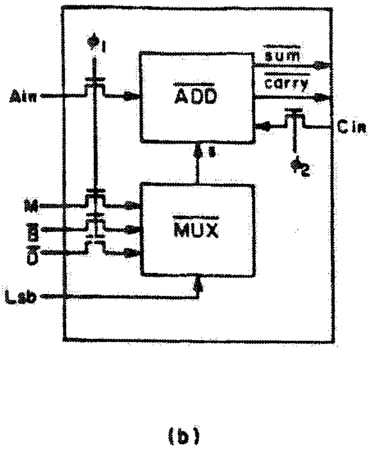


Figure 5.9. Block diagram of one stage of UPE

indicated in Figure 5.9.a[3]. The main processing section of the circuit, referred to as cell MUX_ADD, is indicated in Figure 5.9.b. The rest of the circuit is merely a set of shift registers.

In addition to clock ϕ_1, there are six inputs to cell MUX_ADD: Lsb, M, $\overline{B}$, $\overline{D}$, Ain and Cin, and two outputs: $\overline{sum}$ and $\overline{carry}$. Lsb is active during ϕ_2, and the other five inputs are active during ϕ_1. Note that cell MUX_ADD is a clocked-cell — the smallest possible semantic object in the strict sense (section 5.1). To analyze timing, however, it is more convenient to divide the cell into the following two gate-level subcells: $\overline{ADD}$ and $\overline{MUX}$, the details of which are shown in Figure 5.10 and Figure 5.11, respectively. Parenthesized numbers in these two figures are transistor ON-resistances and wire resistances (if not

[3] A chip containing ten UPE's has been designed, fabricated, and integrated into a sound-synthesis system. This chip handles both positive and negative numbers under 2's complement representation. For simplicity, a portion of the design is omitted (for instance, the sign extender and carry kill circuit), and the circuit described in this section only works for positive numbers. The architecture of the UPE is based on that proposed by Lyon [29].

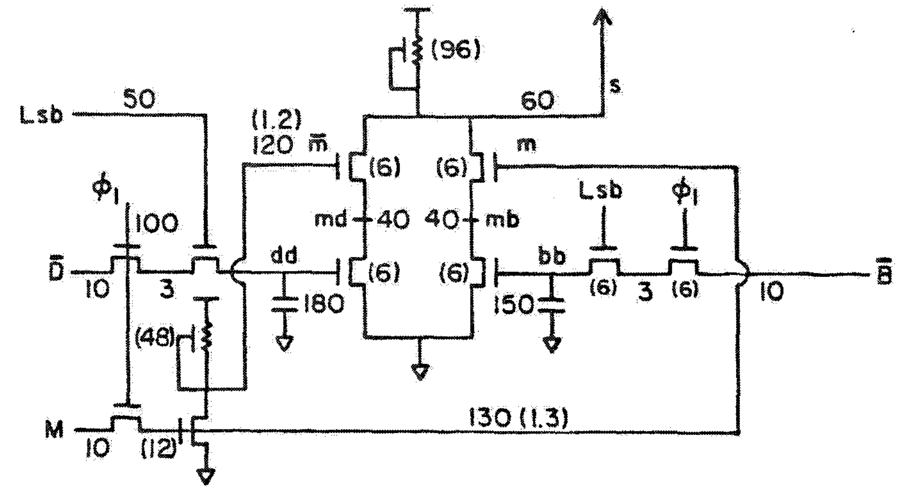


Figure 5.10. The circuit diagram of cell $\overline{\text{MUX}}$

negligible); unparenthesized numbers are node capacitances. The unit for resistances is KΩ ($10^3\Omega$), and that for capacitances is fF (10^{-15}F).

Cell $\overline{\text{MUX}}$

Consider subcell $\overline{\text{MUX}}$ (Figure 5.10). This cell contains five inputs: ϕ_1, Lsb, M, $\overline{\text{B}}$ and $\overline{\text{D}}$, and one output: s. Input Lsb is not high for every clock cycle, and nodes bb and dd act like two internal state variables (the capacitances associated with these two nodes are relatively large). The logic and timing of cell $\overline{\text{MUX}}$ are analyzed through the following steps. Clock ϕ_1 is assumed to be high for (5.6)–(5.10).

1. The logic of cell $\overline{\text{MUX}}$:

   if Lsb **then begin** bb:=$\overline{\text{B}}$; dd:=$\overline{\text{D}}$; "otherwise unchanged" **end**;
   if M **then** s:= not bb **else** s:= not dd (5.6)

2. The timing of signals m and $\overline{\text{m}}$ (T_m, $T_{\overline{m}}$) when input M switches is

   T_m:= max(T_{ϕ_1},T_M)+0.9ns; "$(6 \cdot 140 + \frac{1}{2} \cdot 130 \cdot 1.3)$ps"
   if M **then** $T_{\overline{m}}$:= max(T_{ϕ_1},T_M)+2.4ns "$6 \cdot 140 + 12 \cdot 120 + \frac{1}{2} \cdot 120 \cdot 1.2$"
   else $T_{\overline{m}}$:= max(T_{ϕ_1},T_M)+6.7ns "$6 \cdot 140 + 48 \cdot 120 + \frac{1}{2} \cdot 120 \cdot 1.2$" (5.7)

T_{ϕ_1} is the time when ϕ_1 goes high; T_M is the time when input M switches (with ϕ_1

low). If M does not switch during the current clock period, then both T_m and $T_{\overline{m}}$ are equal to 0. The formula of (5.7) is based on Corollary 5.4 (series connection of transistors across phase boundary), assuming that the input port is driven directly by the signal source (VDD or GND). The effect of driving resistance and loading capacitance (140fF) is dealt with by composition (detailed in section 6.1).

3. The state of bb switches when Lsb is high, and $\overline{B}$ is different from $bb^{(0)}$ (the state at previous clock cycle). The timing of bb (T_{bb}) when it changes state is

```
if Lsb and ($\overline{B} \neq bb^{(0)}$)
then $T_{bb}$:=max($T_{\phi_1}$,$T_{\overline{B}}$,$T_{Lsb}$)+1.8ns.        (5.8)
```

T_{Lsb} is the time when input Lsb goes high; $T_{\overline{B}}$ is the time when $\overline{B}$ switches. Likewise, the timing of dd (T_{dd}) when it changes state is

```
if (LSB) and ($\overline{D} \neq d^{(0)}$)
then $T_{dd}$:=max($T_{\phi_1}$,$T_{\overline{D}}$,$T_{Lsb}$)+2.2ns.        (5.9)
```

4. Finally, the timing of output s (T_s) is expressed in terms of T_m, $T_{\overline{m}}$, T_{bb} and T_{dd} as follows:

```
if M then begin
    if bb then $T_s$:=max($T_{bb}$+1.0ns,$T_{\overline{m}}$+0.7ns)        "s falls"
    else begin                                                           "s rises"
        if dd then $T_s$:=max($T_{bb}$, $T_{\overline{m}}$+9.6ns
        else $T_s$:=min(max($T_{bb}$,$T_{dd}$)+3.8ns,max($T_{bb}$,$T_{\overline{m}}$))+ 9.6ns
    end
end else begin
    if dd then $T_s$:=max($T_{dd}$+1.0ns,$T_m$+0.7ns)                   "s falls"
    else begin                                                           "s rises"
        if bb then $T_s$:=max($T_{dd}$,$T_m$)+9.6ns
        else $T_s$:=min(max($T_{dd}$,$T_{bb}$)+3.8ns,max($T_{dd}$,$T_m$))+9.6ns
    end
end.                                                                     (5.10)
```

"T_s:=max(T_{bb}+1.0ns,$T_{\overline{m}}$+0.7ns)" in the second line of (5.10) is based on Proposition 5.3 (series connection of transistors). The formula in the fourth and fifth lines of

(5.10) is based on the fact that output s does not start charging up until 1) bb goes low, and 2) either $\overline{\text{m}}$ or dd goes low. The delay 9.6ns is due to the capacitances of node s and node mb; 3.8ns is due to the capacitance of node md.

In Chapter 6, the implementation of an event-driven timing simulator is discussed. Every event created during the simulation process corresponds to a node. When this event is evoked, the state of the node is switched, the cell that is driven by the node is excited, and the code that describes the logic and timing of the cell is executed. Let n_0 refer to the node that corresponds to the current event. If it is cell $\overline{\text{MUX}}$ that is excited, then n_0 is ϕ_1 Lsb, M, $\overline{\text{D}}$ or $\overline{\text{B}}$. The logic and timing of cell $\overline{\text{MUX}}$ can be obtained by combining (5.6)–(5.10) as follows:

```
procedure $\overline{\text{MUX}}$;
begin
    if $\phi_1$ then begin
        case $n_0$ of
            M: Eq. (5.7);
            $\overline{\text{B}}$: Eq. (5.8);
            $\overline{\text{D}}$: Eq. (5.9);
            LSB: begin Eqs. (5.8); (5.9); end;
            $\phi_1$: begin Eqs. (5.7); (5.8); (5.9); end;
        end;
        Eqs. (5.6); (5.10);
    end;
end.                                                        (5.11)
```

Cell $\overline{\text{ADD}}$

There are four inputs to cell $\overline{\text{ADD}}$ (Figure 5.11): ϕ_1, Ain, Cin and s. For convenience, this cell is further divided into three subcells: $\overline{\text{MAJ}}$, $\overline{\text{XOR}}$ and XOR, each of which can be dealt with using the same technique as we did with cell $\overline{\text{MUX}}$. The results are summarized in Table 5.4–5.6. The following remarks refer to these three tables.

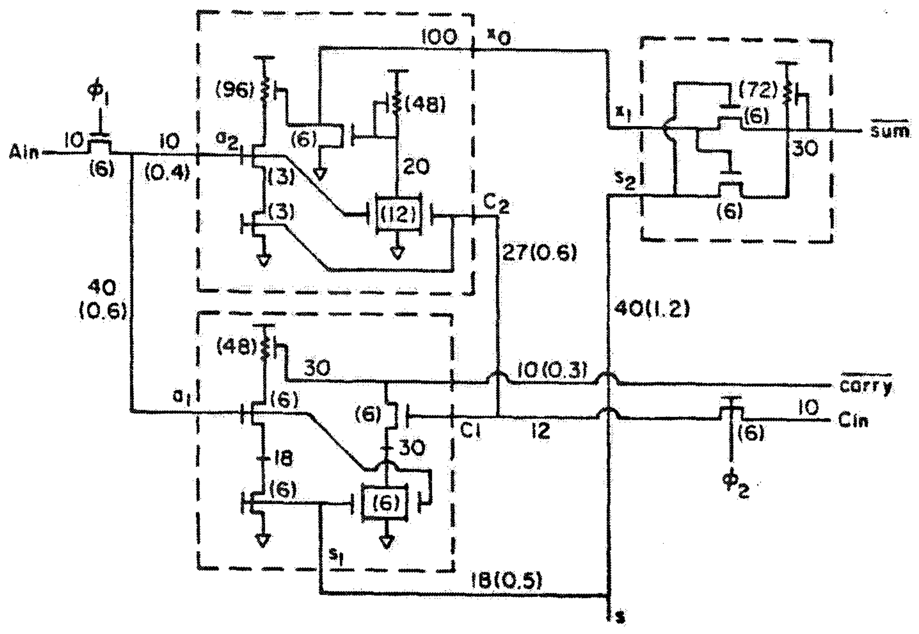


Figure 5.11. The circuit diagram of cell $\overline{\text{ADD}}$

Remarks:

- All input timing in these tables refers to the time when the corresponding input stabilizes and starts to switch the transistors to which it is connected. The effect of wire delays is handled by composition.
- Table 5.4 indicates the logic and timing of cell $\overline{\text{MAJ}}$. There are three inputs to this cell: a1, s1, and c1. n_o is the node that switches at the current event (n_o= a1, s1 or c1). The values in the first three columns of the table indicate the input states after node n_o switches. The fourth column shows the output state; the last three columns indicate the output timing, which is a function of both n_o and the input states. Similarly for Table 5.5 and Table 5.6.
- According to Proposition 5.3, those items in Table 5.4–5.5 that are marked with *'s should be of the form "max(T_1+D_1,T_2+D_2)," where T_1 and T_2 are corresponding input timings, and D_1 and D_2 are constants. In these two tables, however, they

inputs			output	output timing $T_{\overline{carry}}$		
a1	s1	c1	$\overline{carry}$	n_o=a1	n_o=s1	n_o=c1
0	0	0	1	$\min(T_{\overline{carry}}, T_{a1}+1.9ns)$	– –	$\min(T_{\overline{carry}}, T_{c1}+1.9ns)$
0	0	1	1	T_{a1}+3.4ns	T_{s1}+3.4ns	– –
0	1	0	1	T_{a1}+1.9ns	– –	T_{c1}+1.9ns
0	1	1	0	– –	T_{s1}+0.7ns	T_{c1}+0.7ns*
1	0	0	1	– –	T_{s1}+2.8ns	T_{c1}+2.8ns
1	0	1	0	T_{a1}+0.9ns	– –	T_{c1}+0.9ns*
1	1	0	0	T_{a1}+0.6ns*	T_{s1}+0.6ns	– –
1	1	1	0	– –	– –	– –

Table 5.4. Logic and timing of cell $\overline{\text{MAJ}}$

inputs		output	output timing T_{xo}	
a2	c2	xo	n_o=a2	n_o=c2
0	0	0	T_{a2}+1.3ns	T_{c2}+1.3ns
0	1	1	T_{a2}+9.6ns	T_{c2}+9.8ns
1	0	1	T_{a2}+9.8ns	T_{c2}+9.6ns
1	1	0	T_{a2}+0.7ns*	T_{c2}+0.7ns

Table 5.5. Logic and timing of cell XOR

inputs		output	output timing $T_{\overline{sum}}$	
xi	s2	$\overline{sum}$	n_o=xi	n_o=s2
0	0	1	T_{xi}+2.2ns	T_{s2}+2.2ns
0	1	0	T_{xi}+0.4ns	T_{s2}+0.4ns
1	0	0	T_{xi}+0.4ns	T_{s2}+0.4ns
1	1	1	T_{xi}+2.2ns	T_{s2}+2.2ns

Table 5.6. Logic and timing of cell $\overline{\text{XOR}}$

are all simplified into the form "$\max(T_1,T_2)+\max(D_1,D_2)$." For instance, the exact formula for the item in the (n_o=a1) column and (110) row of Table 5.4 should be $\max(T_{a1}+0.5ns, T_{s1}+0.6ns)$. Note that a1 is the node that switches in the current event, so that $T_{a1} \geq T_{s1}$. The above formula is simplified to T_{a1}+0.6ns, as shown in the table. The delay estimated using this simplified formula is always conservative, and the over-estimated amount is at most 0.1ns.

- "– –" in Table 5.4 (cell $\overline{\text{MAJ}}$) means that neither the output state nor the output timing is affected by the current event. For instance, s1 switching low when both a1 and c1 are already low does not reduce the delay to charge up the output node. Although s1 switching high when both a1 and c1 are already high may reduce the delay to discharge the output node, the reduced amount is always very small (at most a few tenths of a nanosecond) and can be neglected. For cells XOR and $\overline{\text{XOR}}$, the output state switches when any input state switches, so that no "– –" appears in these two tables.
- Note that cell $\overline{\text{XOR}}$ is not a semantic object because inputs xi and s2 are laterally connected to VDD from inside the cell, a characteristic of an output, not an input. However, both of these inputs are driven directly by a restoring node (no pass transistor in between) with W/L ratios of the pulldown transistors properly adjusted. Refer to Figure 4.7 for the SPICE simulation result of such a circuit. Although there is a big glitch of the output node when the input states switch from (0,1) to (1,0), the input nodes are quite stable. In this particular composition environment, cell $\overline{\text{XOR}}$ can be treated as if it were a semantic object. Furthermore, the values of output timing in Table 5.6 already take the driving resistances into account because it is more convenient to handle cell $\overline{\text{XOR}}$ this way. These values are valid only for this particular composition. On the other hand, the values in Table 5.4 and 5.5 are applicable to any semantic composition of cells $\overline{\text{MAJ}}$ and XOR.

There are four internal nets of cell $\overline{\text{ADD}}$: input s drives input s1 of cell $\overline{\text{MAJ}}$ and input s2 of cell $\overline{\text{XOR}}$; output xo of cell XOR drives input xi of cell $\overline{\text{XOR}}$; primary input Ain drives input a1 of cell $\overline{\text{MAJ}}$ and input a2 of cell XOR. primary input Cin drives input c1 of cell $\overline{\text{MAJ}}$ and input c2 of cell XOR. Note that, without affecting the result, the parasitic capacitances and resistances of a net can be arbitrarily distributed into the cells to which the net is connected. In net (xo,xi), for instance, all capacitances are lumped into node xi, and the delay of the net becomes part of the internal delay of cell XOR (Table 5.5). In net (s,s1,s2), 60fF are considered inside cell $\overline{\text{MUX}}$ as the node capacitance of output s. The rest of the net does not belong to any cell, and is taken care of by composition. There is no special reason, other than for illustration, to partition these nets this way. The delay between driving and loading nodes of the above four nets can be calculated using the TREE algorithm (detailed in section 6.7).

In Chapter 6, the UPE is used as an example to illustrate an implementation of the hierarchical timing simulation model. First, the logic and timing of one UPE stage is specified using the result of this section. Based on this specification, a multi-stage UPE is simulated, and analyzed for the next level of abstraction (from bit-level to word-level). Finally, a second-order filter consisting of two 32-bit UPE's is analyzed using this derived word-level representation.

5.6 Summary

The extension of the (R, C, D) characterization of two-port RC networks to semantic cells at any higher level of representation explores the real capabilities of our timing model. This chapter serves as a guideline for characterizing the logic and timing behavior of semantic cells independent of their composition environment. The composition is handled automatically by using the TREE algorithm, the correctness of which is guaranteed by Theorem 5.1. The only thing the user has to make sure of is that the composition preserves the semantics (or behavior) of individual cells. As discussed in section 5.1, this checking is usually syntactic, and can be done very easily.

Most discussions of this chapter are for gate-level cells and clocked-cells. To make abstraction possible, we first made the assumption that, at every clock period, every input and output port of a cell switches at most once. Furthermore, all internal nodes stabilize to a fully charged or discharged state before the cell is excited again at the next clock period.

With these assumptions, the timing of a static nMOS PLA was derived analytically based on a single-input/single output model. For a multi-input cell, the situation is more difficult because of the variations in the relative timing of inputs. Proposition 5.3 deals with series connections of transistors in a simple way, which leads to an efficient treatment of the 6-input/2-output circuit of section 5.5. The results presented in this chapter can be applied to either compiled code (hard-coded timing model) or event-driven timing simulation. In an event-driven simulation environment, only one event is evoked and one input switches at one time (although the possibility that several inputs switch at the same physical time is not excluded). The circuit status due to previous input events (at the same clock period) is recorded in a very small number of variables. With every input port, for instance, are associated a logic value and a timing. Some of these variables can be eliminated if the output delay does not depend on them in a significant manner.

Although the logic and timing descriptions derived in this chapter are represented in an executable form, they are almost as accurate as the results obtained by running SDS directly on the transistor listings of the corresponding circuits. It is quite possible to trade complexity for accuracy. The larger the error that can be tolerated, the simpler these descriptions can be. One can expect that a lot of optimizations and simplifications will be excercised when timing is abstracted from the bottom to the top level of a hierarchy.

Chapter 6

Implementation of a Hierarchical Timing Simulator

As mentioned at the beginning of Chapter 5, there are two kinds of cells in our timing simulation model: leaf cells and composition cells. With a leaf cell is associated a logic and timing behavior which is valid for all possible input patterns and loading and driving conditions of the cell. Two examples (PLA and UPE) were given in section 5.3 and section 5.5 to illustrate the principles of deriving the behavior of a cell from its circuit structure. The result is expressed in an executable form, and serves as an abstraction of the cell for interfacing with other cells. With a composition cell, on the other hand, are associated a number of subcells, and a set of nets indicating how these subcells are interconnected. Our hierarchical simulator is based on the "behavior" models of leaf cells and the "structure" information of composition cells.

Note that a leaf cell may be as simple as a single inverter, or as complicated as the entire data path of an ALU. Obviously, it requires the flexibility of a general purpose programming language to specify the behavior of such a cell. This is quite a contrast to most other simulators in which circuits are expressed in terms of only a few different types of primitives whose behaviors are very rigid, and predefined by the simulators.

Instead of designing yet another hardware description language, we embedded the simulator in an existing programming environment. Among several popular languages available today, Smalltalk [18,19] comes closest to our need for the following reasons:

1. The programming model: the object-oriented programming model of Smalltalk precisely matches our semantic cell-oriented simulation model. The "messages," "methods," and "data" of an object correspond respectively to the interface parameters,

internal behavior, and internal states of a cell in our simulation model. By nesting object definitions, cells can be easily specified according to the hierarchical levels appropriate to the design.

2. The debugging capabilities: The environment in which users set up and modify the simulation for their designs, investigate the simulation results, etc., can be naturally done using the hierarchical "inspector" of Smalltalk. The internal nodes and substructures of a design can be accessed in the hierarchical order in which the design is specified. All dialogues are by way of the graphical interface provided by the Smalltalk system. These features are particularly attractive for our application.

The Smalltalk implementation of our hierarchical timing simulation model, called HITSIM (HIerarchical Timing SIMulator), is presented in this chapter [1]. The specification of leaf cells and composition cells is presented in section 6.1 and 6.2. Instance creation, simulation and data abstraction of a cell are discussed in section 6.3–6.5. The UPE circuit of section 5.5 is used as an example during the discussion of these five sections. All the program segments presented in this chapter are in literal Smalltalk code.

6.1 Specification of a Leaf Cell

"Object," "Class" and "Instance" are the three major concepts in Smalltalk. All information in the Smalltalk system is represented as an object. Objects that respond to

[1] The particular suitability of Smalltalk for an embedded behavior-level simulation environment was suggested by Prof. Marina Chen of Yale University. Some preliminary Smalltalk simulation experiments were performed in collaboration with Dr. Chen on "Dolphin" hardware at Xerox, Pasadena, California, courtesy Kerry Laprade, Robert Lansford and Don Stewart. The simulator presented in this chapter was implemented on the experimental "Magnonia" workstation of Tektronix (Beaverton, Oregon) which was generously provided by Ward Cunningham and Kit Bradley.

the same messages in the same way can share the same generic definition. The generic definition is called a class. Objects generated from this definition are called instances of the class.

In structured VLSI design, a cell (or a family of cells) is often specified once, and gets instantiated in several different places. Using the Smalltalk terminology, specification of a cell corresponds to setting up a class, and the actual instantiations of the cell correspond to creating instances of the class.

Corresponding to the two kinds of cells in our hierarchical simulation model, there are two predefined classes in HITSIM: "LeafCell" and "CompositionCell." These two classes contain methods (functions) to transform cell specifications provided by the user into suitable classes and methods for performing timing simulation. All leaf cells specified by the user will become a subclass of the class "LeafCell"; similarly, for composition cells.

A leaf cell can be specified by sending the following message to the class LeafCell.

```
name: #<aString>
inputs: #( one or more <inputSpec>'s )
outputs: #( one or more <outputSpec>'s )
states: #( zero, one or more <stateSpec>'s )
behavior: ' <Smalltalk code> '.                                (6.1)
```

<aString> of (6.1) specifies the name of the leaf cell. <inputSpec>'s, <outputSpec>'s, and <stateSpec>'s specify the name and other attributes of the input ports, output ports, and internal states of the leaf cell:

1. One <inputSpec> corresponds to each input port, and consists of two items: the name and the loading capacitance of the input port. The loading capacitance is either a single value, if there are no pass transistors in the input network, or a pair of values, if there are. In the latter case, the first value corresponds to the input capacitance that is directly connected to the input port (no pass transistors in between), and the second value corresponds to that in the rest of the input network ($C_{i,1}$ and $C_{i,2}$ of

Figure 5.7). For instance, the capacitance pair of input $\overline{\text{B}}$ of cell $\overline{\text{MUX}}$ (Figure 5.10) is (10fF, 153fF). 10fF is the input capacitance corresponding to phase 2 and 153fF is that corresponding to phase 1.

2. One <outputSpec> corresponds to each output port, and consists of three items: the name of the output port, and two values of driving resistance: the first used for $1 \rightarrow 0$ transitions and the second used for $0 \rightarrow 1$ transitions.

3. One <stateSpec> corresponds to one state variable, and consists of only one item: the name of the variable.

<Smalltalk code> in (6.1) is a text of Smalltalk source code for describing the logic and timing behavior of the leaf cell: a mapping from the input states, current internal states and input timings to the output states, next internal states and output timings of the cell. Any construct of the Smalltalk language can be used in this text. If auxiliary variables are needed for the computation, they can be declared here.

Referring to Table 5.5 and Figure 5.11 of section 5.5, the following message specifies the logic and timing of cell $\overline{\text{XOR}}$.

```
LeafCell name: #XorBar
         inputs: #( (x 0) (s 0) )
         outputs: #( (sumBar 72 12) )
         states: #( )
         behavior: 'sumBar ← x eqv: s.
                    TsumBar ← sumBar ifTrue:[PhyTime + 2.2]
                                     ifFalse:[PhyTime + 0.4].'          (6.2)
```

Upon receiving the above message, class "LeafCell" creates a subclass of its own, called "XorBar," with the following four instance variables: "x," "s" and "sumBar" in which the logic values of the corresponding input and output ports can be stored; "TsumBar" stores the timing of output "sumBar." The prefix "T" to the name of an output port is a convention adopted by HITSIM to associate timing with the output port. In general, if a cell contains m inputs, n outputs and l internal states, then $m + 2n + l$ instance variables are created (this statement will be modified later). These variables are referred to in the behavior section of message (6.2). Input timings are not always required to specify the logic and timing of a cell. For instance, the output timing of cell $\overline{\mathrm{XOR}}$ depends only on the time when the cell is excited (The physical time, "PhyTime," a global variable detailed in section 6.3), so that no additional variables are created for individual input ports. In case the timing of a particular input port is important in the specification of a cell, it must be declared explicitly as a state variable.

After class "XorBar" is created with associated instance variables, the behavior section of the message is passed to the Smalltalk compiler, which returns a "compiledMethod" under message heading "cellChanged." Again, this heading is a convention used in HIT-SIM. This compiledMethod will be executed every time the message "cellChanged" is sent to an instance of the class. Note that users always interpret the data types of instance variables in their own way. For cell $\overline{\mathrm{XOR}}$, all input and output states are of type Boolean. For other cells, other data types (integer for instance) may be used. As to delay parameters, real numbers are used in this section (units: KΩ for resistances, fF for capacitances and ns for timing) for descriptive purposes. For performance reasons, however, integers are used in the real implementation with units properly adjusted.

Implied in message (6.2) is that, whenever an input of cell $\overline{\mathrm{XOR}}$ changes state, the same code (cellChanged) is executed. In many practical circuits, however, different actions may be required when different inputs change state. To specify the behavior of such a cell, the following message is used:

```
LeafCell name: #<aString>
         inputs: #( one or more <inputSpec>'s )
         outputs: #( one or more <outputSpec>'s )
         states: #( zero, one or more <stateSpec>'s ).          (6.3)
```

Since the behavior parameter is missing in the message[2], instead of creating a method under heading "cellChanged," message (6.3) creates a number of methods under headings determined by the names of the input ports. The contents of these methods are empty and must be filled by the user. For example, cell $\overline{\text{MUX}}$ of section 5.5 is specified as follows:

```
LeafCell name: #MuxBar
         inputs: #( (Phil 100) (Lsb 50) (M (10 140))
                    (Bbar (10 153)) (Dbar (10 183)) )
         outputs: #( (s 96 6) )
         states: #(bb dd Tbb Tdd Tm Tmbar).                    (6.4)
```

In response to the above message, a class called "MuxBar" is created with the following five methods: "PhilChanged," "LsbChanged," "MChanged," 'BbarChanged" and "DbarChanged." The suffix "Changed" to an input name is a convention adopted by HIT-SIM to associate an input-specific action with the corresponding input port. If, during simulation, the state of an input switches at certain instant of time, then the corresponding method is executed.

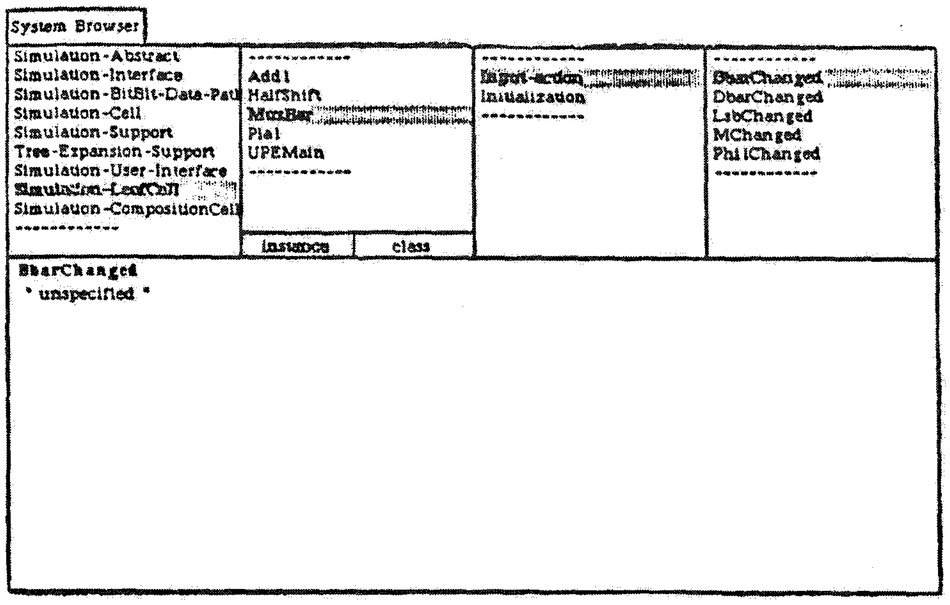
Another point to be noticed from (6.4) is that there are two capacitances associated with each of the following three inputs of cell "MuxBar": M, Bbar and Dbar (there are pass transistors associated with these ports). According to the rule we had before, there would be thirteen variables created for class "MuxBar": Phil, Lsb, M, Bbar, Dbar, s, Ts, bb, dd, Tbb, Tdd, Tm, Tmbar. In fact, six more variables are created, and the total number is nineteen. These six variables are: RM, FM, RBbar, FBbar, RDbar and FDbar. The reason is as follows. Suppose there are no pass transistors in an input network, then the delay of the net is totally independent of the delay inside the cell. However, if there are pass transistors in the input network, then these two delays are not independent, and the communication is established by way of these variables (see the method defined in the

[2] Message (6.3) contains four parameters and is different from message (6.1) which contains five parameters. These two messages are specified independently.

following). Note that two variables are associated with every such input port: one for rising and the other for falling of signals; the convention is to add prefix "R" or "F" to the input name. As a result, the actual number of instance variables for a cell that contains m inputs, k of which contain pass transistors in the input network, n outputs, and l states is $m + 2k + 2n + l$.

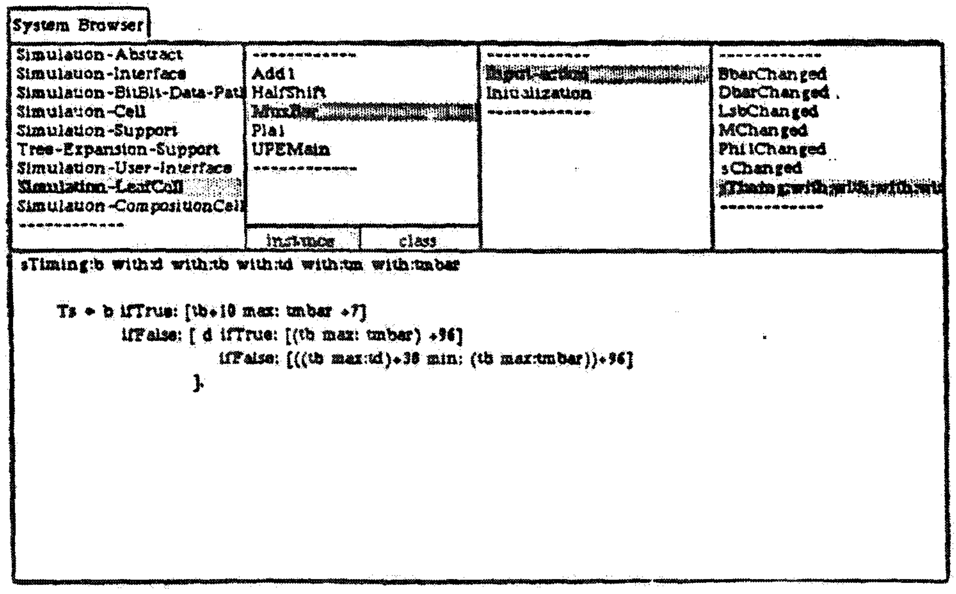
After class "MuxBar" is created, one can use the Smalltalk System Browser to enter the contents of these five input-specific methods (with the sequence of menu selections indicated in Figure 6.1.a) [18]. Referring to (5.6)–(5.11) and Figure 5.10 for the behavior of cell $\overline{\text{MUX}}$, the contents of these five methods are defined as follows:

```
MChanged
  "When this method is executed, "PhyTime" equals max(TM,Tφ1)"
  Phi1 ifTrue:[M ifTrue:[Tm ←PhyTime+RM+0.9. Tmbar ←PhyTime+RM+2.4]
                 ifFalse:[Tm ←PhyTime+FM+0.9. Tmbar ←PhyTime+FM+6.7].
                 self sChanged]            "sChanged is defined later"

BbarChanged
  "When this method is executed, "PhyTime" equals max(TBbar,Tφ1,TLsb)"
  (Lsb and:[Phi1 and:[(Bbar=bb) not]])
  ifTrue:[bb ←Bbar.
        Tbb ←PhyTime+(bb ifTrue:[RBbar] ifFalse:[FBar])+1.8.
        self sChanged].

DbarChanged
  "When this method is executed, "PhyTime" equals max(TDbar,Tφ1,TLsb)"
  (Lsb and:[Phi1 and:[(Dbar=dd) not]])
  ifTrue:[dd ←Dbar.
        Tdd ←PhyTime+(dd ifTrue:[RDbar] ifFalse[FDbar])+2.2.
        self sChanged].

LsbChanged
    (Lsb and:[Phi1]) ifTrue:[self BbarChanged; DbarChanged]


PhilChanged
    Phi1 ifTrue:[self MChanged; LsbChanged].                      (6.5)
```



```
BbarChanged
  " unspecified "
```

(a)



```
sTiming:b with:d with:tb with:td with:tm with:tmbar

    Ts ← b ifTrue: [tb+10 max: tmbar +7]
          ifFalse: [ d ifTrue: [(tb max: tmbar) +96]
                        ifFalse: [((tb max:td)+38 min: (tb max:tmbar))+96]
                   ].
```

(b)

Figure 6.1. Using Smalltalk System Browser for specifying a leaf cell

Note that clocks are not treated as special signals in HITSIM, although they can be made special by the user in the specification of the methods of a leaf cell. For instance, the specification of cell $\overline{\text{MUX}}$ is such that nothing happens when signal Phi1 is low.

The values of RM, FM, RBar, FBbar, RDbar and FDbar are determined when the composition environments of the corresponding input ports are determined (section 6.3). The five methods above refer to message "sChanged" which is defined in two steps as follows:

```
sChanged
     M ifTrue:[s ←bb not.
               self sTiming:bb with:dd with:Tbb with:Tdd with:Tm with:Tmbar]
        ifFalse:[s ←dd not.
               self sTiming:dd with:bb with:Tdd with:Tbb with:Tmbar with:Tm]
sTiming:b with:d with:tb with:td with:tm with:tmbar
     Ts ← b ifTrue:[tb+1.0 max: tmbar+0.7]
            ifFalse:[d ifTrue:[(tb max:tmbar)+9.6]
                         ifFalse:[((tb max: td)+3.8 min: (tb max:tmbar))+9.6]].
```

At the end of the above dialogue, the System Browser becomes the one shown in Figure 6.1.b. Note that the user can always take advantage of the Smalltalk graphical interface for specifying and modifying a cell description. For instance, the behavior section of message (6.1) can be entered or modified after a cell class is created. State variables can also be added or deleted as methods are added, modified or deleted for the cell class.

6.2 Specification of a Composition Cell

A composition cell is specified by sending the following message to the class CompositionCell:

```
name: #<aString>
inputs: #( one or more <inputPortSpec>'s )
outputs: #(one or more <outputPortSpec>'s )
```

```
subcells: #( one or more <subcellSpec>'s )
connections: #( one or more <connectionSpec>'s ).                    (6.6)
```

Every <inputPortSpec> or <outputPortSpec> of (6.6) corresponds to an input or output port of the composition cell, and consists of only one item: the name of the port. The loading capacitances and driving resistances need not be specified because these values are all implicit in the connection list. Also, there are no explicit internal states of a composition cell.

Every <subcellSpec> of (6.6) corresponds to a subcell of the composition cell, and consists of two items: the instance name and the class name of the subcell. A subcell is either a leaf cell or a composition cell. Every <connectionSpec> of (6.6) corresponds to an interconnection net, and consists of two items: the name of the driving node, and a tree structure describing the topology and physical parameters of the net. The driving node of a net is either an output port of a subcell, or an input port of the composition cell. Every entry in the tree structure corresponds to one branch of the interconnection tree, and consists of four items: the capacitance and resistance of the branch, the subtree that loads the branch (<child>), and a branch fanned out from the same branch as this branch (<sibling>). The BNF of <connectionSpec> is given in (6.7), and any net can be described using this construct.

```
<connectionSpec> ::= ( <source> <treeEntry> )
<treeEntry> ::= ( <capacitance> <resistance> <child> <sibling> )
<sibling> ::= <treeEntry> | nil
<child> ::= <treeEntry> | <destination>
<source> ::= ( <subcellName> <outputName> ) | <inputPortName>
<destination> ::= ( <subcellName> <inputName> ) | <outputPortName>       (6.7)
```

Note that <outputName> of (6.7) indicates an output port of subcell <subcellName>, while <outputPortName> indicates an output port of the composition cell. Similarly for <inputName> and <inputPortName>.

As an example, one UPE stage (Figure 5.9) is specified in the following as a composition cell consisting of one instance of cell "UPEMain" and two half-bit shift registers which are instances of the same cell: "HalfShift." Cell "UPEMain" contains cell MUX_ADD and the shift registers at the bottom of Figure 5.9. Cell "HalfShift" contains two inputs: "in" and "phi," and one output: "out." The specification of these two cells (classes) is given in section 6.7.

```
CompositionCell name: #UPEStage
        inputs: #( Phi1 Phi2 Ain Lsb M Bbar Dbar )
        outputs: #( Aout Lsbout Mout Bout Dout )
        subcells: #( (Cm UPEMain) (Cc HalfShift) (Cs HalfShift) )
        connections: #(  (Phi1 (0 10 (Cm Phi1)) )
              ( Phi2 (0.1 10 (Cm Phi2) (0.3 30 (Cs phi) (0.1 10 (Cc phi)))) )
              ( (Cm carryBar) (0 0 (Cc in)) )  ( (Cc out) (0 0 (Cm Cin)) )
              ( (Cm sumBar) (0 0 (Cs in)) )  ( (Cs out) (0 0 Aout) )
              ( Ain (0 0 (Cm Ain)) )
              ( Lsb (0 0 (Cm Lsb)) )  ( (Cm Lsbout) (0 0 Lsbout) )
              ( M (0 0 (Cm M)) )  ( (Cm Mout) (0 0 Mout) )
              ( Bbar (0 0 (Cm Bbar)) )  ( (Cm Bout) (0 0 Bout) )
              ( Dbar (0 0 (Cm Dbar)) )  ( (Cm Dout) (0 0 Dout) ) ).        (6.8)
```

Upon receiving the above message, class CompositionCell creates a subclass of its own, called "UPEStage," with the following fifteen instance variables: "Phi1," "Phi2," "Ain," "Lsb," "M," "Bbar," "Dbar," "Aout," "Lsbout," "Mout," "Bout," and "Dout" store the logic values of the input and output ports; "Cm," "Cc" and "Cs" refer to the three subcells of the cell. These fifteen variables will be created for every instance of the class "UPEStage."

Most of the resistances and capacitances of the nets are equal to 0, because the connections of adjacent stages of UPE are established simply by abutment. However, the delays through these nets are nonzero because of the output resistances and input capacitances of the connection ports. The structure of net (Phi2) is indicated in Figure 6.4

of section 6.3.

As another example, a two-stage UPE is specified as a composition cell consisting of two instances of class "UPEStage" defined above.

```
CompositionCell name:#UPE2
  inputs: #( Phi1 Phi2 Ain Lsb M Bbar Dbar )
  outputs: #( Aout Lsbout Mout Bout Dout )
  subcells: #( (C1 UPEStage) (C2 UPEStage) )
  connections: #( (Phi1 (0 60 (C1 Phi1) (0 140 (C2 Phi1))) )
    (Phi2 (0 120 (C1 Phi2) (0 200 (C2 Phi2))) )
    ( Ain (0 0 (C1 Ain)) ) ( (C1 Aout) (0 0 (C2 Ain)) ) ( (C2 Aout) (0 0 Aout) )
    ( Lsb (0 0 (C1 Lsb)) ) ( (C1 Lsbout) (0 0 (C2 Lsb)) ) ( (C2 Lsbout) (0 0 Lsbout) )
    ( M (0 0 (C1 M)) ) ( (C1 Mout) (0 0 (C2 M)) ) ( (C2 Mout) (0 0 Mout) )
    ( Bbar (0 0 (C1 Bbar)) ) ( (C1 Bout) (0 0 (C2 Bbar)) ) ( (C2 Bout) (0 0 Bout) )
    ( Dbar (0 0 (C1 Dbar)) ) ( (C1 Dout) (0 0 (C2 Dbar)) ) ( (C2 Dout) (0 0 Dout) ) )
                                                                              (6.9)
```

One can easily write a method that takes a parameter *n*, and generates the specification for an *n*-stage UPE.

6.3 Static Structure of a Cell for Simulation

In HITSIM, simulation is always performed on a composition cell. This composition cell may be as complicated as an entire system consisting of several levels of hierarchy, or as simple as a composition cell that contains only one leaf cell. The input and output ports of the composition cell are referred to as the primary inputs and primary outputs. Before simulation is done, the message "instance:#<aName >" is sent to the composition cell, which in response performs the following tasks:

1. An instance of the cell together with all the subcells under its hierarchy is created. Take the UPE2 cell of section 6.2 as an example. C1 and C2 are instance variables of class "UPE2"; both correspond to class "UPEStage." Cm is an instance variable of "UPEStage," and corresponds to class "UPEMain." Therefore, when an instance

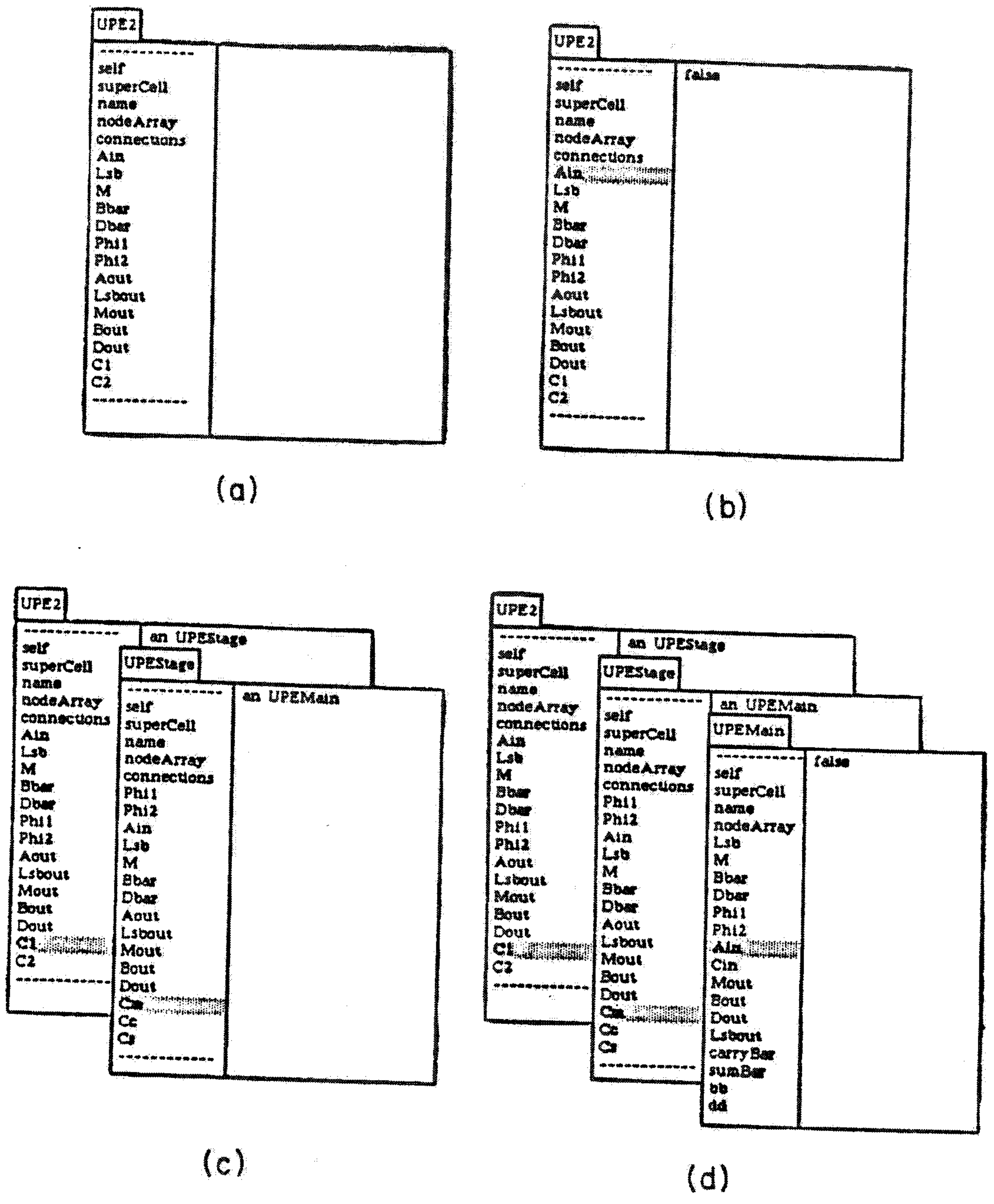


Figure 6.2. Using Smalltalk Inspector to investigate a cell hierarchy

of UPE2 is created, two instances of "UPEStage" are also created, and stored in the C1 and C2 variables of the created instance of UPE2. Recursively, two instances of "UPEMain" are created, and stored in the Cm variable of C1 and C2, respectively. The Smalltalk Inspector can be used to investigate the internal states of a composition cell in a hierarchical manner [18]. For instance, statement " (UPE2 instance:#foo) inspect" creates a window of the form shown in Figure 6.2.a. There are nineteen items (variables) appearing in the left-hand side of the window, among which the last fourteen are cell-specific instance variables (ports, states, timings, etc). The first five items, on the other hand, are cell-generic instance variables:

- "self" points to the cell itself;
- "superCell" points to the super cell of the cell;
- "name" stores the name of the cell;
- "nodeArray" stores various attributes of the ports of the cell described below;
- "connectionArray" stores the connection list of the cell. This information is only associated with composition cells.

Note that the format of the inspector is user-programmable, and that of Figure 6.2.a is the default one provided by the Smalltalk system.

When one of the variable names is selected, a description of the variable is printed in the right-hand side of the window. If the selected variable is a node of the cell, then the state of the node is printed (Figure 6.2.b). The state of the node can also be modified at this point. If the selected variable is a subcell, then the class name of the subcell is printed (Figure 6.2.c). The content of the subcell itself can be inspected by selecting menu item (a pop-up menu) "inspect." Repeating this process, the internal states and structures of a cell can be manipulated in a hierarchical manner (Figure 6.2.d).

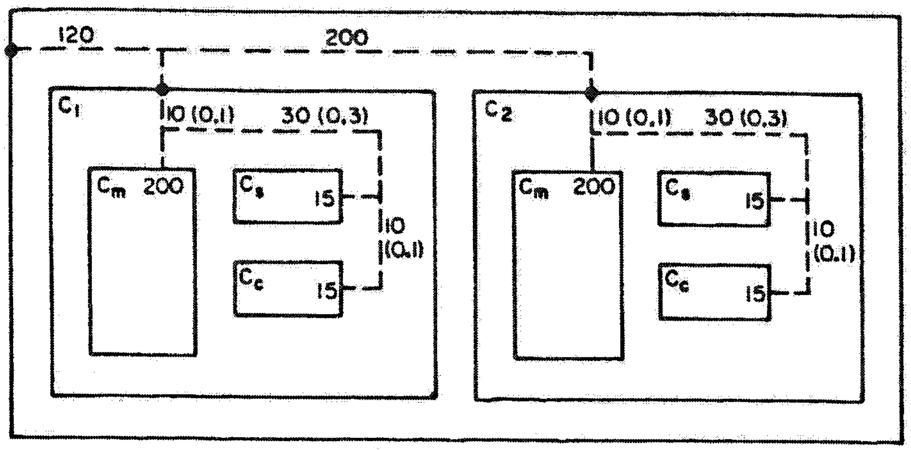


Figure 6.3. Net (Phi2) spans over two levels of composition cells

2. All the nets that span over more than one composition level are flatterned, and proper pointers among nodes in the same net are established. Take net (Phi2) of cell UPE2 as an example (Figure 6.3). This net spans over two composition levels, and each level contains only partial information about the net. There are nine nodes in the net:

   - top level: (Phi2)
   - second level: (C1 Phi2), (C2 Phi2)
   - leaf-cell level: (C1 Cm Phi2), (C1 Cs Phi2), (C1 Cc Phi2),
     (C2 Cm Phi2), (C2 Cs Phi2), (C2 Cc Phi2).

   (Phi2) in the above listing refers to the Phi2 port of the top level composition cell "UPE2"; (C1 Phi2) refers to the Phi2 port of the C1 subcell of UPE2; (C1 Cm Phi2) refers to the Phi2 port of the Cm subcell of the C1 subcell of UPE2; etc. This notation for accessing a node through the cell hierarchy will be used in this chapter.

   There are two nodes in the intermediate-level composition cells: (C1 Phi2) and (C2 Phi2). Through these two nodes, various segments of the net are merged into one tree, from which the delays between the driving node (Phi2) and the loading nodes (the six nodes at the leaf cell level) are calculated using the TREE algorithm. The

	(Phi2)	(C1 Phi2)	(C1 Cm Phi2)	(C1 Cs phi)	(C1 Cc phi)
Loading Capacitance	880	760	270	40	15
Rising Delay	1.76	1.76	1.79*	1.79*	1.79*
Falling Delay	0.88	0.88	0.91*	0.91*	0.91*

	(C2 Phi2)	(C2 Cm Phi2)	(C2 Cs phi)	(C2 Cc phi)
Loading Capacitance	280	270	40	15
Rising Delay	1.76	1.79*	1.79*	1.79*
Falling Delay	0.88	0.91*	0.91*	0.91*

Table 6.1. Delay calculation of net (Phi2)

loading capacitances of the loading nodes and the driving resistance of the driving node are also used in the calculation. For a connection port of a leaf cell, the driving resistance or loading capacitance is specified when the cell is specified (section 6.1). If there are two capacitances specified for an input port, then the first value is used. The driving resistance to a primary input port or the loading capacitance to a primary output port can also be specified by the user (default value is zero). Assuming that the driving resistance to the primary input (Phi2) is 2KΩ for rising and 1KΩ for falling transitions, Table 6.1 lists the total load capacitances and delays of the nine nodes in the net. Those items that are marked with "*"'s are recorded after the calculation; the other items are for temporary usage only.

During the net-flattening process, pointers are established from the driving node to the loading nodes of a net with calculated delay values. When the state of the driving node switches, these pointers and delay values are used to excite the loading nodes at proper points in time. Note that, in addition to the primary inputs and primary outputs, only leaf cells (and their ports) are involved in the simulation process. No overhead is spent on travelling through the intermediate-level composition cells. On the other hand, pointers are stored in the ports of intermediate-level composition cells, so that the design can be manipulated in a hierarchical manner.

In general, a port of a cell at any level of hierarchy is classified into one of the following three categories: driving end of a net, loading end of a net, or port of an intermediate-level composition cell. The driving end of a net is either an output port of a leaf cell or a primary input; a loading end of a net is either an input port of a leaf cell or a primary output. With a node in each of the three categories is associated the following data set:

1. With the driving end of a net (referred to as a "driving node") is associated the following data set:

   a. a pointer to the cell that contains the node and an index for accessing the related attributes of the node that are stored in the symbol table (instance variables) of the cell. The most important attributes are the "state" and the "timing" of the node. Note that what is stored in the "state" variable is actually the "target state" of the node, while the "actual state" is stored elsewhere (item b below). The reason is as follows. Suppose the driving node is an output port of a leaf cell. The "state" variable is updated immediately (and implicitly) when an input of the cell switches and some methods of the cell are executed. In reality, however, the state of the node should not be updated until after certain amount of delay (determined by the output timing). For a primary input, the target state is set by the user.

   b. the "actual state" of the node.

   c. a set of fan-out data, each of which corresponds to a loading end of the net, and contains three items: a pointer to the loading end, and two delay values for $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions, respectively.

   d. a pointer to the corresponding event if there is one scheduled for changing the state of the node later. This pointer is used to cancel or reschedule the event when necessary.

2. With every loading end of a net (referred to as a "loading node") is associated the following data set:

a. (if this node is an input port of a leaf cell) a pointer to the cell that contains the node and an index for accessing the related attributes of the node that are stored in the symbol table of the cell. The most important attributes are the "state" of the node and the "method" that needs to be executed when the state of the node changes. Unlike a driving node, the "state" variable stores the "actual state" of the node.

b. a pointer to the corresponding event if there is one scheduled for changing the state of the node later. This pointer is used to cancel or reschedule the event when necessary.

3. Stored in every port of an intermediate-level composition cell is a pointer that points to the "actual state" of the driving node of the net that contains the port. This pointer is used when the design is debugged in a hierarchical manner using the Smalltalk Inspector (Figure 6.2).

Recall that two capacitance values are specified for an input that contains pass transistors in the input network (section 6.1 (6.4)). Furthermore, two instance variables with prefix "R" and "F" are associated with such an input port. The values of these instance variables can also be determined using the TREE algorithm. Take the net that contains (C1 Bout), (C1 Cm Bout), (C2 Bbar) and (C2 Cm Bbar) of UPE2 as an example (Figure 6.4.a). The two input capacitances of (C2 Cm Bbar) are 10fF and 153fF, respectively. Referring to (5.5) of section 5.4, there are two pairs of delays associated with this net. The first delay pair (one rising and one falling) corresponds to the delay from (C1 Cm Bout) to (C2 Cm Bbar) when clock ϕ_1 is off (phase 2 delay). This delay pair is handled in exactly the same way as net (Phi2). The first input capacitance, i.e., 10fF, is used for the calculation (Figure 6.4.b). The second delay pair corresponds to the delay from (C2 Cm Bbar) to (C2 Cm bb) after ϕ_1 turns on, under the condition that (C2 Cm Bbar) is settled and Lsb is on initially (phase 1 delay). The values of this delay pair are calculated

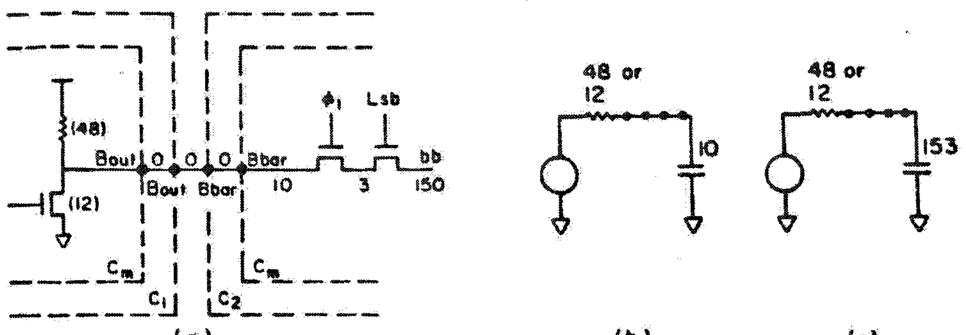


Figure 6.4. Net delay with pass transistors in an input network

based on the tree shown in Figure 6.4.c. The results are stored in the RBbar and FBbar instance variables of cell C2. Note that this technique is general enough to deal with any net of tree structure. Once the delay is calculated, its value never changes. This "static" delay is longer than the real delay in certain circumstances (it is always conservative), for instance, if a net drives two inputs both of which contain pass transistors in the input networks. To be conservative, each input network is assumed always to load the other network (unless the two are mutually exclusive). However, if during a certain clock period, one input port is active and the other is not, then the estimated delay to the first input is longer than the real value. One solution to this problem is to consider the net itself as a process which determines the delays between nodes dynamically (either structural or behavioral model can be used). However, the static approach is adequate for practical purposes. In particular, if there is only one input in the net, then the estimated delay is always correct (Corollary 5.4).

The usage of the two variables "RBbar" and "FBbar" (and four others) makes the specification of cell $\overline{\text{MUX}}$ completely independent of its composition environment. For different compositions of the cell, only the values of these variables are different; the specification of the cell is the same. On the other hand, if one is only interested in a particular composition environment of a cell, then these variables can be eliminated. For instance, the UPE is designed in such a way that the Bbar signal of one UPE stage is always fed from the Bout signal of the previous stage. The delay value of the net can be precalculated and hard-coded into the specification of cell $\overline{\text{MUX}}$.

6.4 Simulation of a Cell

HITSIM is an event-driven simulator. Associated with every event are the time to excite the event, the node to switch, and the target state of the node. When an event is scheduled, a pointer is established from the corresponding node to the event for possible cancellation or rescheduling of the event later. At most one event is associated with a given node. There are two global variables used during simulation: "EventQueue": a global queue in which all events are sorted by time, and "PhyTime": the physical time. The following pseudo-code indicates the main loop of the simulation process.

```
[ EventQueue isEmpty ] whileFalse:
  [ take the first event from the EventQueue;
    update the (global) physical time;
    invoke the event and put affected nodes into the EventQueue].
```

When an event is invoked, "PhyTime" is updated to the time associated with the event. Then, depending on whether the corresponding node of the event is a driving or a loading node, one of the following two actions is taken.

1. driving node: the "actual state" of the node is updated first. Then the loading nodes of the net driven by this node are checked one by one to see if there is an event pending.

   a. If there is no event pending for a given loading node, then an event is scheduled at time "PhyTime+T_{net}" with value equal to the state of the driving node. Depending on the type of transition (rising or falling), "T_{net}" is one of the two delay values precalculated and stored in the fan-out data.

b. If there is an event pending, then the state of the driving node is compared with the target state of the pending event. If these two values are different, then the pending event is cancelled, and the new event is scheduled. If these two values are the same, then "PhyTime+T_{net}" is compared with the excitation time of the pending event. Which one is smaller becomes the excitation time of the pending event. (6.10)

2. loading node: The value of the node is updated first. If the loading node is an input port of a leaf cell, then an appropriate method of the leaf cell is executed. This method is either the same for all inputs of the cell (method "cellChanged"), or is specific for the input port. At the end of this execution, the output ports of the cell are checked one by one to see if there is an event pending, and if the "target state" is the same as the "actual state" of the port. Depending on the result of the checking, one of the following four actions is taken:

   a. the target state is the same as the actual state of the output, and there is no event pending: nothing needs to be done in this case.

   b. the target state is different from the actual state, and there is no event pending: schedule an event with excitation time equal to the output timing and value equal to the target state of the output.

   c. the target state is the same as the actual state, and there is an event pending: cancel the event.

   d. the target state is different from the actual state, and there is an event pending: check to see if the target state is the same as the value of the pending event. If the two values are different, then the pending event is cancelled, and a new event is scheduled in the same way as in case (b) above. On the other hand, if the two values are the same, then the timing of the output is compared with the excitation time of the pending event. Which one is smaller becomes the excitation time of the pending event. (6.11)

Take cell $\overline{\text{MUX}}$ (Figure 5.11, (6.4)) as an example. Suppose the initial states of nodes M, Lsb, bb, Bbar, dd and Dbar are 1, 0, 0, 0, 1, 1, and thus the output state s is 1. In this section, "0," "false" and "low" are used interchangeably, so are "1," "true" and "high." At the beginning of every simulation process (usually one simulation process corresponds to one clock period), a method called "resetTime" is executed for every leaf cell under the hierarchy of the top-level composition cell. This method is also user-definable; the default action is to reset all timing-related variables (Tbb Tdd Tm Tmbar) to zero. For cell $\overline{\text{MUX}}$, one more thing is done: to reset clock Phil to low.

Suppose, at the current clock period, the following events switch the corresponding input states and excite the cell successively. Refer to (6.4) of section 6.1 for the behavior of $\overline{\text{MUX}}$, and Table 6.2 of section 6.7 for the values of the net-dependent instance variables.

time	0.5	1.0	1.5	2.0	3.0
input	Bbar	Lsb	Phil	M	Dbar
target state	1	1	1	0	1

1. Bbar switches high at time 0.5: Method "BbarChanged" is executed. Since Phil is low at this moment, no instance variable other than Bbar is modified.
2. Lsb switches high at time 1.0: Method "LsbChanged" is executed, which in turn invokes methods "BbarChanged" and "DbarChanged." Since Phil is low at this moment, nothing more happens.
3. Phil switches high at time 1.5: Method "PhilChanged" is executed, which in turn invokes methods "LsbChanged" and "MChanged."

   3.1. LsbChanged: Methods "BbarChanged" and "DbarChanged" are invoked in turn.

   - BbarChanged: bb is set to high and Tbb is evaluated to be 10.6 (1.5+7.3+1.8). Then, in method "sChanged," s is set to low, and Ts is evaluated to be 11.6 (10.6+1.0). As a result, an event is scheduled to switch the "actual state" of output s to low at time 11.6 (case (b) of (6.11)). Note that the instance variable s (the target state) is set to low already.

- DbarChanged: dd is the same as Dbar, so nothing happens.

3.2. MChanged: Since M is high, Tm and Tmbar are evaluated to be 9.1 (1.5+6.7+0.9) and 10.6 (1.5+6.7+2.4), respectively. In method "sChanged," signal s is evaluated to be low, and Ts is evaluated to be 11.6 (case (d) of (6.11)). Nothing needs to be modified.

Suppose the order of execution of the above two methods are reversed (in the specification of method "PhilChanged"). Then what happens is the following:

3.1′ Mchanged: Tm and Tmbar are evaluated to be 9.1 and 10.6, respectively. Since bb has not been updated to high yet, so s is evaluated to be high (case (a) of (6.11)). Nothing needs to be done.

3.2′ LsbChanged: the same as item 3.1 above.

Although the efficiency may differ, the result is independent of the order of execution of the two methods. Likewise, if several events are scheduled at the same physical time, the result is the same regardless of the order of excitation of these events. Note that the above statement is only true for semantic cells of disciplined systems, and is not true in general [9].

4. M switches low at time 2.0: Method "MChanged" is invoked, in which Tm is evaluated to be 4.8 (2.0+1.7+0.9) and Tmbar is evaluated to be 10.4 (2.0+1.7+6.7). In method "sChanged," s is evaluated to be high, and Ts is evaluated to be 20 (10.4+9.6) (case (c) of (6.11)). The pending event scheduled at item 3.1 above is cancelled.

5. Dbar switches to high at time 3.0: Method "DbarChanged" is invoked, in which dd is set to high, and Tdd is evaluated to be 14 (3.0+8.8+2.2). In method "sChanged," s is set to low, and Ts is evaluated to be 15.0 (14.0+1.0). An event is scheduled to

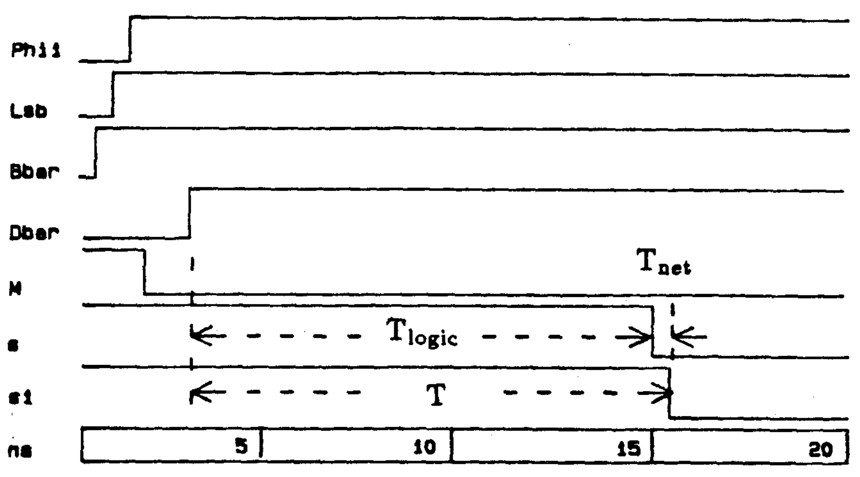


Figure 6.5. Simulation display of cell $\overline{\text{MUX}}$

switch the "actual state" of s low at time 15.0 (case (b) of (6.11)).

6. At time 15.0, the above event is excited. Note that output s fans out to the s1 input of cell $\overline{\text{MAJ}}$ and the s2 input of cell $\overline{\text{XOR}}$ (Figure 5.9). The net delay is the same for both inputs: 0.7ns when s falls, and 5.6ns when s rises (section 6.7). As a result, two events are scheduled to switch the s1 and s2 inputs to low at time 15.7 (case (a) of (6.10)).

Simulation results can be displayed in a number of ways. Because HITSIM is embedded in a programing language system, the user can always reconfigure the simulator in the most preferable way. In addition to text displays, the current implementation of HITSIM also supports waveform displays. Shown in Figure 6.5 is the waveform display of the above simulation process. The following remarks refer to Figure 6.5:

Remarks:

- Node s and node s1 refer to the same logic node. The waveforms of the two nodes are quite similar, except that the high-to-low transition of s1 happens 0.7ns later than that of s. The delay between the rising edge of Dbar and the falling edge of s1 (T of Figure 6.5) can be naturally divided into two segments: T_{logic} and T_{net}. T_{logic} is the internal delay of leaf cell $\overline{\text{MUX}}$, assuming output s is open; T_{net} is the delay due to the interconnection net. The waveform of s is not real; it sets a lower bound on the timing of node s1 under all possible interconnections.

- When Phi1 switches high at time 1.5, an event is scheduled to switch node s low at time 11.6. This event is later cancelled when M switches low at time 2.0. The two input events above do not produce a glitch at output s (at least, the present implementation of HITSIM does not indicate a glitch under these conditions). Usually, there are several stages of transistor groups between an input port and an output port of a behavior-level cell. A small glitch at the input port does not tend to produce a glitch at the output port. On the other hand, if node M switches high at time later than 11.6, then a glitch shows up in the waveform of s. In this sense, HITSIM implements a sophisticated form of "propagation delay" model.
- The termination condition for the above simulation process is that the event queue is empty. The "PhyTime" at this moment sets a lower bound on the period of clock ϕ_1[3]. Another possible termination condition for a simulation process when the maximum allowable clock period is specified is that "PhyTime" exceeds this specified value. All events left in the queue violate the specification, and can be investigated one by one when the simulation process terminates.
- It is possible to simulate an arbitrary number of clock cycles in one simulation process (Figure 6.6). This capability is essential when the borrowing-time technique is employed.

[3] Usually the reference time for a simulation process is selected to be the rising edge of a clock. For illustration purposes, however, the reference time of the above example is selected to be 1.5ns before clock ϕ_1 rises. Thus the actual bound is equal to "PhyTime−1.5."

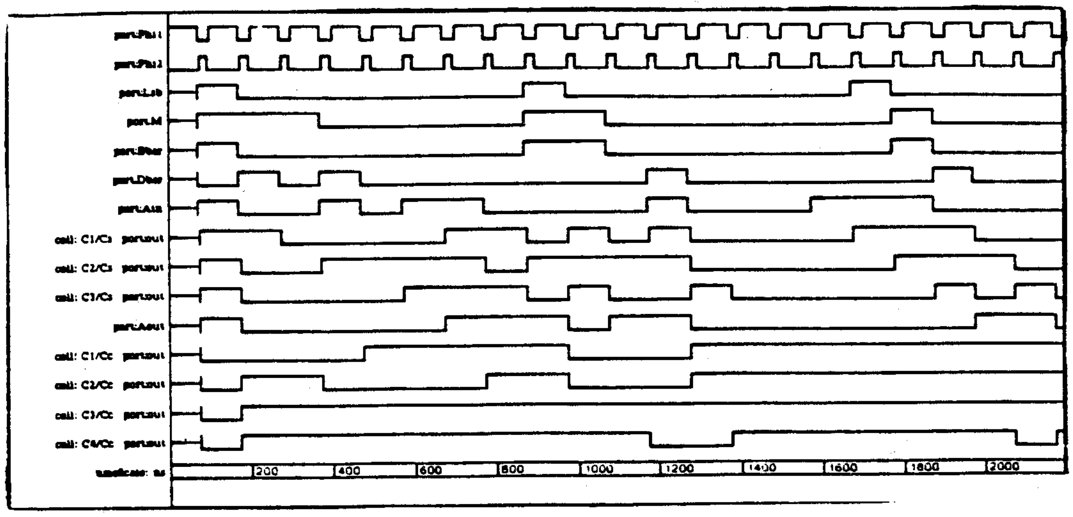


Figure 6.6. Simulation display of a 4-stage UPE (bit-level)

6.5 Data Abstraction

In addition to functional abstraction, our timing model also allows data abstraction. The data used in previous sections for describing the UPE are of type "Boolean." The timing discipline imposed at this level of representation (bit-level) is the "non-overlapping two phase clock."

For an n-stage UPE, $2n$ consecutive bits coming in and out of the UPE are interpreted as one unit[4]. As a result, a new data type "serial word" (integer) is used to describe the behavior of the UPE at the word level. The timing discipline imposed at this level of abstraction is the "data stationary control" [24] of input Lsb. Referring to Figure 6.6, input Lsb goes high every $2n$ clock cycles, synchronizing with the first bit (the least significant bit) of any serial word.

At the word level, the behavior of a 4-stage UPE is specified as follows:

LeafCell name: #UPE4
        inputs: #((Lsb 50) (M (10 140)) (Bbar (10 153))

[4] In Lyon's multiplier[29], n consecutive bits are interpreted as one unit.

```
            (Dbar (10 183)) (Ain (10 50)) )
outputs: #( (Lsbout 48 12) (Mout 48 12) (Bout 48 12)
            (Dout 48 12) (Aout 48 12) )
states: #( )
behavior: 'Lsbout ←Lsb.
            TLsbout ←PhyTime + (8*ClockPeriod).
            Lsb ifTrue:[Aout ← (Bbar bitInvert*M)+
                             (Dbar bitInvert*(M bitInvert))+Ain rem:256.
                    TAout ←TLsbout.
                    Bout ←Bbar. TBout ←TLsbout.
                    Dout ←Dbar. TDout ←TLsbout]'.                (6.12)
```

"ClockPeriod" of (6.12) is the period between two consecutive rising edges of clock ϕ_1. The complexity of one UPE stage is simple enough for the user to figure out the critical path of each clock phase. Thus the minimum allowable value of "ClockPeriod" can be determined by running HITSIM based on the bit-level representation. The loading capacitances and driving resistances specified in (6.12) are used to check if the value of "ClockPeriod" needs to be increased due to the delay of external interconnections. The consistency between the bit-level and word-level representations of the UPE can be established either by comparison of simulation results or by the formal fixed-point argument introduced by Chen [9].

For an n-stage UPE, the two numbers 8 and 256 appearing in the behavior parameter of (6.12) are replaced by $2 * n$ and 2^{2n}, respectively.

Shown in Figure 6.7.a is a simulation result of a 4-stage UPE. The value of "ClockPeriod" is conservatively set to 125ns. Shown in Figure 6.7.b is a second-order filter consisting of two 32-stage UPE's. Based on the word-level representation, the HITSIM simulation result is displayed as a plot of output "Aout" as a function of time (Figure 6.7.c). Note that output U is a truncated version of Aout, with sign bit extended [50]. When this filter is synthesized using the real UPE chip, the output waveform after a D/A converter looks exactly the same as Figure 6.7.c on a CRT screen.

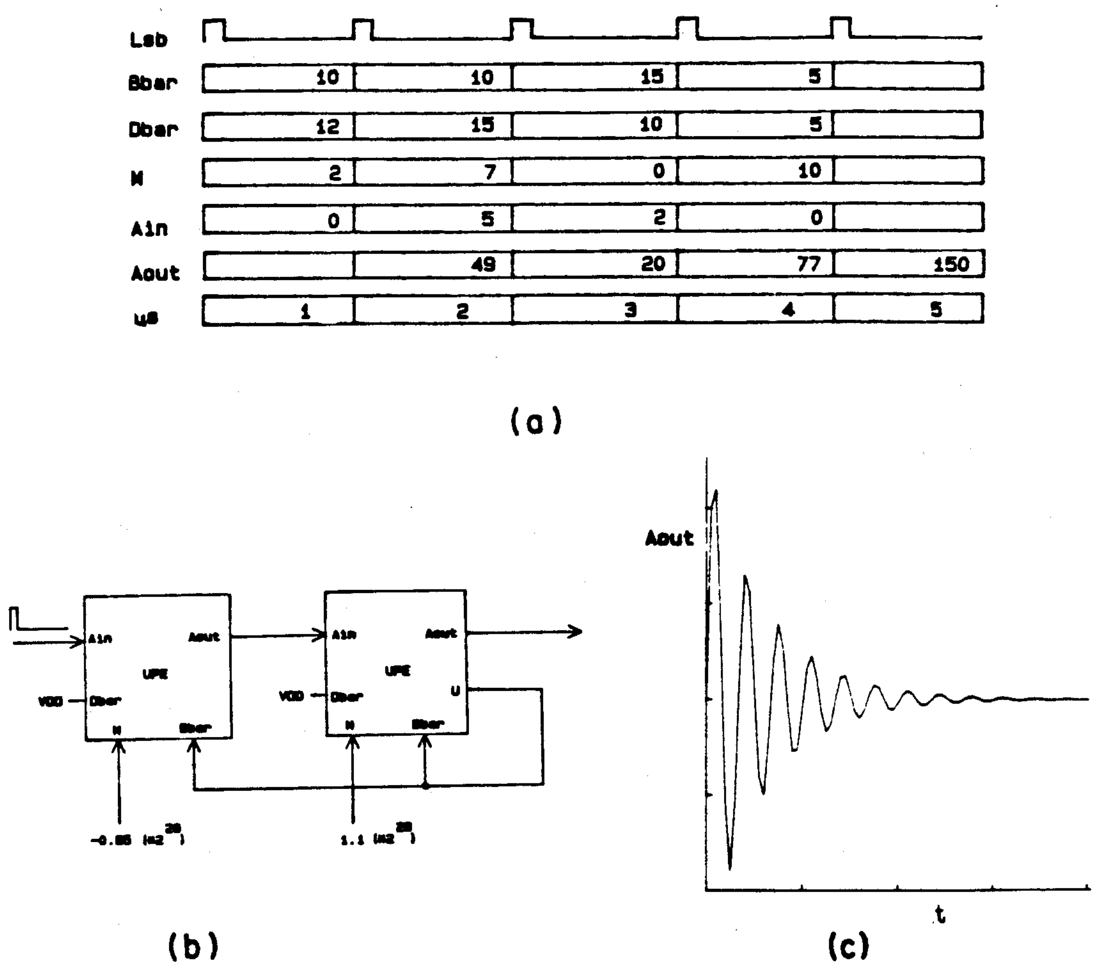


Figure 6.7. Simulation display of the UPE at the word level

6.6 Summary

A Smalltalk implementation of our hierarchical timing simulation model is presented in this chapter. The resulting simulator, called HITSIM, is based on "behavior" models of leaf cells and "structure" information of composition cells:

- Logic propagation: the logic relationship among the ports of a leaf cell is specified by the user. Any constructs and data types allowed in the Smalltalk language can be used. On the other hand, propagation of events from one cell to another is handled automatically by HITSIM. The connection information specified in the composition cells is used for this purpose.

- Delay calculation: The delays from the inputs to the outputs of a leaf cell are also specified by the user. Any timing or circuit simulators can be employed for obtaining this information. For composition purposes, the driving resistances of the output ports and loading capacitances of the input ports also need to be specified. On the other hand, delays between leaf cells are calculated automatically by using the TREE algorithm. Any interconnection nets of tree structure can be dealt with in a simple and general way.

6.7 Appendix: Logic and Timing of UPE

Referring to (6.8) of section 6.2, one stage of UPE is composed from the following three leaf cells: one instance of class "UPEMain" and two instances of class "HalfShift." The two classes "UPEMain" and "HalfShift" are specified as follows:

Cell UPEMain

```
LeafCell name: #UPEMain
         inputs: #( (Phi1 200) (Phi2 200) (Lsb 50) (M (10 140))
                   (Bbar (10 153) ) (Dbar (10 183)) (Ain (10 50)) (Cin (10 39)) )
         outputs: #( (Lsbout 48 12) (Mout 48 12) (Bout 48 12)
                   (Dout 48 12) (carryBar 48 12) (sumBar 72 12))
         states: #(bb dd Tbb Tdd Tm Tmbar x s Tx Ts lsb1 b1 d1).
```

The meaning of the thirteen state variables specified above are as follows:

- bb and dd are the internal states of cell $\overline{\text{MUX}}$;
- x and s are the outputs of cells XOR and $\overline{\text{MUX}}$, respectively;
- lsb1, b1 and d1 are the internal states of the shift registers;
- Tbb, Tdd, Tm, Tmbar, Tx and Ts are the timing of various nodes inside the circuit.

Although many of the state variables are not absolutely necessary (for instance, x and s are outputs of combinational circuits...), they are included for ease of specification. With the eight inputs of cell "UPEMain" are associated the following eight methods:

MChanged
```
Phil ifTrue:[M ifTrue: [Tm ←PhyTime+RM+0.9. Tmbar ←PhyTime+RM+2.4]
               ifFalse:[Tm ←PhyTime+FM+0.9. Tmbar ←PhyTime+FM+6.7].
             self sChanged]
```

BbarChanged
```
(Lsb and:[Phil and:[(Bbar=bb) not]])
ifTrue:[bb ←Bbar.
      Tbb ←PhyTime+(bb ifTrue:[RBbar] ifFalse:[FBar])+1.8.
      self sChanged]
```

DbarChanged
```
(Lsb and:[Phil and:[(Dbar=dd) not]])
ifTrue:[dd ←Dbar.
      Tdd ←PhyTime+(dd ifTrue:[RDbar] ifFalse[FDbar])+2.2.
      self sChanged].
```

LsbChanged
```
(Lsb and: [Phil]) ifTrue:[self BbarChanged; DbarChanged]
```

AinChanged
```
| ta tx |
Phil ifTrue:[x ←Ain xor:Cin.
            ta ←PhyTime+(Ain ifTrue:[RAin] ifFalse:[FAin])+0.3.
            "0.3 is the internal net delay from Ain to a1 (or a2) (Figure 5.11)"
            tx ←ta + (x ifTure:[9.8] ifFalse:[1.3]).   "cell XOR, Table 5.5"
            self carryBar:ta sumBar:tx]
```

CinChanged:
```
| tc tx |
Phil ifTrue:[x ←Ain xor:Cin.
            tc PhyTime+(Cin ifTrue:[RCin] ifFalse:[FCin])+0.2.
            "0.2 is the internal net delay from Cin to c1 (or c2)"
            tx ←tc + (x ifTrue:[9.8] ifFalse:[1.3]).   "cell XOR, Table 5.5"
```

```
                    self carryBar:tc sumBar:tx]
```

Phi1Changed:

```
    Phi1 ifTrue:[self MChanged; LsbChanged; AinChanged; CinChanged]
```

Phi2Changed:

```
    | tout |
    tout ←PhyTime+1.4.
    Phi2 ifTrue:[Mout ←M. TMout ←tout.
                 Bout ←b1. b1 ←Bbar. TBout ←tout.
                 Dout ←d1. d1 ←Dbar. TDout ←tout.
                 Lsbout ←lsb1. lsb1 ←Lsb. TLsbout ←tout]
```

To complete the specification, the following two methods are defined.

sChanged

```
    | tsc |
    M ifTrue:[s ←bb not.
              self sTiming:bb with:dd with:Tbb with:Tdd with:Tm with:Tmbar]
              ifFalse:[s ←dd not.
              self sTiming:dd with:bb with:Tdd with:Tbb with:Tmbar with:Tm].
    tsc ← Ts + (s ifTrue:[5.6] ifFalse:[0.7]).
     "5.6 and 0.7 are due to the net delay from s to s1 (or s2)"
    self carryBar:tsc sumBar:tsc]
```

sTiming:b with:d with:tb with:td with:tm with:tmbar

```
    Ts ← b ifTrue:[tb+1.0 max: tmbar+0.7]
           ifFalse:[d ifTrue:[(tb max:tmbar)+9.6]
                      ifFalse:[((tb max: td)+3.8 min: (tb max:tmbar))+9.6]]
```

carryBar:tc sumBar:ts

```
    carryBar ← (x ifTrue:[s] ifFalse:[Ain]) not.
    TcarryBar ←tc + (carryBar ifTrue:[Ain ifTrue:[2.8]
                                         ifFalse:[Cin ifTrue:[3.4] ifFalse:[1.9]]]
                              ifFalse:[0.9]).     "cell MAJ, Table 5.4"
    sumBar ←x eqv:s.
    TsumBar ←ts + (sumBar ifTrue:[2.2] ifFalse:[0.4])   "cell XOR, Table 5.6"
```

There are ten net-dependent instance variables of cell "UPEMain." They are associated with the five input ports: "M," "Bbar," "Dbar," "Ain" and "Cin" with prefix "R" and "F," respectively. Under the composition environment of the UPE (Figure 6.4 and (6.9)), the values of these ten variables are listed in Table 6.2.

	M	Bbar	Dbar	Ain	Cin
R (rise)	6.7	7.3	8.8	2.4	1.9
F (fall)	1.7	1.8	2.2	0.6	0.5

Table 6.2. Net-dependent instance variables of one UPE stage

<u>Cell HalfShift</u>

```
LeafCell name: #HalfShift
         inputs: #( (in (5 20) ) (phi 15) )
         outputs: #( (out 48 12) )
         states: #( )
         behavior: ' phi ifTrue:[out ← in not.
                                 Tout ←PhyTime+ (in ifTrue:[Rin+0.1]
                                                    ifFalse:[Fin+0.5]) ]. '
```

The values of Rin and Fin under the composition environment of cell "UPEStage" (6.8) are 1.0 and 0.2, respectively.

Chapter 7

Conclusions

The model and simulation algorithms presented in this thesis were developed to cope with the increasing complexity of VLSI systems. The model supports both structure and behavior representations of designs in a uniform manner. At the structure level, time constant-related primitives (the R, C, D parameters) are used to characterize the integral behavior of digital MOS circuits. The simulation algorithms are much more efficient than other circuit and timing simulators in which lower-level primitives (currents and voltages) are used and differential behavior is considered. The composition rules of the R, C, D parameters are derived analytically from the Kirchoff's current and voltage laws so that the consistency with physics is established. At the behavior level, the logic and timing of a semantic cell can be abstracted from its implementation to allow hierarchical treatment of a complicated design. The intention is to balance the circuit-level and system-level considerations for developing a simulation model.

7.1 An Integrated Design System

Two experimental simulators have been implemented and presented in this thesis: SDS for the structure representation and HITSIM for the behavior representation of designs. These two simulators can be combined with other tools to form an integrated design system that fully supports the structured design methodology. The design flow of one such system currently under integration is indicated in Figure 7.1. The blocks bounded by bold lines are programs, and those bounded by regular solid lines are data sets. The blocks bounded by dash lines are tasks that are currently performed the user; possible automation of these tasks are discussed in section 7.2. In addition to SDS and HITSIM,

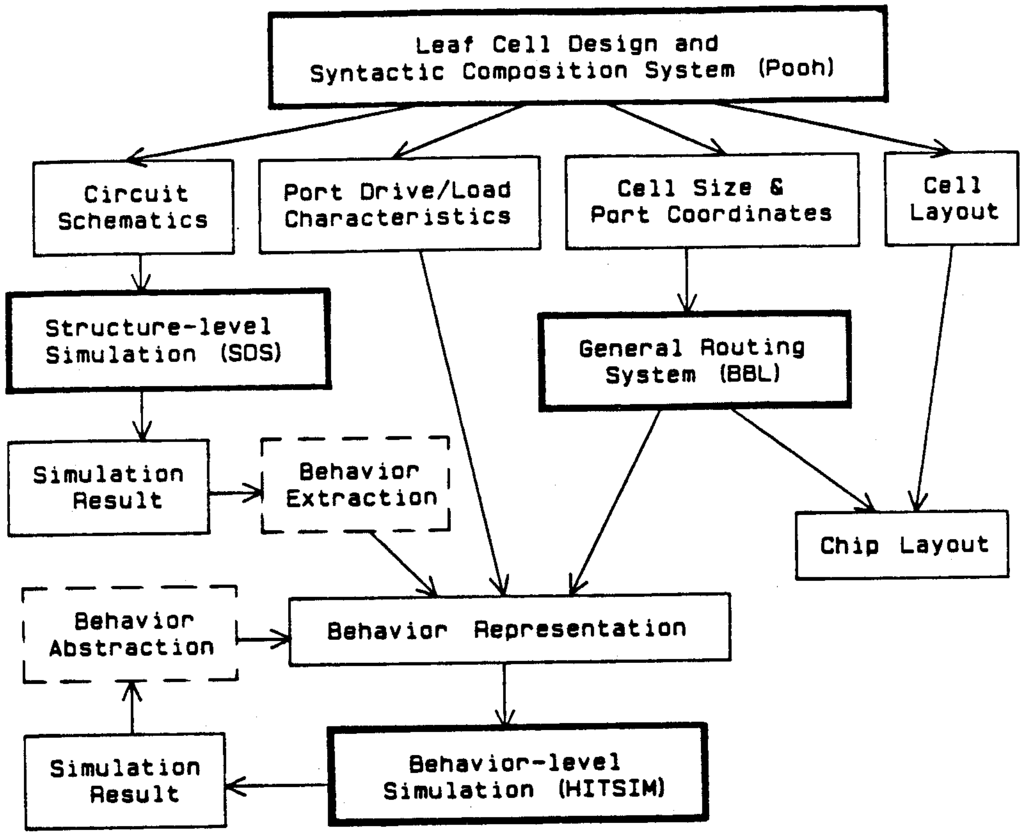

Leaf Cell Design and Syntactic Composition System (Pooh)
Circuit Schematics
Port Drive/Load Characteristics
Cell Size & Port Coordinates
Cell Layout
Structure-level Simulation (SOS)
General Routing System (BBL)
Simulation Result
Behavior Extraction
Chip Layout
Behavior Abstraction
Behavior Representation
Simulation Result
Behavior-level Simulation (HITSIM)

Figure 7.1. An integrated design system

two other programs used are the Pooh leaf cell design and syntactic composition system [1] developed by Whitney [52,53] and the BBL general routing system developed at UC Berkeley by Chen, Kuh, etc. [11]. These two systems are selected because they both fit in

[1] The Pooh system consists of a graphic editor, called Tigger [49], for (physical or syntactic) leaf cell design, and a syntactic composition system, called PoohComp, which supports both port fusion and river routing. To run hierarchical simulation, the result of a syntactic composition must be a semantic cell. This resulting cell, called a functional block in many design systems, is used as a (logical or semantic) leaf cell in the HITSIM simulator.

our general framework in a clean and natural way:

- The Pooh system manipulates and generates circuit schematics (listing of transistors and wires and their sizes) and design-rule-correct layouts based on a unified representation (called Pooh representation). This is quite a contrast to the traditional approach of extracting circuit schematics from physical layouts, a process that is not only timing consuming, but also incapable of determining the semantic boundaries within a circuit for performing hierarchical simulation.
- The BBL system handles arbitrarily shaped rectilinear blocks, minimizes layout area and assures 100% routing completion. This system also allows routing to be done in a hierarchical manner.

Given the specification of a target circuit, the user first determines the timing strategy, the set of leaf cells to be designed, and the composition hierarchy for building up the circuit. Every leaf cell can be designed using the Pooh system which, upon completion, generates the following four pieces of data: 1) the circuit schematics of the cell for performing SDS timing simulation, 2) the driving resistances and loading capacitances of the ports required for HITSIM simulation, 3) the size of the cell and the coordinates of the ports for performing BBL routing, and 4) the physical layout of the cell which will be combined with the BBL output to form the complete layout of the chip.

Based on the results of the SDS simulation, the user determines the behavior and timing of each cell. The behavior and timing descriptions of a collection of cells, together with the following two data, are used for performing HITSIM simulation: 1) the driving resistances and loading capacitances of the ports generated by the Pooh system, and 2) the tree structure and physical parameters of the connection wires generated by the BBL system. Note that both the behavior and timing specifications of cells and the routing data are maintained in a hierarchical manner. With proper functional and data abstraction of the specifications of the cells, the user can flatten the design at any desired level for performing the HITSIM simulation.

7.2 Suggestions for Future Research

The present implementation of HITSIM relies on the user for performing the following two tasks (Figure 7.1):

1. **Behavior Extraction:** Given the structure (transistor and wire listing) of a semantic cell, the behavior of the cell has to be derived manually.

2. **Behavior Abstraction:** Given the behaviors and net listing of a collection of semantic cells, the abstract behavior of the composite cell has to be derived manually.

In general, the first task (behavior extraction) cannot be fully automated [13]. However, by restricting the constructs that are allowed in the design, it is quite possible for computers to perform the task. The key issue is that the performance of the design should not be sacrificed too much by imposing these restrictions. Recently, Frey and Mead have developed a new logic family that satisfies all the desirable properties of VLSI designs: high speed, high density, low power, no analog hazard, etc [33]. Furthermore, the behavior of any design following the composition rules of this logic family is directly derivable from its circuit structure, the correctness of which is guaranteed. Our timing model matches with this logic family very nicely; the integration of the two seems a natural research project to pursue.

As to the second task (behavior abstraction), although the description of the composite cell can be trivially derived by symbolic manipulation of the descriptions of the component cells, no abstraction or optimization can be easily automated during this process. The task of abstraction seems better performed by humans rather than by machines. An interesting project would be to design a reasonable interface to aid the user performing this task. Certain properties such as the consistency between the two representations should be guaranteed by the support of the interface.

References

[1] Agrawal, V.D., "Synchronous Path Analysis in MOS Circuit Simulator," Proc. of the 19th Design Automation Conference, pp.629–635, 1982.

[2] Aho, A.V., Hopcroft, J.E. & Ullman, J.D., "The Design and Analysis of Computer Algorithms," Chapter 5, Addison-Wesley, 1974.

[3] Baker, C., "Artwork Analysis Tools for VLSI Circuits," MIT/LCS/TR-239, May 1980.

[4] Bryant, R.E., "A Switch-level Simulation Model for Integrated Logic Circuits," MIT/LCS/TR-259, Doctoral Dissertation, MIT, March 1981.

[5] Bryant, R.E., "A Switch-Level Model and Simulator for MOS Digital Systems," IEEE Trans. on Computers, vol. C-33, no.2, pp.160–177, Feb. 1984.

[6] Bryant, R.E., Schuster, M. & Whiting, D., "MOSSIM II: A Switch-Level Simulator for MOS LSI: User's Manual," 5033:TR:82, Computer Science, Caltech, 1982.

[7] Chawla, B.R., Gummel, H.K. & Kozak, P., "MOTIS — An MOS Timing Simulator," IEEE Trans. on Circuits and System, CAS-22, no.12, pp.901–910, Dec. 1975.

[8] Chen, M.C. & Mead, C.A., "A Hierarchical Simulator Based on Formal Semantics," Proc. of the 3rd Caltech Conference on VLSI, pp.207–223, March 1983.

[9] Chen, M.C., "Space-Time Algorithms: Semantics and Methodology," 5090:TR:83, Doctoral Dissertation, Computer Science, Caltech, May 1983.

[10] Chen, M.C. & Mead, C.A., "A Methodology for Hierarchical Simulation and Verification of VLSI Systems," in preparation.

[11] Chen, N-P., Hsu, C-P. & Kuh, E.S., "The Berkeley Building-Block Layout System for VLSI Design," Proc. of the IFIP TC 10/WG 10.5 International Conference on VLSI, pp.37-44, Trondheim, Norway, Aug. 1983.

[12] Chiang, C–L., "Distributed RC Delay Line Model and MOS PLA Timing Estima-

tion," 5010:DF:83, Computer Science, Caltech, Dec. 1983.

[13] Cutland, N., "Computability," Cambridge University Press, 1980.

[14] Desoer, C.A. & Kuh, E.S., "Basic Circuit Theory," Chapter 17, McGraw-Hill, 1969.

[15] Elmore, W.C., "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," Journal of Applied Physics, vol.19, no.1, pp.55–63, Jan. 1948.

[16] Ghausi, M.S. & Kelly, J.J., "Introduction to Distributed-Parameter Networks with Application to Integrated Circuits," Chapter 1, Holt, Rinehart and Winston Inc., 1968.

[17] Glasser, L.A., & Penfield, P., "VLSI Circuit Theory," Proc., of the IEEE International Large Scale Systems Symposium, pp.499–501, Oct. 1982.

[18] Goldberg, A., "Smalltalk-80, the Interactive Programming Environment," Addison-Wesley, 1984.

[19] Goldberg, A., & Roson, D., "Smalltalk-80, the Language and Its Implementation," Addison-Wesley, May 1983.

[20] Horowitz, M.A., "Timing Models for MOS Pass Networks," Proc. of the International Symposium on Circuits and Systems, pp.198–201, 1983.

[21] Horowitz, M.A., "Timing Models for MOS Circuits," SEL83-003, Doctoral Dissertation, Stanford Univ., Dec. 1983.

[22] Johannsen, D.L., "Silicon Compilation," TR:4530:81, Doctoral Dissertation, Computer Science, Caltech, 1981.

[23] Jouppi, N.P., "TV: An nMOS Timing Analyzer," Proc. of the 3rd Caltech Conference on VLSI, pp.71–86, March 1983.

[24] Kogge, P.M., "The Architecture of Pipelined Computers," McGraw-Hill, 1981.

[25] Kung, H–T., "Let's Design Algorithms for VLSI Systems," Proc. of the 1st Caltech

Conference on VLSI, pp.65–90, Jan. 1979.

[26] Lelarasmee, E., Ruehli, A.E. & Sangiovanni-Vincentelli, A.L., "The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits," IEEE Trans. on Computer-Aided Design of ICAS, vol. CAD-1, no.3, pp.131–145, July 1982.

[27] Lin, T–M. & Mead, C.A., "Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits," Proc. of the 3rd MIT Conference on Advanced Research in VLSI, pp.93–99, Jan. 1984.

[28] Lin, T–M. & Mead, C.A., "Signal Delay in General RC Networks," IEEE Trans. on Computer-Aided Design of ICAS, vol. CAD-3, no.4, Oct 1984.

[29] Lyon, R.F., "Two's Complement Pipeline Multipliers," IEEE Trans. on Communications, vol. COM-24, no.4, pp.418–425, April 1976.

[30] Mead, C.A. & Conway, L.A., "Introduction to VLSI Systems," Addison Wesley, 1980.

[31] Mead, C.A., Reference [30], Chapter 1.

[32] Mead, C.A., "Structural and Behavioral Composition of VLSI," Proc. of the IFIP TC 10/WG 10.5 International Conference on VLSI, pp.3–8, Trondheim, Norway, Aug. 1983.

[33] Mead, C.A. & Frey, A., "A Logic Family," unpublished report, Jan 1984.

[34] Moore, G.E., "Are We Really Ready for VLSI," Proc. of the 1st Caltech Conference on VLSI, pp.3–14, Jan. 1979.

[35] Murphy B.T., "Microcomputers: Trends, Technologies, and Design Strategies," IEEE Journal of Solid-State Circuits, vol. SC-18, no.3, pp.236–244, June 1983.

[36] Nagel, L.W., "SPICE2: A Computer Program to Simulate Semiconductor Circuits," ERL Memo, no. ERL-M520, Electronics Research Laboratory, University of California, Berkeley, 1975.

[37] Ousterhout, J.K., "Crystal: A Timing Analyzer for nMOS VLSI Circuits," Proc. of

the 3rd Caltech Conference on VLSI, pp.57–70, March 1983.

[38] Penfield, P. & Rubinstein, J., "Signal Delay in RC Tree Networks," Proc. of the 2nd Caltech Conference on VLSI, pp.269–283, March 1981

[39] Rem, M., & Mead, C.A., "A Notation for Designing Restoring Logic Circuitry in CMOS," Proc. of the 2nd Caltech Conference on VLSI, pp.399–411, Jan. 1981.

[40] Rubinstein, J., Penfield, P., & Horowitz, M., "Signal Delays in RC Tree Networks," IEEE Trans. on Computer-Aided Design of ICAS, vol. CAD-2, no.3, pp.202–211, July 1983.

[41] Ryser, H.J., private communication.

[42] Scott, D. & Strachey, C., "Toward a Mathematical Semantics for Computer Languages," Polytechnic Institute of Brooklyn Press, 1971.

[43] Seitz, C. "System Timing," Reference [30], Chapter 7.

[44] Strang, G., "Linear Algebra and its Applications," Chapter 6, 2nd. ed., Academic Press, 1980.

[45] Tamura, E., Ogawa, K. & Nakano, T., "Path Delay Analysis for Hierarchical Building Block Layout System," Proc. of the 20th Design Automation Conference, pp.403–410, 1983.

[46] Terman, C.J., "Simulation Tools for Digital LSI Design," VLSI Memo 83–154, Doctoral Dissertation, MIT., Oct. 1983.

[47] Thomas, D.E. & Nestor, J.A., "Defining and Implementing a Multilevel Design Representation with Simulation Applications," IEEE Trans. on Computer-Aided Design of ICAS, vol. CAD-2, no.3, pp.135–145, July 1983.

[48] Varga, R.S., "Matrix Iterative Analysis," Prentice-Hall series in automatic computation, 1962.

[49] Von Herzen, B., Master Thesis, in preparation.

[50] Wawrzynek, J. & Lin, T–M. "A Bit Serial Architecture for Multiplication and Interpolation," 5067:DF:83, Computer Science, Caltech, Jan. 1983.

[51] Wawrzynek, J., Lin, T–M. & Mead, C.A., "A VLSI Approach to Sound Synthesis," the 10th International Computer Music Conference, Paris, France, Oct, 1984.

[52] Whitney, T., Doctoral dissertation, in preparation.

[53] Whitney, T. & Mead, C.A., "Pooh: A Uniform Representation for Circuit Level Designs," Proc. of the IFIP TC 10/WG 10.5 International Conference on VLSI, pp.401-411, Trondheim, Norway, Aug. 1983.