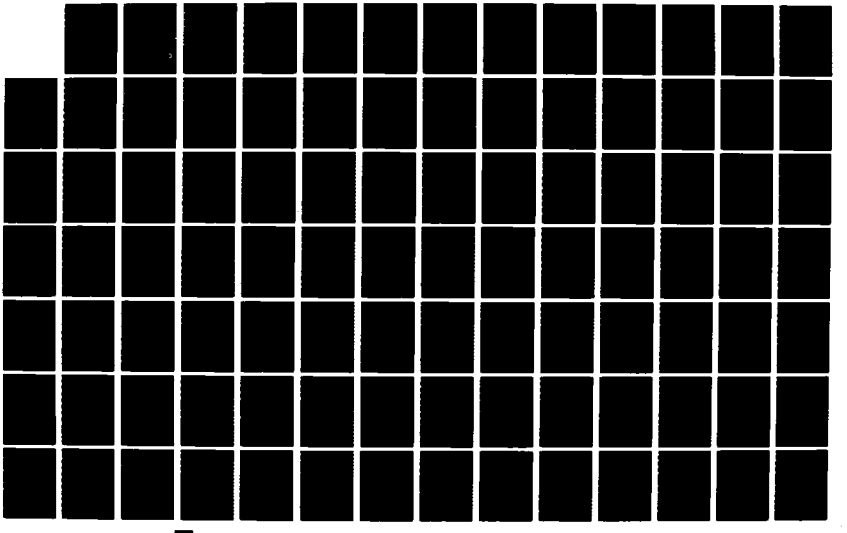
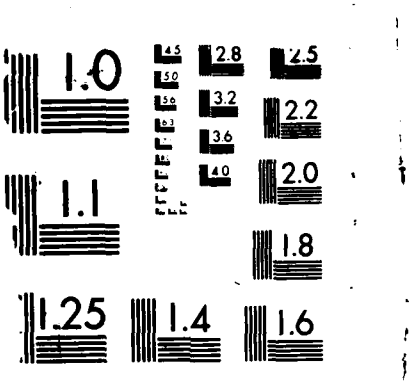
AD-A182 869 RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI(U) 1/2
ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB F LAI
JUL 87 UILU-ENG-87-2244 N00014-84-C-0149

UNCLASSIFIED F/G 9/1 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963-A

DTIC FILE COPY

July 1987 UILU-ENG-87-2244

COORDINATED SCIENCE LABORATORY
College of Engineering

AD-A182 869

RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI

Feipei Lai

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Approved for Public Release. Distribution Unlimited.

87 7 31 061

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-87-2244	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Semiconductor Research Corporation Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) ONR: Arlington, VA 22217 SRC: Research Triangle Park NC 27709
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Semiconductor Research Corp. & Joint Services Electronics Program (JSEP) 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER SRC: 86-12-109 JSEP: N00014-84-C-0149
8c. ADDRESS (City, State, and ZIP Code) See block 7b.	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI

12. PERSONAL AUTHOR(S)

Lai, Feipei

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	July 1987	117

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	circuit optimization, CMOS, VLSI, iJADE,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A closed-loop design system, iJADE, has been developed in Franz LISP. iJADE is a hierarchical CMOS VLSI circuit optimizer. Using a switch-level timing simulator and a timing analyzer, the program pinpoints the critical paths. The path delay reduction algorithms and a rule-based expert system are then applied to adjust transistor sizes such that the speed of the circuit can be improved while keeping constraints satisfied. iJADE is also capable of detecting and correcting the timing errors of synchronous circuits. The circuit is described in SPICE-like input format and then partitioned into blocks. Delays are computed on a block-by-block basis hierarchically, using a simple model based on input rise time, block type, and output load.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

19. Continued

In synchronous VLSI circuits, the detection and correction of timing errors due to clock skew requires accurate simulation of clock waveforms in addition to data path waveforms. To detect timing errors the hold time and the set-up time of each latch are checked with realistic signal waveforms. To correct the timing errors, two procedures are used: (1) resizing transistors in clock paths and (2) resizing transistors in critical paths and shortest signal paths. Experimental results from CMOS circuits including a 4-bit ALU show that iJADE-optimized circuits perform competitively or even better in terms of glitch avoidance, propagation delay time, and chip area. The use of this program can significantly reduce the design time over the conventional manual approach.

RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI

BY

FEIPEI LAI

B.S., National Taiwan University, 1980
M.S., University of Illinois at Urbana-Champaign, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI

Feipei Lai, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1987

A closed-loop design system iJADE has been developed in Franz LISP. iJADE is a hierarchical CMOS VLSI circuit optimizer. Using a switch-level timing simulator and a timing analyzer, the program pinpoints the critical paths. The path delay reduction algorithms and a rule-based expert system are then applied to adjust transistor sizes such that the speed of the circuit can be improved while keeping constraints satisfied. iJADE is also capable of detecting and correcting the timing errors of synchronous circuits. The circuit is described in SPICE-like input format, and then partitioned into blocks. Delays are computed on a block-by-block basis hierarchically, using a simple model based on input rise time, block type, and output load.

In synchronous VLSI circuits, the detection and correction of timing errors due to clock skew requires accurate simulation of clock waveforms in addition to data path waveforms. To detect timing errors the hold time and the set-up time of each latch are checked with realistic signal waveforms. To correct the timing errors two procedures are used: (1) resizing transistors in clock paths, and (2) resizing transistors in critical paths and shortest signal paths. Experimental results from CMOS circuits including a 4-bit ALU show that iJADE-optimized circuits perform competitively or even better in terms of glitch avoidance, propagation delay time, and chip area. The use of this program can significantly reduce the design time over the conventional manual approach.

Accession For
NTIS
DTIC
COPY
INSPECTED
6
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation and gratitude to Professor Timothy N. Trick, my dissertation advisor, for his invaluable guidance and support during the course of my graduate studies. Professor Chung-Laung Liu, my dejure advisor, was always there whenever I needed help. I owe very much to his understanding, encouragement, and direction. He is both a great teacher and a very good friend. I would also like to thank Professor Sung-Mo Kang for his excellent technical advice. Professor Saburo Muroga and Sylvian R. Ray are appreciated for being members of dissertation committee and their suggestions. Professor Dwight D. Hearn was my TA job supervisor helped me through a difficult year. I wish to thank Professor Vasant B. Rao and all the members of the Circuits and Systems Group at the Coordinated Science Laboratory, Urbana, for many interesting discussions and helpful suggestions. Also the kindness, wisdom and support from Professor Nachum Dershowitz would never be forgotten.

Finally, the dissertation would not have been done without the encouragement, patience, and support from my family. They have always been a great source of inspiration to me. This thesis is dedicated to all of them.

TABLE OF CONTENTS

CHAPTER 1.

INTRODUCTION

The speed of an integrated circuit is limited by its critical paths. When the frequency of the clock signals or strobe signals is increased so that critical paths fail to function properly due to insufficient time for the signal to propagate through that logic path, the performance is said to have reached its limit regardless of how well the other logic paths can function. Therefore it is obviously desirable to improve the speed of these logic paths until all logic paths are equally critical or until the increase in silicon area required for the speed improvement becomes intolerable.

The goal of VLSI design automation is to speed up the design process without sacrificing the quality of the implementation. A common means of achieving this goal is through the use of optimizations tool. Several delay-time optimization approaches have been published [Rue 77], [Kan 81], [Gla 84], [Lee 84], [Fis 85], [Mat 85], [Kao 85]. Most of them deal with the delay equation for critical paths or for single cells only. Mathematical optimization algorithms are often used to minimize delay times. The optimization method used to determine the "best" sizes of the transistors in a chain is based on the delay model. But often "optimization" in the mathematical sense may bear no relationship with reality, because it might lead to a driver with channel widths too large to be implemented on chip. A design truly optimized for delay time is seldom practical. This is because the silicon area increases very rapidly when the minimum delay time is approached. Furthermore any approach that neglects the loading effect of transistors other than those in the critical paths is often misleading. This is because as the original critical path is sped up, another path might become a new critical path whose delay is approximately the same as the original delay.

This dissertation presents a new design optimization method. The design system iJADE combines an analytic tool (timing simulator) with a rule-based expert system. iJADE mimics the problem solving behavior of a human designer and deals with the dynamic design environment. First, it analyzes the circuit to be optimized and compiles the performance information. Then based on results from analysis phase the expert system decides what actions should be taken to adjust the sizes of the transistors. Since critical paths may change after the size adjustments, a closed-loop evaluation and refinement of design parameters are used to handle such dynamic situations.

The global approach to timing performance optimization should involve operations at the logic, topological and physical level of the circuit description. In particular, at the logic level, it can modify the internal structure of the logic gates and their interconnection inside each combinational module [Mur 79]. At the topological level, it repositions the modules, and as a consequence their gates, to reduce the interconnection delay along the critical paths. At the physical design level, it optimizes the gate sizes to improve the switching speed. In this dissertation we will concentrate on the physical level optimization to improve circuit performance. The device sizing strategy used here is a heuristic descent technique. The size of the device implementing the drivers is adjusted by changing its width, which is a linear function of the channel width w ; the device length is kept constant except for a few situations.

Given a CMOS circuit with N transistors of sizes (channel widths) $x_1, x_2, \ldots, x_N$, the following question is considered: How can the circuit's performance be optimized by adjusting the x_i s? Two figures of merit are of special interest, namely the minimum clock period (T) at which the circuit will operate and the active chip area (A) monitored as the sum of transistor sizes.

In Chapter 2, the algorithms and techniques of hierarchical programming and data bases are discussed. Under the well-known structured-design methodology, a design is partitioned into several levels of hierarchy. This partitioning helps designers focus on one particular level of design at any given time and allows the complexity of a large system to be managed effectively. A hierarchical design is best supported by a hierarchical simulator for determining its functionality and performance. Currently most circuit optimization is done manually with feedback from circuit simulation. Before layout, designers design electronic circuits temporarily assuming the sizes of some circuit parameters, analyze the circuits by CAD programs, and then modify the circuits based on the analysis. After having obtained more specific values for circuit parameters from actual layout, designers finalize the design of the electronic circuits and simulate the laid-out circuits by CAD programs [Mur 82]. It is very time consuming and impractical to analyze the entire chip in VLSI design. We propose a timing optimization system iJADE that uses a switch level timing simulator and a rule-based expert system as a decision maker to replace the circuit designer. The switch-level timing simulator will provide timing information much faster than circuit simulation such as with SPICE2 [Nag 75]. The expert system mimics an experienced circuit designer's process. The circuit is described in terms of transistors of different types and sizes, along with first-order estimates of the interconnect resistance and capacitance.

A frame is a generalized property list. It can have any number of slots. The slots can have any number of facets. The facets, in turn, can have any number of values. This structure has the desirable feature of uniformity at every level. Schematic frames are those frames which hold the information about the circuit elements as they are initially entered and further classified by the program. Just below the root (whole) there are eleven frames: transistor, capacitor, resistor, model, node, latch, sub-call, input, subcircuit, block, and strongly connected component. The three data base operators (store, remove, fetch) are used to build and manipu-

late the data base. A hierarchical frame data base which is consistent with the hierarchy of the circuit description is then constructed. Before the timing optimization there are typically hundreds of critical paths that are as much as a factor of two slower than the slowest path in the final design of VLSI chips. With careful circuit design the speed of a chip can be improved. noise can be eliminated and clock skew can be reduced to a tolerable level [Sho 82].

In Chapter 3. a new switch-level timing simulator JADE with several features is discussed. JADE is a hierarchical switch-level timing simulator for CMOS circuits. The circuit is described to the program in a SPICE-type input format as a multi-level hierarchy of subcircuits and/or transistors and sources. The subcircuits at the lowest level in the hierarchy are composed of MOS transistors. resistors. and capacitors. The entire circuit is first partitioned into various blocks at each level of the hierarchy. At the lowest level a block could be one of four types. namely. an input source. a PMOS-driver block consisting of P-type dc-connected MOS transistors. an NMOS-driver block consisting of N-type dc-connected MOS transistors. a pass-block consisting of dc-connected pass transistors and resistors. The entire circuit can be partitioned in time linearly proportional to the number of MOS devices in the circuit.

The signal delays in JADE are computed hierarchically on a block-by-block basis. For blocks at the lowest level in the hierarchy the delays are computed using a simple. yet fairly accurate. delay model that takes into account the input slew rate. the configuration of transistors within the block. and the output load. The characteristics of the delay operation are stored in delay tables whose entries are computed by simulating three basic circuit primitives on an accurate circuit simulation program such as SPICE2. Previous attempts at switch-level timing simulator for NMOS circuits have used five primitives which results in extremely large and unmanageable delay tables. However. JADE is able to manage with only three basic primitives from which are constructed fairly small delay tables (around 120 entries). The delay through a block can then be computed fairly accurately from these delay tables by mapping the block

to an equivalent primitive and using a simple time-scaling technique [Rao 85].

At each level of the hierarchy a directed graph is constructed with a vertex for every block in that level and a directed arc from vertex v_i to v_j if an output node of block i is an input node to block j. Strongly connected components in this graph are detected and collapsed into a single vertex thereby resulting in a directed *acyclic* condensation graph. JADE then simulates the blocks corresponding to the vertices of the condensation graph in topological order. A waveform-relaxation based dynamic windowing method is used to simulate those blocks within strongly connected components [Whi 84]. The program then computes two parameters for each circuit node, namely, the *earliest starting time* computed by scanning the vertices of the condensation graph in the forward topological order, and the *latest completion time* obtained by scanning in the reverse order. A node is said to be *critical* if both parameters are equal. A block is said to be *critical* if it has both a critical input node and a critical output node. A critical path is defined to be the longest directed path in the graph composed only of critical blocks and/or subcircuits. Hence JADE is able to point out the critical path at each level in the hierarchy. The hierarchical approach used in JADE saves a considerable amount of memory space and the program executes at least 100 times as fast as SPICE2, yet produces discrete waveforms with timing typically within 10 percent of SPICE2.

In Chapter 4, the path delay reduction algorithms and rule-based expert system are discussed. iJADE is a hierarchical CMOS VLSI circuit optimizer. Using a switch-level timing simulator and a timing analyzer, the program pinpoints critical paths. The circuit starts with all minimum size transistors, and iJADE only adjusts those which need to be changed to improve the performance. The path delay reduction algorithms and rule-based expert system are applied to size the transistors such that the speed of the circuit can be improved while certain constraints are still satisfied. The equalized input arrival time algorithm is used to reduce glitches when the difference between input arrival times is greater than a threshold value. The

path delay reduction algorithms handle the more general circuit tuning techniques so as to speed up the execution time. For example, the RC time constant is used as the first-order estimation of how large the size scaling factor should be to achieve the timing specification. The rule-based expert system takes care of the more specific cases and makes the deductions from the information provided by the analysis tools. Because of the situation-action property in circuit design we choose the *forward reasoning* as our inference engine technique. There are about 20 rules to adjust the size of transistors in different situations. Experimental results on a 4-bit ALU and other circuits show that iJADE-optimized CMOS circuits perform competitively or even better in terms of glitch avoidance, propagation delay time, and chip area.

In Chapter 5, a timing error detector and corrector for synchronous circuits is discussed. The clock pulse distribution has been a serious problem in VLSI design when logic elements are physically separated but need to be gated simultaneously [Gla 77]. In synchronous VLSI circuits, the detection and correction of timing errors due to clock skew requires accurate simulation of clock waveforms in addition to data path waveforms. To detect timing errors the hold time and the set-up time of each latch are checked with realistic signal waveforms. To correct the timing errors two procedures are used: (1) resizing transistors in clock paths, (2) resizing transistors in critical paths and shortest signal paths. An intelligent decision making algorithm is used to judge where and how large the transistors should be resized. A 160-transistor example demonstrates that the use of this program can significantly reduce the design time over the conventional manual approach.

Finally, in Chapter 6, we provide some conclusions along with some suggestions for future research. Power dissipation is another important concern with the performance of very large-scale circuits. Beginning early with the chip floor plan, it would be helpful to consider laying the power and ground wires on a VLSI chip. This will provide the designer with the chance to estimate the power distribution at the pre-layout stage. Observation of the hot spots

on the whole chip is helpful for the designer to achieve better thermal immunity for the circuits.

CHAPTER 2.

iJADE SYSTEM IMPLEMENTATION

2.1. Introduction

The program iJADE is written in Franz LISP and runs on a VAX under the Unix operating system. The whole system consists of five parts: (1) the switch-level timing simulator. (2) the switch-level timing analyzer, (3) the path delay reduction algorithms. (4) a rule-based expert system. and (5) the frame data base. The switch-level timing analyzer finds worst-case critical paths timing performance. The switch-level timing simulator provides more accurate timing information than the timing analyzer and it also yields functional information. The path delay reduction algorithms deal with the general case of circuit tuning techniques. They help to speed up the program execution time. The rule-based expert system stores the circuit domain knowledge and applies the optimization techniques. Through pattern and functional recognition the expert system decides the proper device size according to its loading capacitance and topological structure. The frame data-base consists of the target technology file. circuit topology file and the calculated performance information. The relation and interaction between these five parts is depicted in Figure 2.1.

2.1.1. Signal Representation

Our primary concern is to provide a fast and accurate optimization tool for VLSI circuits which gives adequate information on the performance of circuits with a reasonable expenditure of computation time. A new class of digital simulators has recently emerged specifically for simulating MOS VLSI circuits. These switch-level simulators [Bry 81], [Jou 83], [Ter 83], [Ous

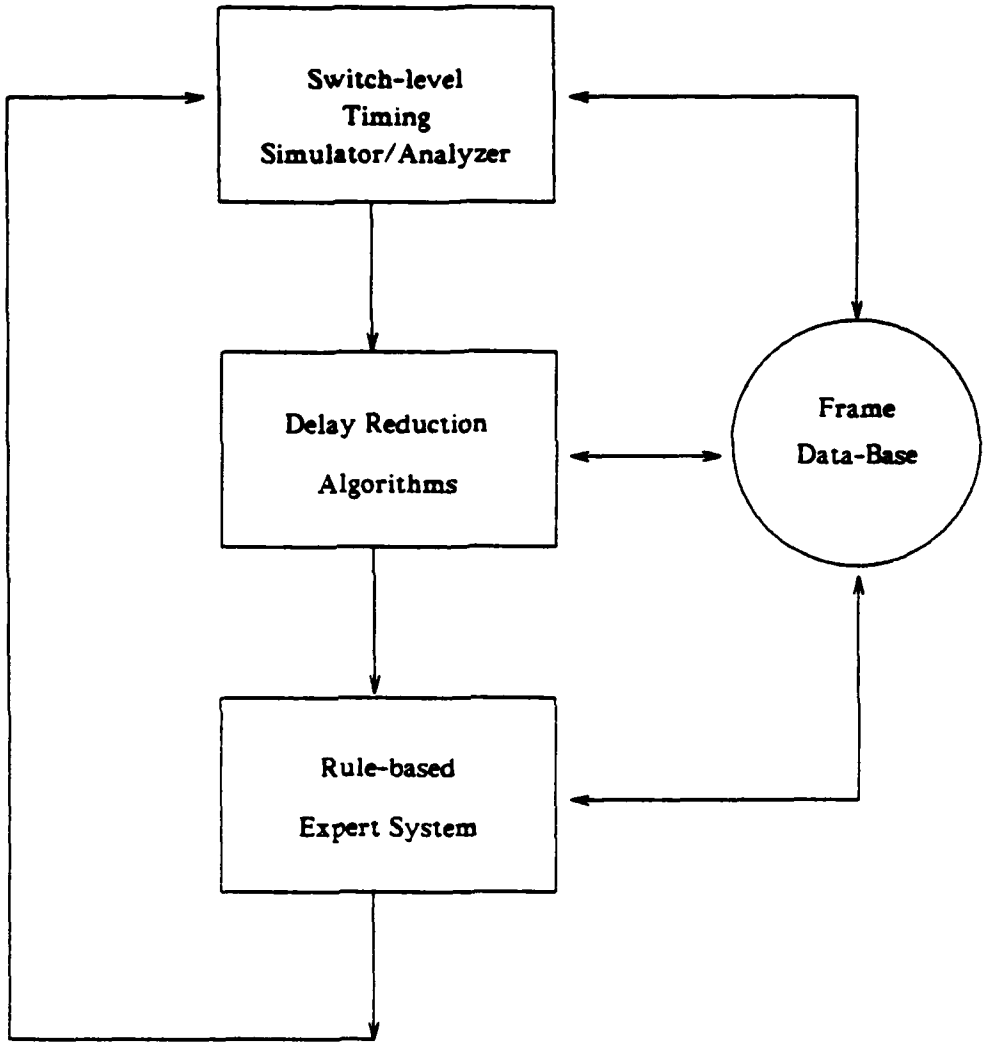


Figure 2.1. iJADE system.

85] model an MOS circuit as a set of nodes connected by transistor switches. Each node occupies a discrete number of states 0, 1, or u for the intermediate or unknown state, and each switch is either *open*, *closed*, or in an intermediate state. Digital simulators, in general, operate at sufficient speeds to test entire VLSI systems, since the circuit behavior is modeled at a logical rather than a detailed electrical level. We start with the signal representation to express the analog waveforms in terms of discrete type values. Let t_0 and t_f denote the initial and final times of the simulation. At any time $t \in [t_0, t_f]$, the three state digital signal $X_n(t)$ at

node n is related to its analog counterpart $V_n(t)$ as follows:

$$X_n(t) = \begin{cases} 0 & \text{if } 0 \leq V_n(t) \leq V_l \\ u & \text{if } V_l < V_n(t) < V_h \\ 1 & \text{if } V_h \leq V_n(t) \end{cases} \tag{2.1}$$

where V_l and V_h are two threshold voltages chosen such that $0 < V_l < V_h < V_{dd}$. Here, u is an intermediate state between the steady low (0) and high (1) states used to represent signals in transition.

A *transition* in a ternary signal is defined as a change in the ternary value of the signal at a certain time instant. Thus, to completely specify a transition, we need to specify both the type of transition and the time at which it occurs. A *transition type* is an ordered pair (x, y) where $x, y \in (0, 1, u)$ and $x \neq y$. There are four possible transition types, namely, (0, u), (u, 1), (1, u), (u, 0). The data structure of a transition is expressed as follows:

$$([0, 1], u, t_1), \text{ or } (u, [1, 0], t_2).$$

If, however, a ternary signal is constant throughout the time interval $[t_0, t_f]$ then it does not undergo any transitions. We represent such a signal by a sequence consisting of a single transition of a suitable type taking place before t_0. Thus a waveform that is always 0 is represented by $(u, 0, -1)$, and a constant 1 signal by $(u, 1, -1)$, where the integer -1 represents all time points $t < t_0$. A constant u signal, though seldom occurring in practice, can also be represented by $(0, u, -1)$. The entire simulation time interval $[t_0, t_f]$ is discretized by choosing a *minimum resolvable time*, denoted by h_{min}, so that a time point t can be represented by an integer k if $t \in [t_0 + k * h_{min}, t_0 + (k+1) * h_{min}]$. Thus two different time points within this interval are considered indistinguishable and are represented by the same integer k. The value of h_{min} is usually chosen to be very small, typically two or three orders of magnitude smaller than the rise or fall times of the analog signals. A *transition interval* for a node is the time interval during which the node is in the intermediate state, u. Associated with each transition interval I for a node n_k is a fanout list of blocks and/or subcircuits, denoted by $F(I)$. Let t_1 and t_2 denote the initial and final times of the transition interval I. Its data structure is

expressed as follows:

$$(t_1, t_2, F(I)).$$

Let $S = \alpha_1, \alpha_2, \ldots, \alpha_p$ be a *sequence* of transitions where each $\alpha_j = (x_j, y_j, k_j)$. The sequence S is said to be *chronological* if $k_1 < k_2 < \cdots < k_p$. A chronological sequence is said to be *compatible* if (x_j, y_j) is an allowable transition type for each $1 \leq j \leq p$ and $y_j = x_{j+1}$ for each $1 \leq j \leq p-1$, where p is the number of transition in a sequence. In a compatible sequence therefore, the final value of every term in the sequence is equal to the initial value of the succeeding term.

Two sequence $S_a = (\alpha_i)_{i=1}^{p}$ and $S_b = (\beta_i)_{i=1}^{p}$ with the same number of terms, and where $\alpha_i = (x_i, y_i, k_i)$ and $\beta_i = (x_i', y_i', k_i')$, are *type equal* if $x_i = x_i'$ and $y_i = y_i'$ for each $1 \leq i \leq p$ and *time equal* if $k_i = k_i'$ for each $1 \leq i \leq p$. The two sequences are *equal* if they both type equal and time equal. It must be noted that two sequences can be compared for equality if and only if they have the same number of terms.

Two successive terms α_i and α_{i+1} in a compatible sequence with $x_i \in [0, 1]$ are said to form a *complete pair* of transitions if $y_{i+1} = \neg x_i$ and a *partial pair* if $y_{i+1} = x_i$. A compatible sequence of transitions is said to be a *complete sequence* if it has no partial pairs. Given two compatible and complete sequences $S_a = [(x_i, y_i, k_i)_{i=1}^{p}]$ and $S_b = [(x_i, y_i, k_i')_{i=1}^{p}]$ that are type equal, we define a measure on the difference in transition times between the two sequences to be

$$\rho(S_a, S_b) = \max_{1 \leq i \leq p} \left| \frac{k_i - k_i'}{k_i} \right|.$$

2.2. Hierarchical Data Structure and Algorithms

CAD for hierarchical design has to satisfy two following conditions. First, it must support the hierarchical design methods. Secondly, it must provide improvements in capacity and

efficiency that can be expected from hierarchical tools. Precise hierarchical algorithms in which structured designs can be described are the chief sources of support of the design method. CAD programs must ensure fast excursions through the hierarchical structures, both through the levels at a certain design stage and also between design stages.

By retaining the hierarchy of a design in internal data structures, the problem of duplicate data storage is completely eliminated. While a flattened data structure for a VLSI design consumes huge amount of storage, a hierarchical approach only needs small portions of storage. In addition, any increase in circuit complexity that is obtained by better utilization of structure and repetition does not increase the size of a hierarchical data structure, in contrast to the flattened data structure. Therefore, hierarchical tools can solve the capacity problem of design data storage effectively. The run time of CAD programs is affected in a similar way. The execution of CAD programs on modules instead of on a complete circuit in itself constitutes an advantage because of memory allocation problem.

2.2.1. The Algorithms of iJADE

iJADE system consists of several structure building procedures and a design evaluation and parameters refinement loop. The algorithms of iJADE are described as follows:

1. Read the circuit file and design specifications.
2. Calculate the parasitic capacitances contributed by the devices.
3. Build the frame data base.
4. Partition the circuit hierarchically.
5. Order the partitioned blocks, subcircuits, and strongly connected components.
6. IF sequential THEN trace clock paths and identify latches.

(Select the tools: Timing Simulator or Timing Analyzer)

WHILE (rules can be applied) DO

7. Calculate the equivalent lump capacitance and resistance.

8. Calculate propagation delay times and search critical paths.

9. IF Sequential THEN

    Trace critical paths between latches

    Timing error detection

    Timing error correction.

10. Execute the path delay reduction algorithms.

11. Execute the production system.

12. Modify transistor sizes and/or insert buffers.

13. Update the frame data base.

UNTIL (specification satisfied).

14. IF success THEN (Print out results) ELSE (Print out suggestions).

2.2.2. Frame Data Base

A frame is a generalized property list [Win 81]. It can have any number of slots. The slots can have any number of facets. The facets, in turn, can have any number of values. This structure has the desirable feature of uniformity at every level. Schematic frames are those frames which hold the information about the circuit elements as they are initially entered and further classified by the program. Just below the root there are ten frames: transistor, capacitor, resistor, model, node, subcircuit-call, input, subcircuit, block, and strongly connected component. A hierarchical frame data base which is consistent with the hierarchy of the circuit description is then constructed. We store the structure on the property list of each frame under the property *frame*. Given this structure for frames, we implement the following functions to administer the data base.

1. (FGET frame slot facet): fetches information:

2. (FPUT frame slot facet data): stores information;

3. (FREMOVE frame slot facet data): removes a specific data;

4. (FREMOVE* frame slot facet): removes information of the access path;

5. (FREPLACE frame slot facet data): replaces with the new data;

6. (FGET-V-D frame slot): fetches information. The VALUE facet is inspected first. If nothing is found, then the DEFAULT facet is also inspected.

7. (FGET-I frame slot): fetches information. If nothing is found in the VALUE facet, then the function looks at the frames found in the given frame's AKO slot under the VALUE facet. This enables a frame to inherent information from the frames related to by the AKO relation, an abbreviation for a-kind-of relation.

* (FGET-Z frame slot): fetches information. It uses values, defaults, and value-finding function, not only as found in the given frame but also in the frames related by the AKO relation.

Schematic frames are those frames which hold the information about the circuit elements as they initially entered and further classified by the program. Just below the root "whole" there are eleven frames: transistor, capacitor, resistor, input, model, node, subcircuit, subcircuit-call, block, latch, strongly connected component(SCC). More detailed information about the contents of each frame can be found in Appendix A.

Data are stored in two kinds of representations: (1) flattened type, and (2) hierarchical type. The flattened type stores those data that are few but specific to some objects. The hierarchical one stores the structural information that is common to a group of flattened representations. The flattened type is linked to the hierarchical type through "ako" such that it can inherit properties from its ancestor, as shown in Figure 2.2. The usage of links in the frame data base assures the efficient use of storage space by reducing redundant data in the data base.

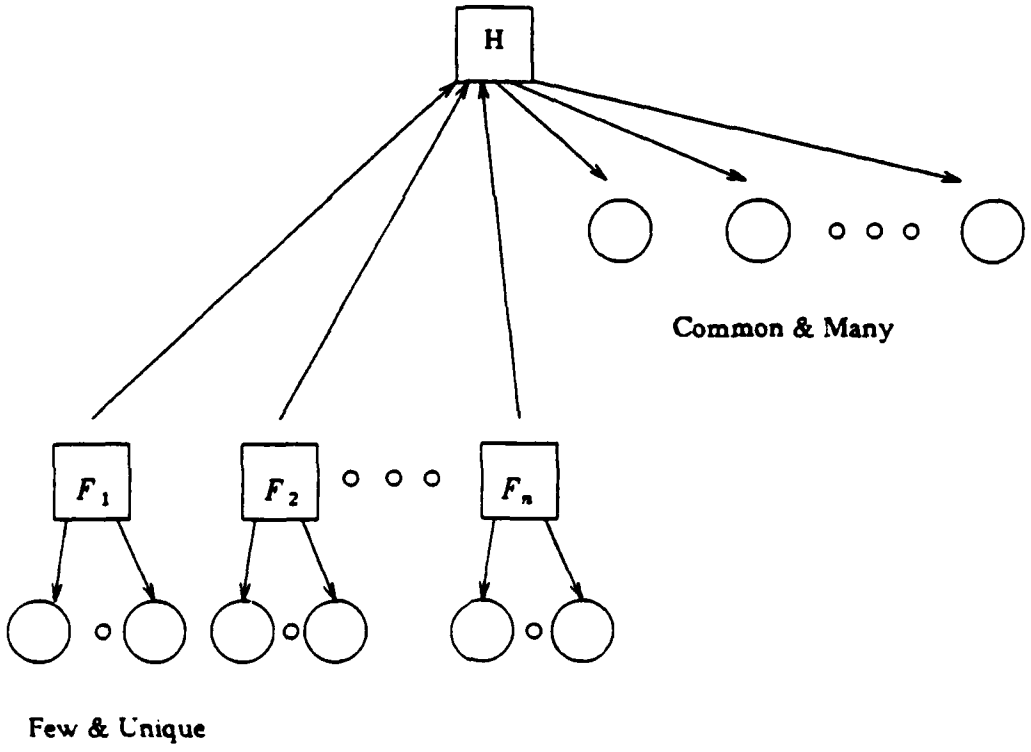


Figure 2.2. Inheritance property.

The *active* devices are stored in a stack with the current active device at the top of the stack. We do the *push* operation when entering a subcircuit and the *pop* operation when leaving a subcircuit. The list structure in LISP is best used in the stack implementation to keep the hierarchical relationship. The recursive nature of list processing makes the analysis of the subcircuits faster and easier. The hierarchical description is needed to locate and identify the problem site in VLSI circuits quickly and accurately.

2.2.3. Hierarchical Algorithms

The most significant advantage of hierarchical algorithms is that some operations can be done only once for all the same subcircuits in the different hierarchical levels. In the iJADE

system. the partitioning. ordering. and some static properties like two terminal nodes of the equivalent resistor algorithms are done hierarchically.

Let S_i denote the sequence of transitions computed by switch-level simulation. and let V_i be the actual *analog* waveform at a node $n_i \in N$ in the network. We can obtain the three-state digital equivalent of V_i using the transformation in Equation 2.1. Let S_i^* denote the sequence of transitions corresponding to this ternary digital equivalent. The aim of our switch-level timing simulator is to compute S_i that is time-comparable to S_i^*. such that. $\rho(S_i^*, S_i) < \epsilon$ where ϵ is a measure of the accuracy of the timing simulation. The function *simulation* is a recursive algorithm to simulate the circuit hierarchically as depicted below.

Two important functions. *pass-up* and *pass-down* are used to pass the information between different levels of the hierarchy. They act as information bridge in the hierarchical structure.

```
function simulation (circuit)
    For all the blocks. subcircuits.
    and SCC of the circuit in topological ordering DO
        BEGIN
        If module ∈ subcircuits Then
            (pass-down information)
            (simulation module)
            (pass-up information).
        If module ∈ NMOS-driver Then
            (driver-simulation module).
        If module ∈ pass-block Then
            (transition-interval-calculation)
            (pass-block-simulation module).
        If module ∈ SCC Then
            (windowing-relaxation-simulation module).
        END.
    END.
```

function pass-up
    For each output node of the subcircuit.
        Pass information from lower level to higher level.
    END.

function pass-down
    For each input node of the subcircuit.
        Pass information from higher level to lower level.
    END.

2.2.4. Simulation of Driver-block

Let Ω_f be a driver-block that is to be simulated with n_o as its output node and $INPUT(\Omega_f)$ as its input nodes. For each input node n_i, let S_i denote the sequence of transitions at that node, and let $z_i \in L$ denote the ternary value of the node signal at some time instant. Also, let S_o be the sequence of transitions to be computed and Z_o denote an instantaneous value of the signal at node n_o. A driver-block can be viewed as a network of switches between the drain and source nodes of its driver transistors whose conduction states are controlled by the ternary signals at the gate terminals. The basic idea in conventional switch-level simulation is that the signal at a node can only be changed by a signal at a stronger node and can change the signals only at weaker nodes.

We now introduce the notion of *zero-delay* through a block in a network. By this we mean that there is no delay element present in the block and that at any instant of time the ternary value of the output signals can be determined from those at its input signals at the same instant of time. The algorithm to obtain the zero-delay sequence of transitions at the output node of a driver-block begins with the simplification of parallel transistors of the driver-block. It then picks an internal vertex v in this simple graph and eliminates v from the graph and then simplifies the resultant graph. This process of elimination followed by simplification is repeated for each internal vertex. The end result is a simple graph with two vertices, namely, the output vertex and the V_{dd}/V_{ss} vertex. The *simplification* of a graph is a graph obtained by collapsing all parallel edges into a single edge whose edge sequence is the sum ($\vee$) of the

sequences of the parallel edges. We define the *elimination* of a vertex v from a simple graph as a procedure involving the following two steps:

(1) For every pair of vertices a and b adjacent to v in the graph, add an edge between a and b with the edge sequence of this new edge being the product ($\wedge$) of the sequences corresponding to the edges (v , a) and (v , b), respectively.

(2) Delete the vertex v and all the edges incident on it from the new graph obtained in step 1.

Once the zero-delay sequence is obtained the transition times are delayed by a delay operator and the whole sequence is filtered using techniques to be discussed below.

2.2.5. Delay and Filter Operator

The task of the delay operator is to alter the transition times of the zero-delay sequence so that the resulting ternary waveform corresponds fairly close to the ternary equivalent of the analog waveform if computed by an accurate circuit simulator. Each application of the delay operator is followed by a filtering operation which accounts for the effect of delaying a complete pair of transitions on the future transitions in the sequence. The filtering operator also transforms a partial pair transitions into a form that can be handled by the delay operator.

The delay operator is characterized by delay functions which are computed by an accurate circuit simulator for a set of standard circuit primitives and stored in tables. The primitive circuits do not change as long as the technology remains fixed. Hence, the computations of the delay functions are performed only once for each technology. The delay operator computes

```
function driver-simulation
    (zero-delay-sequence)
    (delay-operator)
    (filter-operator)
    END.
```

new values for transition times in a complete pair of transitions at a certain node in a general block in two steps. First, a mapping technique is used to transform the block into a configuration that resembles one of the primitive circuits for which the delay functions have been computed. Time scaling is then used to transform the new configuration into a standard primitive. Then, the delay values can be obtained through a table lookup. The three primitive circuits and circuit transformations are discussed in Chapter 3.

Consider any sequence S of transitions. We mark a term of S as "delayed" if the delay operator has been used previously on this term, otherwise, we mark it "undelayed." The subsequence of S consisting of all terms marked "delayed" is called the *delayed* part of S. The rest of the sequence is the *undelayed* part. Thus, we can consider any sequence of transitions to be the concatenation of its delayed part and undelayed part. Let us consider S as an input sequence to the filtering operator. The output of the filtering operation will then be a sequence S' which is computed as described below. First, the filtering operator replaces any partial pair $(x, u, k_j), (u, x, k_{j+1})$ of transitions in the undelayed part of S by two complete pairs $(x, u, k_j), (u, \neg x, k_j+1), (\neg x, u, k_{j+1}-1), (u, x, k_{j+1})$. This is done by the function *complete* used below. We will also make use of the function *window*(S, k_a, k_b) that returns those transitions in S occurring between k_a and k_b. The function that performs the filtering operation is given below.

2.2.6. Simulation of Pass-block

A pass-block consists of DC-connected pass transistors and resistors. Consider the set T_t of transition times of the signals at the gate nodes arranged in an ascending order. These time points divide the time interval of simulation into several *phases* such that during each phase $\phi_j = (k_j, k_{j+1})$ the signal at each gate node is at a fixed ternary value, i.e., 0, u, or 1. The time

```
function filter (S)
    (complete S)
    For each transition in S marked "undelayed" DO
        BEGIN
            (x . u . k_j )← first transition in S :
            (u . ¬x . k_{j+1})← second transition in S :
            k_j'. k_{j+1}' ← delay (k_j . k_{j+1}):
            (window S' 0. k_j ):
            y ← final value of S' :
            If (equal y x) Then (append [(x . u . k_j'). (u . ¬x . k_{i+1}' )] S' ).
            If (equal y u ) Then (append (u . ¬x . k_{j+1}) S' ).
            S' ←(cddr S' ):
        END.
    (return S' ):
    END.

function transition-interval-calculation
    For each transistor in the block DO
        Get the transition time of the gate discrete waveform sequence.
    Sort the transition times list.
    Composite the transition times pairs.
    END.
```

k_j is the *initial time* and the time k_{j+1} is the final time of phase ϕ_j. Let $s_{i,j}$ denote the fixed ternary state of the signal at gate node n_i during phase ϕ_j. We partition the set of drain and source nodes of pass transistors in the pass-block into three subsets:

(1) N_i. the set of nodes of input strength.

(2) N_p. the set of nodes of pull-up strength. and

(3) N_n. the set of nodes of normal strength.

Given the sequences of transitions at each node in N_i and N_p in the pass-block. the task is to compute the sequences of transitions at the nodes in N_n. We simulate the pass-block in the first phase ϕ_1 followed by the next phase and so on. updating the node sequences for the normal nodes in each phase. The simulation of a phase ϕ_j begins by constructing an undirected graph H_j with vertex set corresponding to the drain and source nodes of the pass transistors and the two terminals of the resistors. and the edge set E_j initially empty. For each resistor. an edge is inserted between the two terminals of the resistor. For each pass transistor. an edge

is inserted between drain and source if $s_{i,j}=1$. Each connected component of the graph represents a switching network with nodes connected by two terminal switches that are in the closed state. Consider a component C_r of the graph. Let $STRONG_r$ denote the subset of the strongest nodes in C_r, where the node strengths are ordered as $input > pull-up > normal$. The *strength* of the component C_r is then defined to be the strength of its strongest node(s). If $|STRONG_r|>1$ and the strength of C_r is either *input* or $pull-up$ and these strongest nodes are not at the same state (0, u, 1), then a *conflict* is declared at each normal node in the component. In case a node is experiencing a conflict in the present phase ϕ_j, the intermediate state u is assumed. A pass transistor is said to have a state u' if its gate node is at the u state in the present phase but occupies a 1 in the next phase. A transistor in the u' state in the present phase is in an *intermediate* conducting state but would occupy a *closed* state during the next phase. The transistors whose gate signals are in the 0 state or in a u state but not in a u' state are ignored during the present phase.

If the strength of C_r is *normal*, then *charge sharing* is said to take place among the normal nodes in the component. Given any sequence of transitions, one can define the *initial value* of the signal to be the ternary value before the occurrence of the first transition and the *final value* to be the one after the last transition. For each node $n_v \in C_r$, let $S(n_v)$ denote the existing sequence of transitions at the node and s_v denote the final value of this sequence. We define an *equivalent voltage* v_{eq} corresponding to the ternary signal s_v as $v_{eq}=0, 0.5*V_{dd}$, or V_{dd} depending on whether $s_v=0$, u, or 1, respectively. The *charge* on a node n_v is defined to be the product $v_{eq}*CAP(n_v)$, where $CAP(n_v)$ is a lumped capacitance from node n_v to ground. In the case of charge sharing among the nodes of normal strength in a component, the total charge in the component is computed by summing up the charges on each node in the component and this quantity is divided by the total capacitance to yield a final voltage. The final ternary value s_f reached by all the nodes in the component after charge sharing

is then computed from v_f as s_f =0, u, or 1 depending on whether $v_f \leq V_l$, $V_l < v_f < V_h$, or $V_h \leq v_f$, respectively, where V_l and V_h are the low and high threshold voltages. The function *pass-block-simulation* is described next.

2.2.7. Simulation of Strongly Connected Components

In this section we discuss the use of a special windowing technique to simulate the driver-block and pass-block within a strongly connected component (SCC). The algorithm splits the entire time interval of interest [0, K] into various time slots or windows such that all pairs of signal transitions take place entirely within one of these windows. This is achieved by maintaining a sequential list of intervals of transitions which are updated dynamically as the algorithm progresses. The algorithm is, in a sense, event-driven, since only those circuit blocks that are active within a window are processed and the fanouts of the output nodes of these blocks are scheduled for future processing. The *windowing-relaxation-simulation* is described below.

```
function pass-block-simulation
    For each transition interval of the block DO
        BEGIN
            (linking)
            (strength)
            If (equal strength 'conflict) Then
                (conflict).
            If (equal strength 'normal) Then
                (normal).
            If (equal strength 'pull-up) Then
                (transformation)
                (delay)
                filtering).
            If (equal strength 'input) Then
                (delay)
                (filtering).
        END.
    END.
```

2.2.8. Timing Analysis

For circuit or timing simulation a set of input test vectors is provided to get the corresponding output waveforms. Although in principle this approach can catch every sort of timing problem, it is very expensive to perform. Moreover, it is difficult to construct a set of test data which is guaranteed to cover all relevant cases. An input-independent timing analyzer is needed to find the worst-case timing performance. We calculate the worst-case equivalent resistance for driver block instead of dynamic equivalent resistance in simulator. In the timing analyzer an exhaustive search algorithm is used in function *worst-r* to find the worst-case equivalent resistance of a driver block.

```
function windowing-relaxation-simulation
    For each interval DO
        For each fanout DO
            BEGIN
                Simulation of the first window.
                Update interval with output node's waveform.
                If converge Then piece waveform together.
            END.
        If no overlap with first window
            Then remove first window
            Else modify the interval.
    END.

function worst-r
    Start with one of the two equivalent resistor nodes.
    For all the paths DO
        Expand the first path with the neighbors of the leading node.
        Update the leading node and the serial resistance of new paths.
        If a path reaches the other end of the block Then
            Update the maximum equivalent resistance.
            Remove the path from the path-list.
        Else Appends new paths to the path-list.
        END.
    END.
```

2.2.9. Critical Path Generation

Function *critical-paths* is a recursive algorithm to trace the critical path hierarchically. Each node is processed once - when the delay time from the start node to all nodes feeding it have been determined. By this means, the output of the node being analyzed will just be its own delay time plus the time of the maximal delay input. When the maximum delay times for all nodes have been determined, the maximum time for the end point will be known. Determining the critical path then corresponds to just tracing back through the maximal delay inputs feeding each node until the start node is reached [Hit 82].

function critical-paths
    Start backtracking from the node N_{md} with the maximum delay time.
    If $N_{md} \in OUTPUT(Block_i)$
        Then (trace-block)
        Else (Go-down one level of the hierarchy) (trace-subcircuit).
    END.

function trace-block
    Append the block to critical paths.
    Find a critical node N_{cr} from $INPUT(Block_i)$.
    If $N_{cr} \in INPUT(WHOLE-CIRCUIT)$
        Then STOP.
    If $N_{cr} \in OUTPUT(Block_j)$
        Then (trace-block)
        Else if $N_{cr} \in OUTPUT(Subcircuit)$
            Then (Go-down one level of the hierarchy)
            Else (Go-up one level of the hierarchy)
        (trace-subcircuit).
    END.

function trace-subcircuit
    Pass down N_{cr} from upper level.
    If $N_{cr} \in OUTPUT(Block_k)$
        Then (trace-block).
    If $N_{cr} \in OUTPUT(Subcircuit)$
        Then (Go-down one level of the hierarchy)
        Else (Go-up one level of the hierarchy).
    (trace-subcircuit).
    END.

2.2.10. Clock Path Tracing

In synchronous circuit timing analysis we identify the latch and its clock triggering times in order to check its timing validity. The clock path is traced from the defined clock node forward through the clock buffer in different hierarchical levels. The algorithm is similar to the one that traces critical paths except the latter does it backward.

function clock-path
    Start from the defined clock node N_{clk}.
    If N_{clk} is the only input of $Block_i$
    except power supply and ground.
        Then (trace-block-clock)
        Else (Go-down one level of the hierarchy) (trace-subcircuit-clock).
    END.

function trace-block-clock
    Append the block to clock paths.
    Append output nodes $Block_i$ to clock-nodes.
    For all new clock nodes DO
        If $N_{new-clk}$ is the only input of $Block_j$
        except power supply and ground.
        Then (trace-block-clock).
        (Go-down one level of the hierarchy forward)
        (trace-subcircuit-clock).
        (Go-up one level of the hierarchy forward)
        (trace-subcircuit-clock).
    END.

function trace-subcircuit-clock
    Pass $N_{new-clk}$ from previous level.
    If (null $N_{new-clk}$) Then STOP.
    If $N_{new-clk}$ is the only input of $Block_k$
    except power supply and ground.
        Then (trace-block-clock)
        Else STOP.
    (Go-down one level of the hierarchy forward)
    (trace-subcircuit-clock).
    (Go-up one level of the hierarchy forward).
    (trace-subcircuit-clock).
    END.

2.2.11. Propagation of Capacitance Changes

The capacitance changes produced from sizing transistors must be propagated to the same position but in the different levels of the hierarchy. The procedure provides up-to-date capacitance loading information so that the later tuning of the path delay has the appropriate capacitance for transistor sizing. The function *change-lump-capacitance* is a recursive algorithm described as follows.

2.3. Summary

We began this chapter by defining transitions between ternary states and showing how sequences of transitions can be used to represent ternary digital waveforms of signals. A frame data base structure for circuit topology representation was discussed. Data base operators were

```
function change-lump-capacitance
    Change the gate capacitance.
    For all subcircuits that have the node as one of its output nodes DO
        (load-down)
    (load-up)
    END.

function load-down
    Change the node lump capacitance.
    For all subcircuits that have the node as one of its output nodes DO
    (load-down)
    END.

function load-up
    If the node is one of the input nodes of a module Then
        Get the corresponding node in the upper level.
        For all subcircuits that have the node as one of its output nodes DO
            (load-down)
        (load-up)
    END.
```

implemented to facilitate the manipulation of the hierarchical data base. The hierarchical frame data base is the common data base that stores circuit structure, timing performance and all the information shared by the analysis tool and the rule-based expert system. An integrated system with a common shared data base does not have to transform the data format between the different phases of the CAD system. We also presented algorithms that perform a switch-level timing simulation, the critical path tracing, and some operations in circuit optimization. A circuit is divided into four kinds of structures: driver-block, pass-block, SCC, and subcircuit. In the case of a driver-block we showed that the zero-delay state of its output node at any instant of time is a function of the states of its input nodes at the same time instant. For a pass-block, we presented a more complex, and somewhat heuristic, approach utilizing the full power of conventional switch-level simulation. In the case of an SCC, an event-driven, dynamic windowing relaxation algorithm is used to calculate the waveforms of its output nodes.

The input-independent timing analyzer uses the worst-case equivalent resistance and the slowest input slew-rate to calculate the worst-case timing performance. In the case of an SCC, a single pass algorithm is used instead of the relaxation algorithm as in the timing simulator. A Project Evaluation and Review Technique (PERT) model was used to trace critical paths of the circuit. The algorithm traces the hierarchy of the circuit structure and presents its components in a unique flattened representation. The flattened name is combined from the block name and the current active device list. The clock node and clock buffer are traced forward to identify latches and the clock triggering time to check the timing validity of the synchronous paths.

CHAPTER 3.

HIERARCHICAL SWITCH-LEVEL TIMING SIMULATOR

3.1. Introduction

The problem of switch-level timing simulation of a digital circuit can be summarized as follows. Consider the analog waveform $V_n(t)$, $t \in [t_0, t_f]$ at a certain node n in a digital circuit and choose $p-1$ threshold values, ordered as $v_1 < v_2 < \cdots < v_{p-1}$. Define the p-state digital equivalent of V_n to be

$$X_n(t) = x_i \text{ if } v_i < V_n(t) \leqslant v_{i+1}$$

where $x_0, x_1, \ldots, x_{p-1}$ are the p digital states and v_0 and v_p are the minimum and maximum values of the analog waveforms respectively. We also define

$$T_n = [t_k : V_n(t_k) \in (v_1, v_2, \ldots, v_{p-1})].$$

Thus T_n is the set of threshold crossing times of the analog waveform at node n in the circuit, or alternatively, the set of state transition times of its p-state digital equivalent. The aim of a switch-level timing simulator is to obtain the p-state digital equivalent X_n for each $n \in N$, with special emphasis on computing the elements of $T = \bigcup_{n \in N} T_n$, where N denotes the set of nodes of interest to the user.

The switch-level approach has been popularized recently by Bryant and others [Bry 81], [Jou 83], [Ous 85] as a simulation model for MOS digital systems. In this chapter we describe an application of the switch-level model to timing simulation. JADE is a hierarchical switch-level timing simulator for CMOS circuits implemented in LISP. The circuit is described to the program in a SPICE-type input format as a multi-level hierarchy of subcircuits and/or transistors, resistors, lumped capacitors and voltage sources. The signal delays in JADE are computed hierarchically on a block-by-block basis. For blocks at the lowest level in the hierarchy the

delays are computed using a simple, yet fairly accurate, delay model that takes into account the input slew rate, the configuration of transistors within the block, and the output load. The important issue in timing estimation of MOS circuit is whether or not the triggering transistor is fully turned on during the transition interval, and this depends on the input rise-time, the output load, and transistor size. We will use a notion of two-threshold delays to measure the effect of the slope of the input waveform on the timing at the output of a logic gate or a functional block. We treat V_{in} to be an analog ramp waveform with a full swing of V_{dd}. This waveform will then cross two threshold voltages V_l and V_h where $0 < V_l < V_h < V_{dd}$. Let t_1 and t_2 denote the two threshold crossing times. Let t_1' and t_2' be the output threshold crossing times. We defined $\Delta_{in} = t_2 - t_1$ as a measure of the delay of the input signal and two delay quantities, $\Delta t_1 = t_1' - t_1$, known as the *inertial delay*, and $\Delta t_2 = t_2' - t_1'$, known as the *rise/fall delay*. The voltage and timing relation of the delays is illustrated in Figure 3.1.

3.2. Circuit Partitioning and Ordering

A MOS digital circuit $\Omega(N, M, R)$ consists of a set of nodes N interconnected by a set of MOS transistors M and resistors R. There are four types of nodes: *input* nodes, *output* nodes, *pull-up* nodes, and *normal* nodes. Input nodes, which are modeled as ramp voltage sources, provide the strongest signals to the circuit from the outside. Examples of input nodes include the power supply, the ground node, as well as all the input clock signals. A pull-up node is attached to the power supply V_{dd} via some PMOS-driver(s) and linked to the ground via some NMOS-driver(s) only. The remaining nodes in the circuit are classified as normal nodes. A output node is an input node to another block or a print node. A output node can be a pull-up node or any DC connected node in the pass block.

For circuit partitioning, the pass transistors with a PMOST and NMOST pair connected in parallel are first identified. Then a pull-up node candidate is marked if it is connected to both PMOS and NMOS transistors excluding pass transistors. The potential pull-up node would be a

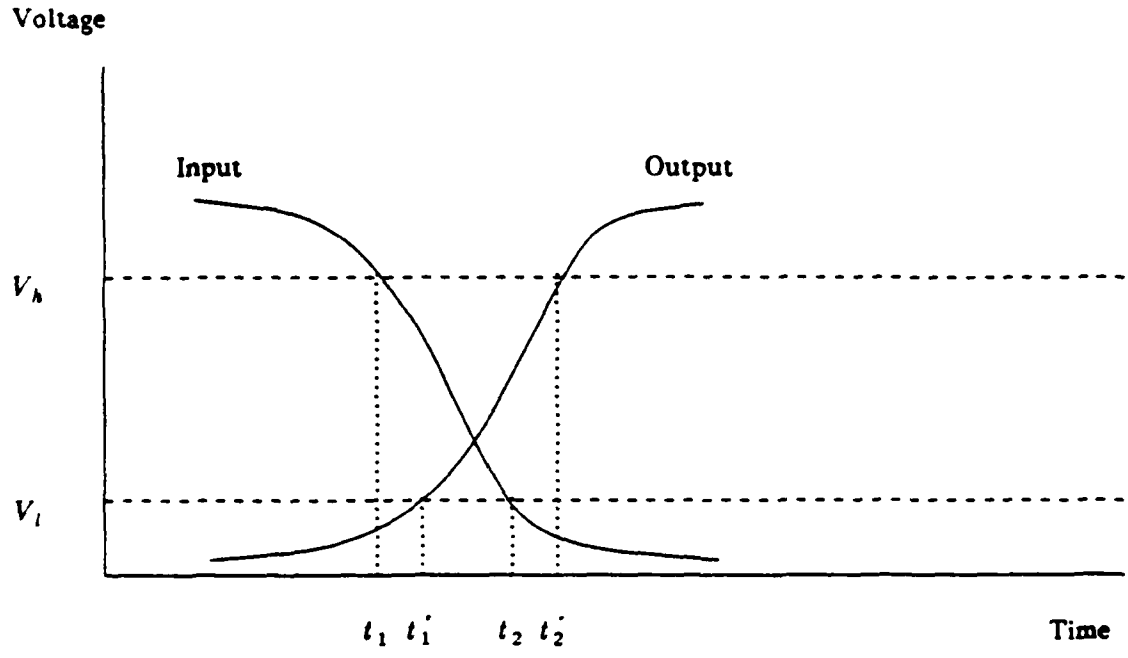


Figure 3.1. Switch-level delay time definition.

real pull-up node if it is connected through a DC path to the power supply or ground node. For each node of the circuit we check the DC paths one by one to see if there is any pass transistor pair. Nodes and transistors in a circuit are partitioned into various subcircuits or blocks by splitting the input nodes and pull-up nodes, where each block could be one of four types, *input source*, *PMOS-driver block*, *NMOS-driver block*, or *pass block*. In the node splitting phase, the new symbolic name of the split node is the combination of its original name and its DC connected neighbor name. We then search DC connected components using the new name as an indicator. After DC connected components are identified we return each combined name to the original one. The node splitting method is illustrated in Figure 3.2.

The PMOS-driver block is a DC connected component of PMOS transistor(s) between the power supply and pull-up nodes. The NMOS-driver block is a DC connected component of

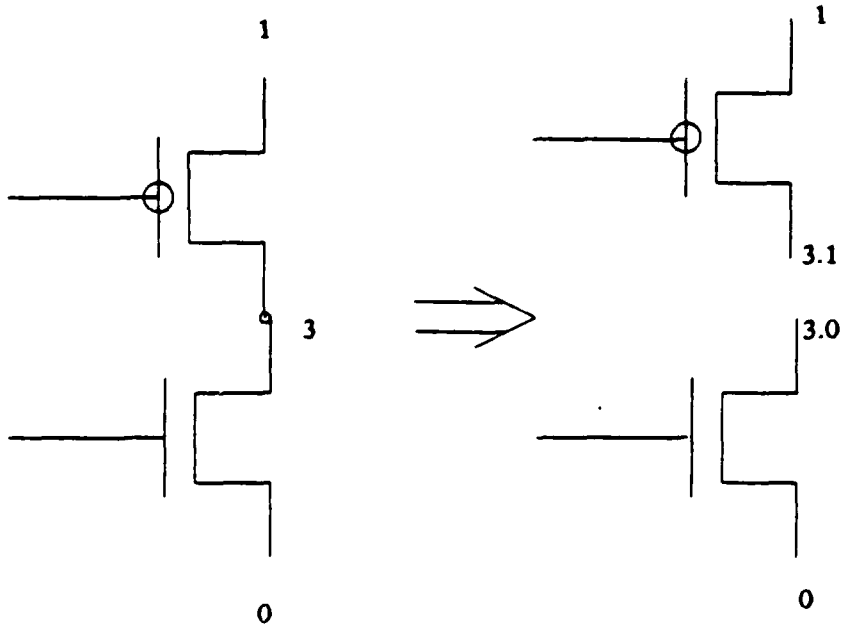


Figure 3.2. Node splitting method.

NMOS transistor(s) between the ground node and the pull-up node. The other DC connected components containing pass transistor(s) and resistor(s) are called pass-blocks.

Let Σ denote the set of partitioned blocks in the circuit and let $\Omega(N, M, \Sigma)$ denote the partitioned MOS circuit. Furthermore, let $INP(\Omega_i)$ and $OUT(\Omega_i)$ respectively denote the set of input nodes and output nodes of a block $\Omega_i \in \Sigma$. For each node $n_i \in N$ in the network, let $fanout(n_i)$ denote the *fanout list* for the node which is the set of blocks in Σ having n_i as an input node, and let $fanin(n_i)$ denote its *fanin list* which is the set of blocks with n_i as an output node. Thus,

$$fanout(n_i) = [\Omega_j : n_i \in INP(\Omega_j)]$$

and

$$fanin(n_i) = [\Omega_j : n_i \in OUT(\Omega_j)].$$

At each level in the hierarchy a directed graph $G(V, E)$ is constructed with a vertex for

every block in that level and a directed arc from vertex v_i to v_j if an output node of block Ω_i is an input node to block Ω_j. We will then say that G represents the partitioned MOS circuit Ω at that level. A directed graph is said to be *strongly connected* if for every pair of vertices v_i and v_j there exists at least one directed path from v_i to v_j and at least one from v_j to v_i. A maximal strongly connected subgraph is called a *strongly connected component* (SCC) [Rei 77]. It can be determined with time complexity $O(|V|+|E|)$, where V is the set of vertices and E is the set of arcs in the graph G. After the SCC have been determined, the edges going into the blocks from outside of the SCC would be redirected from the blocks to the SCC. The edge redirection method is shown in Figure 3.3.

A simple use of the depth-first search technique on the digraph determines a labeling of the vertices of an acyclic digraph G=(V, E) with integers 1, 2, ..., $|V|$, such that if there is a directed edge from vertex i to vertex j, then $i<j$; such a labeling is called a *topological sort* of the vertices of G. The time complexity is also $O(|V|+|E|)$, since every edge is traversed once and the procedure TOPOSORT is called once for each vertex.

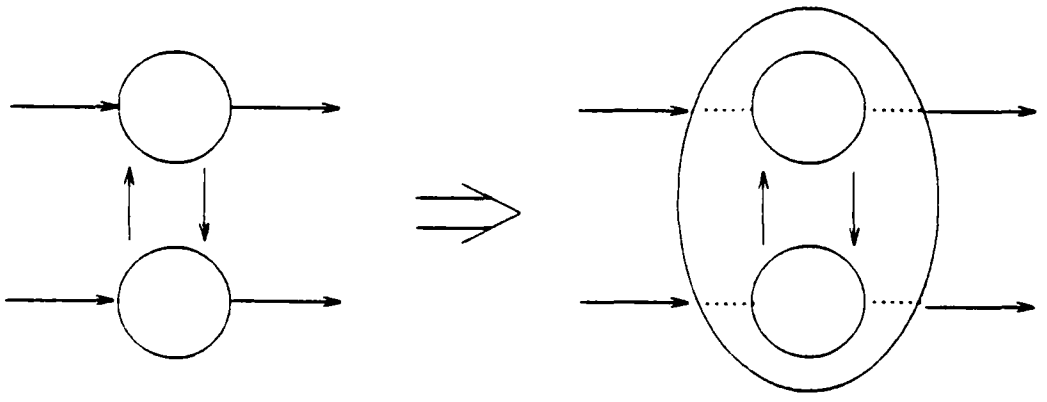


Figure 3.3. SCC edge redirection.

3.3. Delay Operation

Consider a block $\Omega_i \in \Sigma$ in the partitioned circuit $\Omega(N, M, \Sigma)$. Let $n_{in} \in INP(\Omega_i)$ be some input node to Ω_i and $n_{out} \in OUT(\Omega_i)$ be an output node of Ω_i. Let $d(n_{in}, n_{out})$ represent the *maximum* delay when a signal travels from n_{in} to n_{out} through the block Ω_i. The primary objective of the delay operation on block Ω_i is to determine $d(n_{in}, n_{out})$ for every pair of nodes (n_{in}, n_{out}) such that $n_{in} \in INP(\Omega_i)$ and $n_{out} \in OUT(\Omega_i)$.

3.3.1. Multiple Rise/Fall Delay

When overlapping input transitions cause an output transition, the delay is calculated by a procedure specific to an element type. For each block we calculate an equivalent input sequence S_{eq}. For example, consider overlapping input transitions in a NOR gate. We compute $S_{eq} = S_1 \vee S_2$, where S_1 and S_2 are the input sequences to the NMOS-driver gates. Since only one NMOS-driver transistor needs to be ON to make the output 0, the delay time is determined by the fastest rising input. On the other hand, the output rising delay time is determined by the slowest falling input of the PMOS-driver transistors. We define the block delay as $t_d = \Delta t_1 + \Delta t_2$. For each path, two propagation delays are calculated: one for rising-path start, and one for falling-path start. Blocks along a given path may be inverting, noninverting, or indeterminate. For indeterminate blocks, as in the case of the exclusive-NOR gate, both rise and fall delay are calculated and the worst case is assumed. In the case of blocks within an SCC, we start with the latest arrival input node and accumulate the delay times along the path until we encounter a cycle. An example is given in Figure 3.4 to illustrate the delay times accumulation inside an SCC. During the signal propagation period we record the worst rise delay for a PMOS-driver block, fall delay for a NMOS-driver block, and both rise and fall delays for a pass block [Al-H 85].

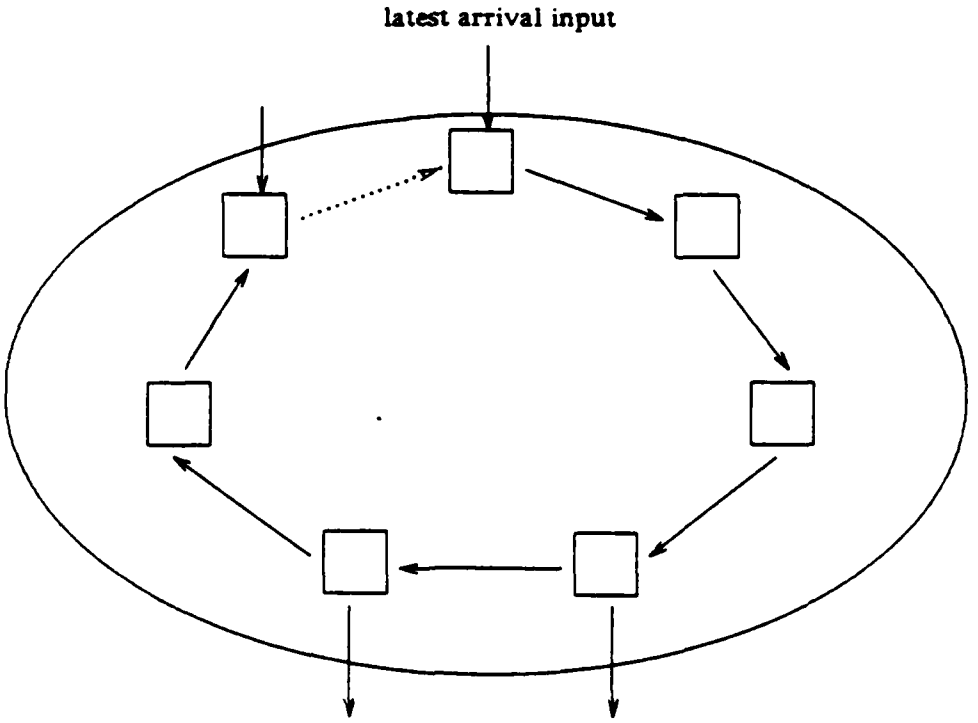


Figure 3.4. Delay times accumulation method inside an SCC.

3.3.2. Equivalent Capacitance

As depicted in Figure 3.5, each MOS transistor has five separate voltage-dependent capacitances coupling its four electrodes. To simplify the calculations of switching times all capacitance effects are lumped into a single total capacitance C_t which is connected to the output node of each inverter or gate [Hod 83]. Voltage-dependent effects of junction capacitance are removed by defining equivalent linear capacitances C_{eq}, which require the same change in charge as the nonlinear capacitors for a transition between two voltage levels, V_1 and V_2. With $V_2 > V_1$,

$$C_{eq} = \frac{\Delta Q}{\Delta V} = \frac{Q(V_2) - Q(V_1)}{V_2 - V_1} \equiv K_{eq} C_{j0}.$$

The depletion-layer capacitance per unit area C_{j0} of an abrupt n^+p junction is

$$C_{j0} = \left(\frac{q \epsilon_n N_A}{2\phi_0}\right)^{1/2}.$$

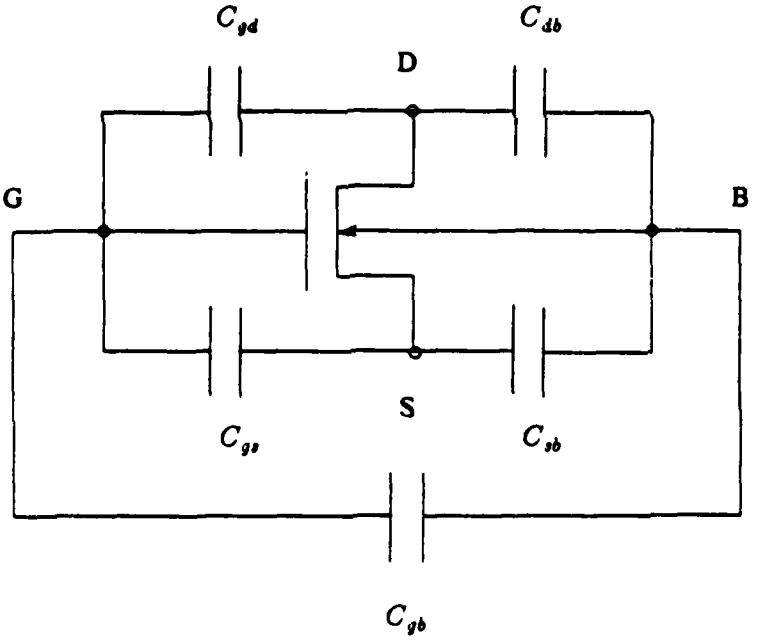


Figure 3.5. MOS transistor capacitance.

where q = electronic charge, ϵ_s = permittivity of silicon, and N_A = acceptor impurity concentration.

The built-in junction potential ϕ_0 is calculated from physical parameters as follows.

$$\phi_0 = V_T \ln\left(\frac{N_A N_D}{n_i^2}\right),$$

where V_T = the thermal voltage, N_D = donor impurity concentration, and n_i = the intrinsic concentration.

K_{eq}, the dimensionless constant used to relate C_{eq} to C_{j0} for specified values of V_1 and V_2, is calculated as follow.

$$K_{eq} = \frac{\phi_0}{(V_2 - V_1)(1-m)}\left[\left(1 - \frac{V_2}{\phi_0}\right)^{1-m} - \left(1 - \frac{V_1}{\phi_0}\right)^{1-m}\right],$$

where m value depending on what type of junction in the transistor.

Sidewall capacitances cannot be ignored for modern MOS processes; the sidewall capacitance per unit area is higher than C_{j0} because the n^+ source and drain abut the p^+ field diffusion.

Adequate accuracy is achieved by taking the sidewall area as the product of diffusion perimeter P and the junction depth X_j, neglecting the curvature of the sidewall and the gradient in field doping.

$$CJSW = C_{j0} X_j P$$

The capacitance between the drain and substrate, C_{db}, is the sum of the junction capacitance $Area * C_{j0}$ and the sidewall capacitance $CJSW$. The parasitic capacitance between the gate and drain C_{gd} is the product of lateral diffusion LD and the width of the transistor W,

$$C_{gd} = C_{ox} LD \ W,$$

where C_{ox} is the oxide capacitance between the gate and channel of the transistor. Finally, the lumped capacitance C_t is made up as follows: $C_t = K_{eq} C_{db} + C_{gd}$.

3.3.3. Equivalent Resistance

The transistor is approximated as a resistor in a lumped RC model. The equivalent lumped device capacitance is calculated and lumped with the load capacitances at the output node. The load and device capacitance are assumed to be constant, and bootstrapping effects are not simulated. The equivalent resistance is a function of input slew rate and $\beta(=W/L)$ i.e., $R_{eq} = \beta f(\Delta_{in})$. More specifically we take $R_{eq} = (\Delta t_1 + \Delta t_2)/(1.6\ C)$. To further illustrate this equivalent resistance concept, consider the following example, shown in Figure 3.6. The delay of the above circuit will be calculated in the following transformed circuit, shown in Figure 3.7, where $C_{eq} = C_2 + \frac{R_1}{R_{eq}}(C_1 + C_2)$.

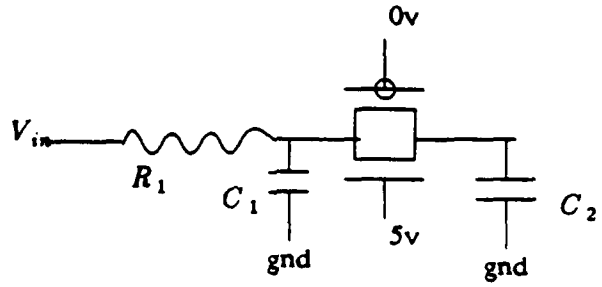


Figure 3.6. Pass transistor with interconnection resistance.

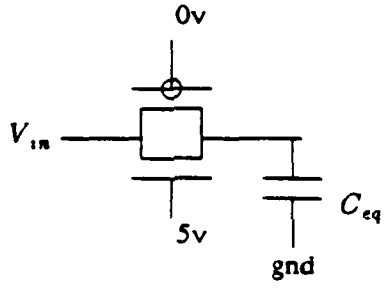


Figure 3.7. Equivalent pass transistor.

3.3.4. Primitive Cells Simulation and Transformation

For CMOS circuits, knowing the delay characteristics of three different circuit primitives is sufficient, within reasonable limits of accuracy, to compute delay through any general PMOS-driver block, NMOS-driver block, or pass-block. The three primitives are simulated using an accurate circuit simulator SPICE2 for various input slew rates, and the delay values are extracted and stored in delay tables. This can be done in a pre-simulation phase. During simulation, JADE then maps a PMOS-driver block, NMOS-driver block, or pass-block into one of the three primitives and obtains the appropriate delay value through a table lookup method, interpolating when necessary. The configurations of the three basic primitives are shown in Figure 3.8, 3.9, 3.10 respectively. Previous timing simulator for NMOS circuits, namely

MOSTIM [Rao 85], has used two additional primitives, i.e., type 4 shown in Figure 3.11 and type 5 shown in Figure 3.12. We can transform the primitive 4 circuit, by using the Elmore equivalent capacitance concept, to the primitive 2 of the delay operator as shown in Figure 3.13. Here $C_{eq}=C_1+C_2(1+\frac{R_{pass-1}}{R_{drv-p}})$ when C_2 is being charged, and $C_{eq}=C_1+C_2(1+\frac{R_{pass-0}}{R_{drv-n}})$ when C_2 is being discharged. R_{pass-1} : equivalent resistance of the pass transistor when transmitting logic "1". R_{pass-0} : equivalent resistance of the pass transistor when transmitting logic "0". We can transform the primitive 5 circuit, according to the Elmore equivalent capacitance concept with some modification, to the primitive 3 of the delay operator as shown in Figure 3.14. Here $C_{eq}=C_2(1+\frac{C_2}{(C_1+C_2)}\times\frac{R_{drv-p}}{R_{pass-1}})$ when C_2 being charged up, and $C_{eq}=C_2(1+\frac{C_2}{(C_1+C_2)}\times\frac{R_{drv-n}}{R_{pass-0}})$ when C_2 being discharged.

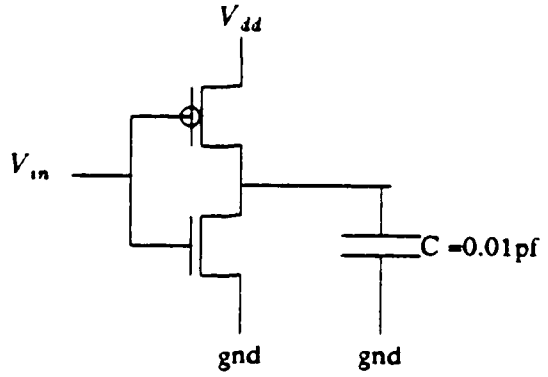


Figure 3.8. Primitive 1 of the delay operator.

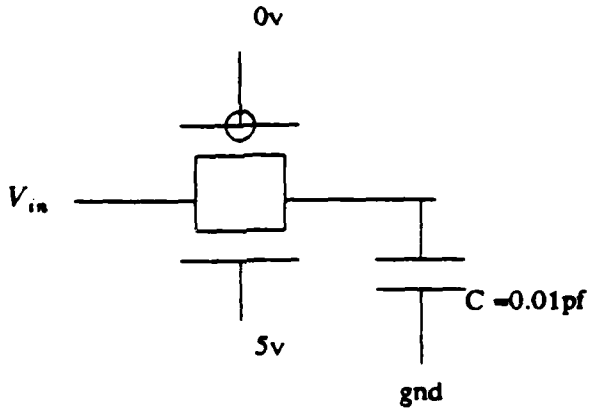


Figure 3.9. Primitive 2 of the delay operator.

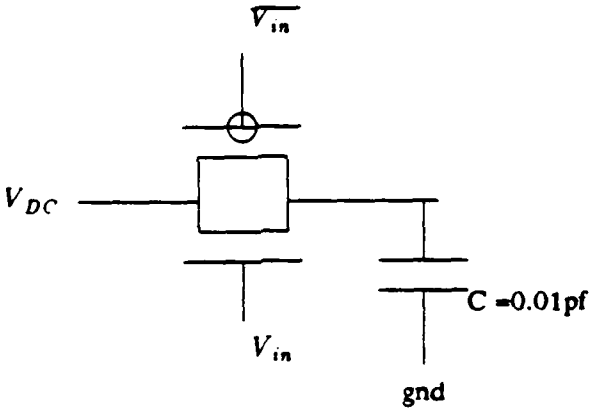


Figure 3.10. Primitive 3 of the delay operator.

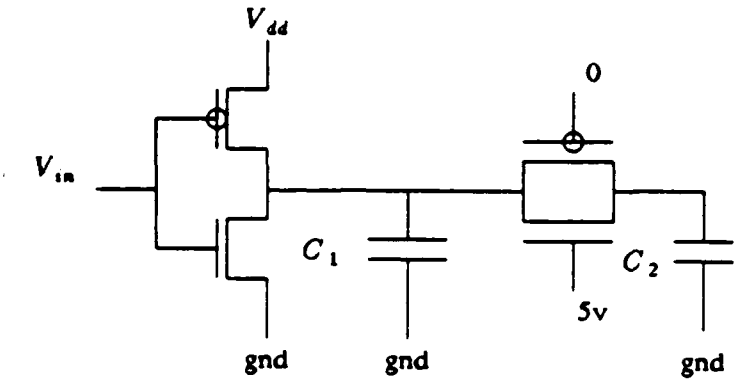


Figure 3.11. Primitive 4 of the delay operator.

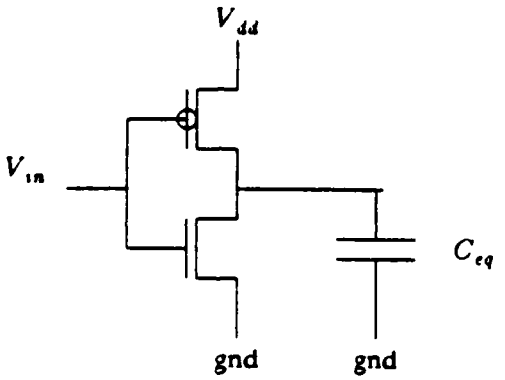


Figure 3.12. Equivalent circuit of primitive 4 of the delay operator.

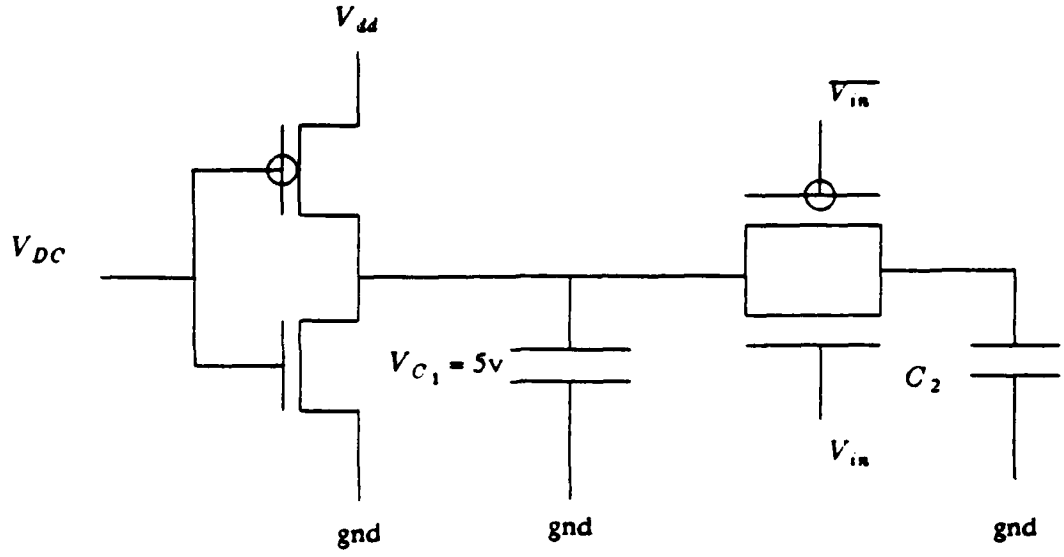


Figure 3.13. Primitive 5 of the delay operator.

3.3.5. Dynamic Windowing Relaxation Algorithm

We review the use of a special windowing technique to simulate those blocks within a strongly connected component (SCC). A transition interval for a node is the time interval during which the node is in the intermediate state u. The entire time interval $(0, t_f)$ is partitioned into windows by taking the initial and final times of each transition interval I as the

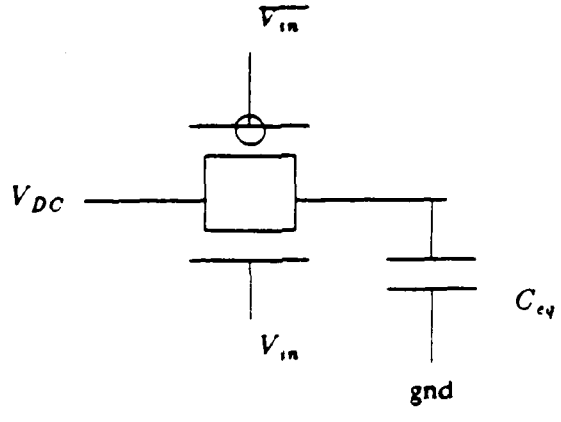


Figure 3.14. Equivalent circuit of primitive 5 of the delay operator.

boundaries of the windows. Associated with each transition interval I for a node n_k is a fanout list of blocks. The main idea is to use the *windowing* technique in the waveform relaxation procedure as suggested in [Whi 84], wherein it is shown that the number of iterations is exponentially proportional to the size of the time interval of simulation. This algorithm splits the entire time interval into windows such that all pairs of signal transitions take place entirely within one of these windows. This is achieved by maintaining a sequential list of time intervals which is dynamically updated by the inclusion of the transition intervals at the output nodes of the block as the algorithm progresses. It is event-driven, i.e., only those blocks that are active in a window are processed during that window and the fanouts of the output nodes of these blocks are scheduled for processing in the future. We require no a priori ordering of blocks within an SCC because of event-driven property.

3.3.6. Time Scaling

In this section we show how we can compute the delay values for nonstandard primitives from the delay tables for standard primitive computed in the previous section. For any primitive i, where $i = 1, 2, 3$, we can write the first-order differential equation for the output waveform in the simplified form:

$$\frac{dV_o(t)}{dt} = \frac{1}{\sigma_i} f_i(V_o(t), V_{in}(t))$$

where

$\sigma_1 = R_D C_L$. R_D : equivalent resistance of the driver block.

$\sigma_2 = \sigma_3 = R_{pass} C_L$. R_{pass} : equivalent resistance of the pass block and C_L : load capacitance.

Each pass transistor is characterized by two resistance tables, one is used when the transistor is transmitting logic "0" and one is used when it is transmitting logic "1".

Let $g_i(\Delta_{in})$ denote the delay functions tabulated as a function of input slew rate Δ_{in}. for

standard primitive i, where $i = 1, 2,$ or 3.

Now we can compute the delay for the nonstandard primitives from those computed for standard primitives as follows:

1) Compute the scaling factor $\alpha = \sigma_i / \sigma_{i_s}$, where σ_{i_s} is the RC time constant for the standard primitive i.

2) If $i = 1, 2,$ or 3, then obtain $\Delta t_o = \alpha g_i (\Delta_{i_n} / \alpha)$, for $o = 1, 2$.

3.4. Critical Path Evaluation

Let $G(V, E)$ be a directed graph representing the given partitioned CMOS circuit $\Omega(N, M, \Sigma)$. Suppose the strongly connected components of G have been identified and collapsed into single vertices thereby resulting in an acyclic *condensation* digraph $\tilde{G}$. Each vertex in $\tilde{G}$ could now correspond to a set of blocks in a feedback-loop in Ω. Since $\tilde{G}$ is acyclic, there exists a *topological ordering* on its vertices. This would induce a corresponding ordering on the blocks in Σ. We will refer to this ordering as a topological ordering on the blocks of Σ.

For each circuit node $n \in N$ we define the *earliest start time*, $E(n)$ [Pre 77] as follows:

(1) Initially, set $E(n) \leftarrow 0$, for each node n in the graph.

(2) Scan the blocks in Σ in *forward* topological order. Let Ω_t denote the block currently under consideration. For each pair of nodes $j \in INP(\Omega_t)$ and $k \in OUT(\Omega_t)$ set $E(k) \leftarrow \max\{E(k), [E(j) + d(j, k)]\}$, where $d(j, k)$ is the delay-time from input-node j to output-node k through block Ω_t.

We also define the *latest completion time*, $L(n)$, for each circuit node $n \in N$ as follows:

(1) Initially set $L(n) \leftarrow E(n)$, for all nodes n in the graph.

(2) Scan the blocks in Σ in *reverse* topological order. Let Ω_i denote the block currently under consideration. For each pair of nodes $j \in INP(\Omega_i)$ and $k \in OUT(\Omega_i)$ set $L(j) \leftarrow \min\{L(j), [L(k) - d(j,k)]\}$.

A node n in the circuit is said to be *critical* if $E(n) = L(n)$. A block is said to be a *critical block* if it has both a critical input node and a critical output node. A critical path in the graph $\tilde{G}$ is defined to be a directed path of maximal length consisting entirely of vertices corresponding to critical blocks in the circuit.

3.5. Timing Analyzer

In general, the data structure and delay computation algorithms are the same as in the timing simulator. The difference is that a timing analyzer uses an input independent approach; both its strength and its weakness stem from this difference. In simulation, a specific set of input signals is applied to a circuit. The simulator predicts the functional behavior of the circuit so that the designer can see if it matches the desired behavior. In timing analysis, the only goal is to see if the circuit meets its timing specifications; since the function of the circuit is not being tested, specific signal values are largely irrelevant. The timing verifier attempts to find a combination of values that results in the worst possible timing behavior. The input-independent approach provides the main advantage of timing verification over simulation. By considering all possible signal values at once, a timing analyzer is guaranteed to locate any performance bottlenecks in a single run. In contrast, the effectiveness of a set of simulations depends on the selection of input data; pathological conditions may be undetected if they are not triggered by the particular inputs fed to the simulator. The value-independent approach is also responsible for the main difficulty in timing verification. When a timing verifier ignores specific signal values, it may report critical paths that can never occur under real operating conditions. These false paths tend to camouflage the real problem areas. In practice, all timing verifiers include a few mechanisms that designers can use to restrict the range of values

considered by the program. JADE provides a mechanism called i/o specification, which is used to handle pass transistor structures by controlling information flow through switches.

The major advantage of combining the timing simulator and the timing analyzer is that all worst-case critical paths can be uncovered from timing analysis and the accurate calculation of delay can be provided by the simulator with the corresponding input vector. The data structure, partitioning, and ordering algorithms are identical for both timing simulation and timing analysis. The worst-case equivalent block resistance and input slew rate are always assumed for the timing analysis purpose to get the worst-case critical paths. In the timing analyzer the pass block delay is calculated by using the Elmore RC time constant, while in the timing simulator the delay is calculated from table-lookup and time scaling techniques.

3.6. Examples

Several representative examples have been simulated in JADE to show the applicability and the accuracy of the algorithm. The first example is an exclusive-nor (XNOR) circuit shown in Figure 3.15. The circuit is treated as a pass block since it connects to no power supply node and no ground node. The node's voltage waveforms are shown in Figure 3.18 for comparison. The dash line waveforms are from JADE and the solid line waveforms are from SPICE2. The second example is a D-latch circuit shown in Figure 3.16. The circuit contains a feedback loop. The strongly connected components are simulated using the dynamic windowing relaxation algorithm. The results from both the SPICE2 and the JADE simulation are shown in Figure 3.19. The third example is a 4-bit adder shown in Figure 3.17. The circuit is defined hierarchically. The 4-bit adder is composed of two 2-bit adders, and the 2-bit adder is composed of two 1-bit adders. The 1-bit adder is defined by nine NAND gates. In this example we show that JADE can handle the hierarchical structure without flattening and takes advantage of the hierarchical approach. The four summation bits and the carry-out bit waveforms are shown in

Figure 3.20. The execution time and memory usage are shown in Table 3.1.

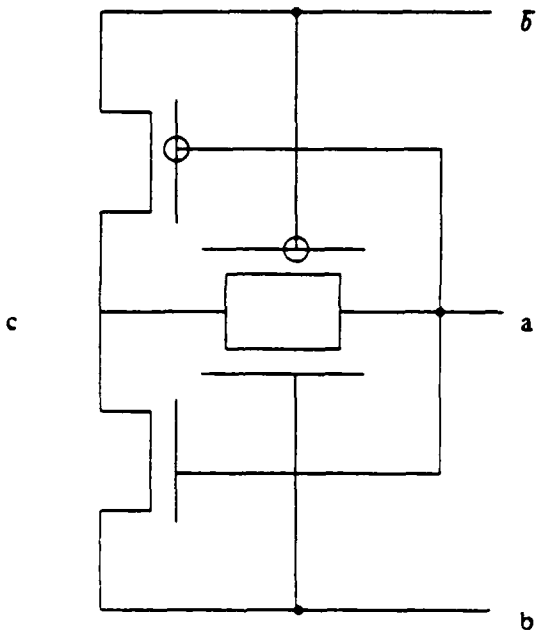


Figure 3.15. XNOR circuit.

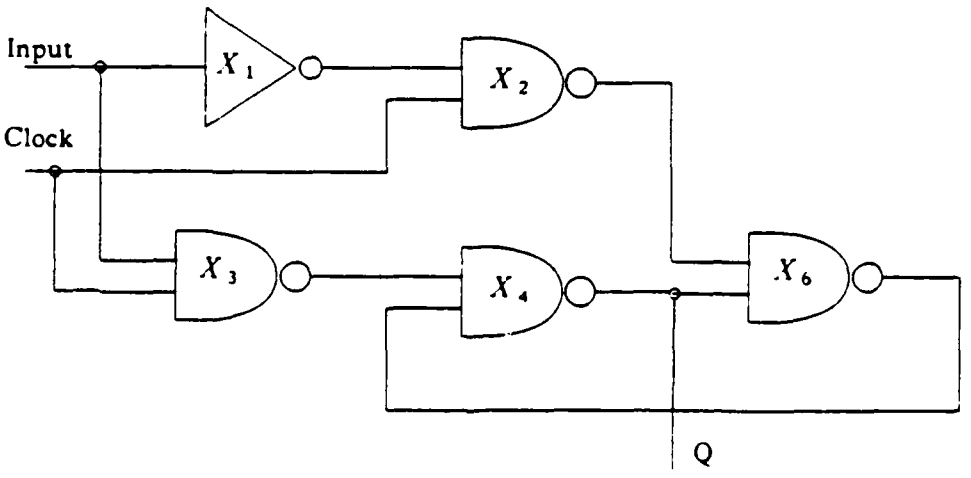


Figure 3.16. D-latch circuit.

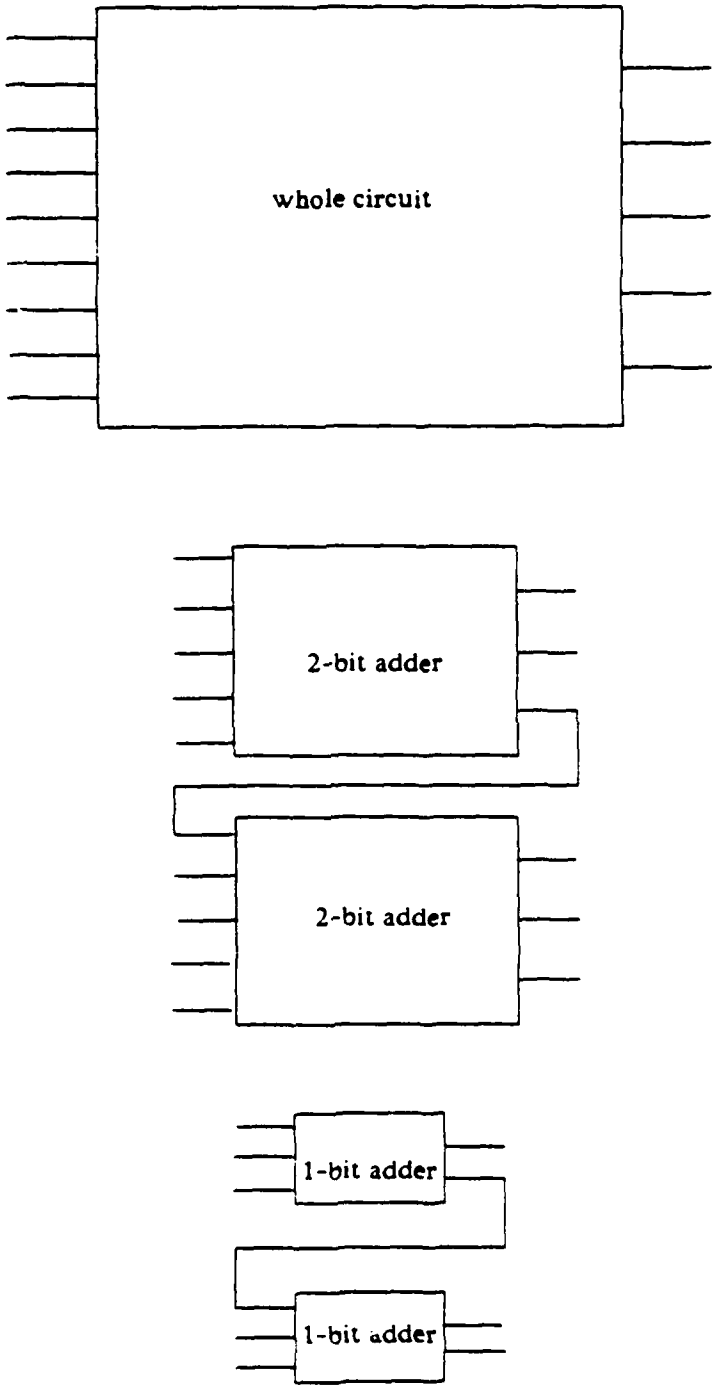


Figure 3.17. 4-bit adder block structure.

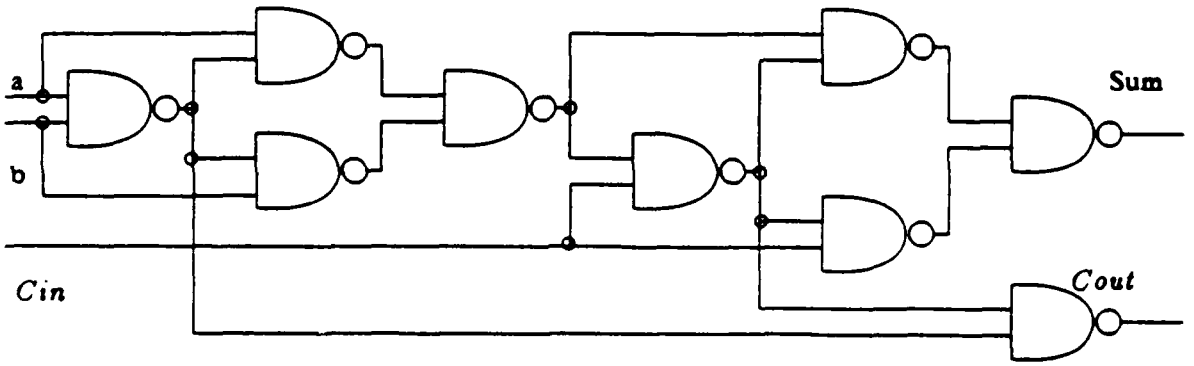


Figure 3.17. Continued.

	XNOR	D-Latch	Adder(4-bit)
# transistors	4	18	142
SPICE2(second)	70	320	5469
JADE(second)	10	24	60
Memory Space(KB)	9	13	31

Table 3.1. Execution time and memory usage performance.

3.7. Summary

What we have presented here is a new type of timing simulation. With regard to performance, JADE compares very favorably to other attempts at timing simulation such as [Nom 82], [Tam 83], [Ter 83], [Hwa 86] in several respects. First, the hierarchical approach saves a considerable amount of memory space. JADE is able to pinpoint critical paths. Second, a small number of delay lookup tables and equivalent circuit transformations relieve the pre-simulation phase workload. Finally, the frame data base and LISP environment make it compatible with the expert system approach to optimizing circuit performance in the next chapter.

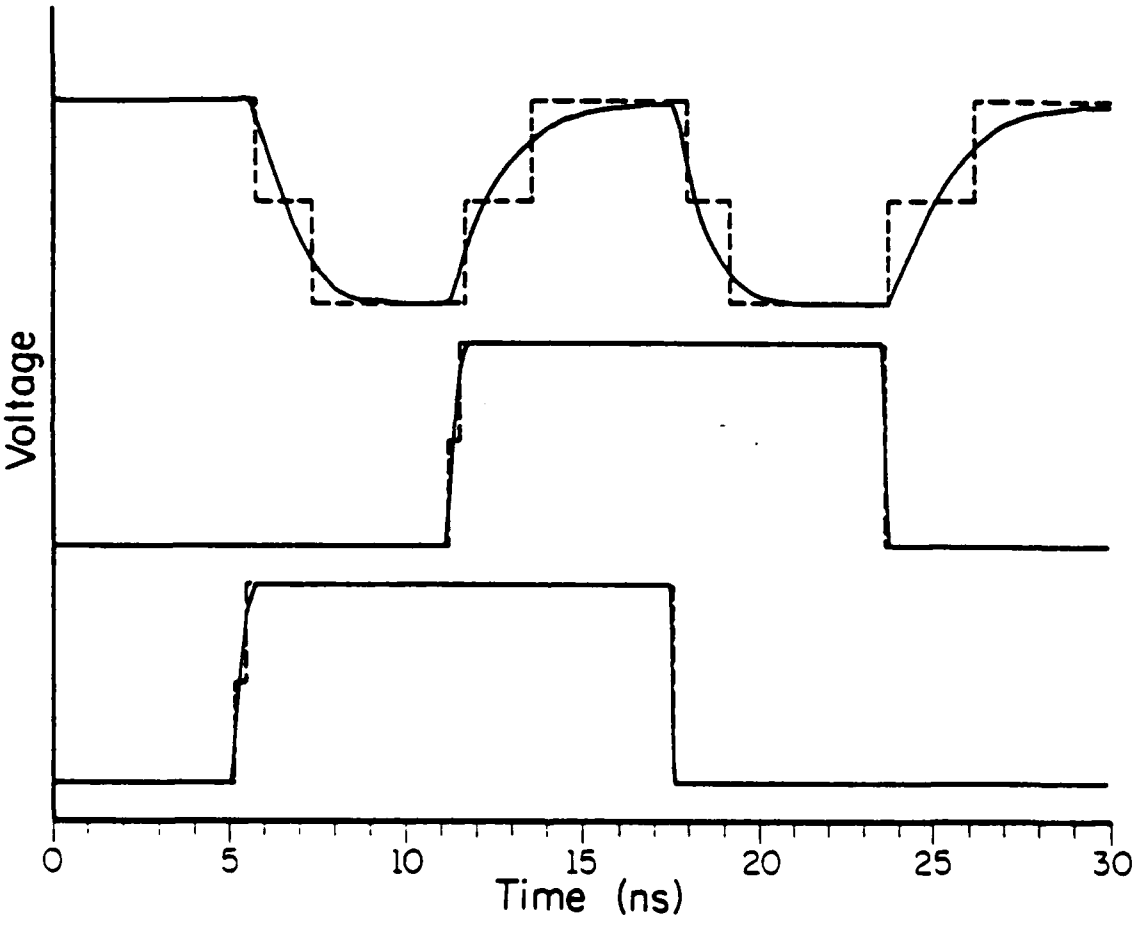


Figure 3.18. XNOR circuit output waveforms.

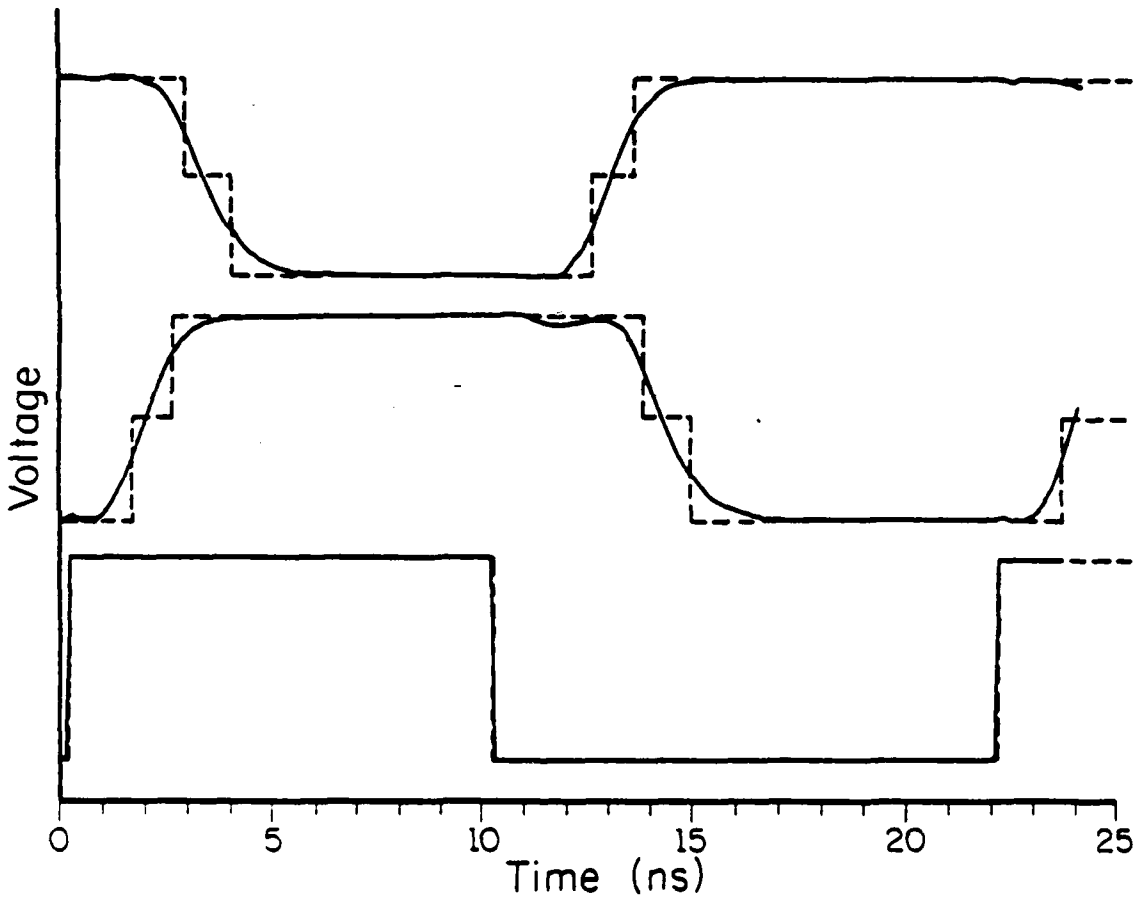


Figure 3.19. D-latch circuit output waveforms.

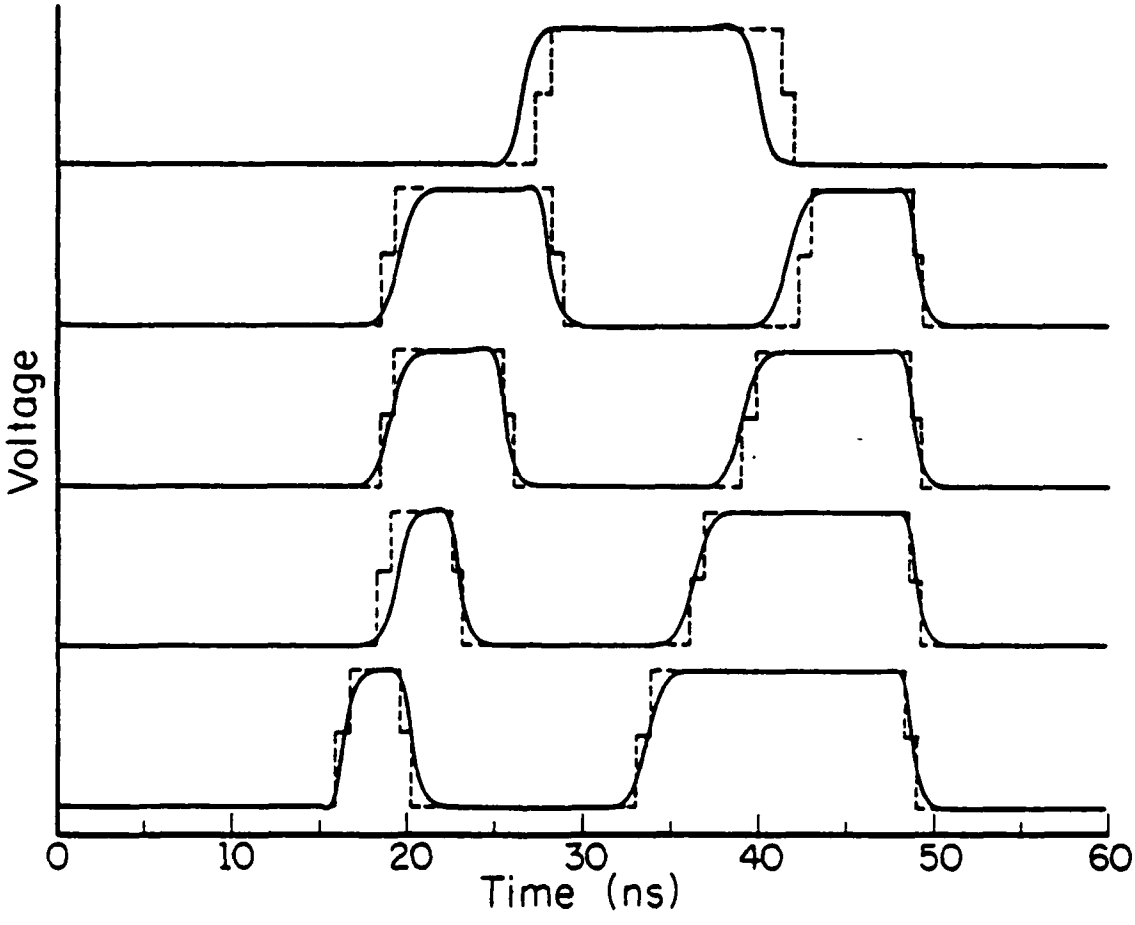


Figure 3.20. 4-bit adder output waveforms.

CHAPTER 4.

CMOS VLSI TIMING OPTIMIZER

4.1. Introduction

The unit size or fixed ratio transistor geometries circuits produced by silicon compilers is not appropriate for high performance VLSI design. A better timing performance can be reached by adjusting the transistor sizes properly. The circuit optimization techniques and algorithms appropriate for use in VLSI design should have low computational complexity, yet must be accurate enough for high performance circuit design. While some stages of chip design can be handled well by algorithms, special cases like circuit and clock tuning require the application of knowledge-based expert systems [Kow 85]. If a satisfactory algorithmic solution to a problem is known it should be used. When such a solution is not known or where a problem has many "special cases" which dominate the run time of the algorithm, the problem is a good candidate for an expert system based approach [Buc 84]. The conventional design iterations that involve circuit extraction, simulation, designer's check and modification are excessively tedious and time consuming. We propose a rule-based timing optimizer using a switch-level timing simulator to calculate the timing performance. iJADE combines the algorithmic solution and rule-based expert system approach. The algorithm provides a simple first order approximation of the proper device sizes for each transistor. The optimization rules of the expert system are then applied to fine tune the circuit performance. The circuit is described in terms of transistors of different types and sizes, along with the estimates of interconnect resistance and capacitance. Before the timing optimization there will be hundreds of critical paths that are a much as a factor of two slower than the slowest path in the final design. With careful circuit design the speed of a chip can be improved, noise can be eliminated and clock skew can be reduced to a

tolerable level [Sho 82].

Given a circuit and its limit on the chip area A , the circuit optimizer's task is to minimize the propagation delay of the circuit. Although power dissipation is another important performance measure of VLSI circuits, we will assume that the constraint on power dissipation can be represented by the constraint on the chip area of the active devices. This assumption is reasonable since power dissipation is proportional to the active chip area.

To optimize the timing performance of a circuit, we introduced a hierarchical circuit optimizer iJADE. iJADE uses an accurate switch-level timing simulator or a switch-level, input-independent timing analyzer to identify the worst-case critical paths. iJADE attempts to solve timing problems by adjusting the size of the transistors or by inserting buffers for large capacitive loads. As a closed-loop system iJADE calculates delay, searches for critical paths, and modifies the circuit iteratively until the specifications are met. iJADE is composed of five parts : (1) the switch-level timing simulator JADE [Lai 87a], (2) the switch-level timing analyzer, (3) the path delay reduction algorithms, (4) a rule-based expert system, and (5) the frame data base. iJADE reads the circuit description with the design specifications and points out critical paths after accurate calculation of propagation delay times. The path delay reduction algorithms and rules are then applied to optimize the timing performance. After circuit optimization the timing performance is reevaluated to check whether further optimization is necessary. This process gets repeated until the design meets the timing specifications or the design can not be improved further.

4.2. Optimization

Several techniques have been proposed for the optimization of large circuits on VLSI chips. Most techniques allow only a few design parameters each logic circuit, which severely limit the number of logical circuits which can be optimized. Ideally, the timing model may use

only one global parameter per logic circuit. An example of such a model is a MOSFET circuit where all the FET gates are adjusted in proportion to a single design parameter - transistor channel width W.

VLSI circuit optimization has gained much more importance with the advent of silicon compilation. In fact, high performance silicon compilers must take timing into account. Probably the most common way to adjust the timing of MOSFET integrated circuits is the adjustment of the FET gates [Rue 77].

Several authors have studied optimization work of this nature. General purpose optimization packages such as DELIGHT [Nye 81] and APLSTAP [Bra 81] perform much of the work in the optimization process. They iteratively improve the design solution as a designer would, but by employing nonlinear optimization algorithms and choose the next solution point more accurately and efficiently than a human could. The key advantage is that an optimal solution is reached. However the optimization process tends to be computationally expensively for a number of reasons. First, since the optimization package is general purpose in nature, it cannot exploit properties of digital MOS logic and use algorithms which would be more problem specific and hence potentially faster. Second, because the optimization package is isolated from the circuit's data base, communicating solely via the simulator, there is no mechanism to access the circuit's structural description or to embed additional information in the data base which could assist the optimization. This hampers the application of more efficient algorithms that would require such provisions. Third, the circuit's signal path delays must be determined fairly accurately; this generally entails the use of a device level simulator such as SPICE2, which is rather expensive computationally. For these three reasons, general purpose optimizers are typically restricted to circuits with at most about thirty design parameters.

In an effort to address larger designs, some researchers have investigated more specialized techniques [Rue 77]. By using a resistive model for transistors and neglecting the changes in a

logic gate input capacitance induced by sizing its transistors. these works were able to simplify the optimization problem greatly. They reformulated the original problem. a minimization subject to nonlinear constraints. as an unconstrainted minimization. This allows for much simpler optimization algorithms. leading to fast convergence times. Nonetheless. the simplifications needed to reformulate the problem seriously reduce the accuracy of both the power minimization and the satisfaction of the delay constraints. making the approach inappropriate for high performance circuit design.

Other authors have aimed for fast computation times by simplifying the logic gate models and the optimization techniques. Examples are TV [Jou 83] and ANDY [Tri 86]. Both tools use resistor models for transistors instead of the computationally expensive device level models. Heuristics. rather than nonlinear optimization algorithms guide the sizing of transistors in critical paths. In particular. TV speeds up paths by widening the transistors of slow logic gates. while Andy uses a fixed sizing ratio from a gate when a chain drives a large capacitive load and then reduces the speed and power consumption of gates that are not on the critical paths. Although these approaches are computationally fast enough to be applied to large circuits. our problem domain requires more accuracy and efficiency. The resistor model is not accurate enough for high performance design. and the heuristic sizing rules are inexact. The heuristics are an attempt to decouple the sizing problem to the point where individual logic cells can be sized independently of the bulk of the circuit. and consequently these rules take limited account of interactions among cells and signal paths.

Timing. power consumption. and chip area are not independent. We need to do some tradeoff between them to make a good circuit. It is indeed necessary to increase transistor sizes beyond the minimum size to overcome capacitive loadings from interconnection wires [Kan 81]. While combinational systems are faster than synchronous systems. they are troubled by a phenomena called *race*. which could result in the system assuming an error state. A *critical race* occurs when a change in input causes the system to leave a stable state for

another stable state, with the final result depending on the way in which signals propagate through logic gates. Critical race can sometimes be eliminated by restricting the tolerable variation in the propagation delay of the input signals.

The difference among the input arrival times of multiple input gates can cause serious glitch problems and even logic failures. The input paths of the gates in critical paths should be tuned such that all the inputs almost arrive at the same time. For each block in the critical paths when the difference among each input arrival time is greater than a fraction of the block delay, the *equalize* function is activated.

$$\text{IF } \Delta Input\_Arrival\_Times > c * DELAY(BLOCK_i) \text{ THEN}$$

$$(\text{equalize } block_i).$$

where c is a constant. The algorithm tunes the slowest input path and the magnitude of the differences decides how much the scale should be increased.

4.2.1. Difficulties in Optimization

Once the critical paths of a design are known, it may not be clear how to best improve the performance of the present circuit [Jou 83]. Difficulties typically arise for three reasons:

1. If a block early in the path is sped up, another path may become the worst path, with a new delay just less than that of the original path.
2. If a block late in the path is sped up, the critical path may branch out near the end before the improved block, leaving other blocks in the path with approximately the original delay.
3. By speeding up a given block, previous blocks may be slowed down significantly due to increased loading from the changed block, reducing the overall speedup.

4.2.2. Optimization Techniques

A rule-based system *SOCRATES* which optimizes combinatorial logic has been presented [Coh 85]. The system performs substitutions of equivalent gate configuration in order to reduce the area of the implementation. The logic is optimized by performing a serious of local transformations on the circuit. Given a critical path, there are several general circuit optimization techniques. [Gla 84]. Due to the complexity of the problem, we consider only the following two methods. We use the backward tuning approach to deal with the problem due to increased loading from the changed block, and the iteration approach to handle the dynamic situation of critical paths.

1. Change the widths and lengths of various transistors in the schematic. This is the most straight forward technique and has the minimum impact on the layout. The β ratio for a CMOS inverter is optimized in regard to propagation delay time, independent of the size of the successive inverter, if β is equal to the square root of the electron-hole mobility ratio [Kan 83].

2. Insert one or more buffer stages between a high impedance source and a low impedance load. There is an optimum number of buffer stages that can be decided by the ratio of loading capacitance to driver gate capacitance.

4.2.3. Algorithm Approach

The switching speed of a CMOS gate is limited by the time taken to charge and discharge the load capacitance C_l. The delay is affected by the input slew rate, the configuration of transistors within the block, and the output load. The RC timing model has generally been used in the transistor sizer to estimate the delay of the circuit. This allows for much simpler optimization algorithms, leading to fast convergence times. Nonetheless, the simplifications needed to reformulate the problem seriously reduce the accuracy of both the power minimiza-

tion and the satisfaction of the delay constraints, making the approach inappropriate for high performance circuit design. However a design truly optimized for delay time is seldom practical. This is because the silicon area increases very rapidly when the minimum delay time is approached [Lee 84].

The function of the optimizer is to speed up the critical paths by tuning the transistors in slow blocks and/or subcircuits while satisfying other constraints. The circuit starts with the minimum size transistor configuration. The sizing scale is decided by a first order approximation of the RC time constant.

If $T_D > T_{max}$ Then

$$\alpha = \frac{T_D}{T_{max}} \frac{C_{new-load}}{C_{old-load}}$$

$$\beta = \frac{C_{\delta parantic}}{C_{new-load}}$$

$$Scale = \alpha(1+\beta).$$

Where, T_{max} is the maximum delay specified by the user and T_D is the critical path delay. $Scale_1$ is the first sizing factor. $C_{new-load}$ is the new loading capacitance and $C_{old-load}$ is the old loading capacitance. $C_{\delta parantic}$ is the increased parasitic device capacitance contributed from the α sizing operation. The transistor sizing operation starts with the output stage of the circuit and propagates the changed capacitance backward to the driving stage for the coming sizing operation.

4.2.4. Application of Knowledge Engineering

Knowledge based expert systems (KBES) have been applied to various fields and area both in CAD and non-CAD [Zip 83], [Lob 84], [DeM 85], [Wil 85]. The success or failure of knowledge engineering techniques depends greatly on the nature of the problems to which they

are applied. This section attempts to identify a number of characteristics that should be observed before committing a problem to KBES applications [Bir 86].

1. The first and the most important criterion to consider is whether the application under consideration has an algorithmic solution. Applications which have algorithmic solutions will generally run slower with knowledge engineering techniques than with an algorithmic solution. This is due to the large amounts of computational overhead in implementing knowledge engineering techniques.

2. Most of the problems which do not have algorithmic solutions have large problem spaces. It is virtually impossible to enumerate all possible cases and represent them in a coherent way. Another characteristic of these problems is the lack of good evaluation function to measure the quality of partial solutions. This is mainly due to a number of factors that must be considered at the same time.

Often the class of difficult problems where knowledge engineering techniques are applicable are problems where an optimal solution is not necessary. A solution that satisfies a few parameters is sufficient.

4.3. Rule-based Expert System

For many CAD problems, either an efficient algorithm is not known or handling "special cases" makes the algorithm inefficient. In these situations rule-based expert system technology offers a possible solution. The purpose is to build CAD systems incorporating an expert designer's knowledge in order to cope with the growing complexity of digital system design.

The optimal algorithmic solution to the timing optimization in CMOS VLSI is not known. Furthermore the problem has many "special cases" which dominate the run time of the algorithm. Therefore this problem becomes a good candidate for a rule-based expert system approach. For example, the optimal width ratio of PMOST and NMOST for minimum delay is

calculated and implemented by the algorithmic approach, while the widths of the precharge PMOST and inverter buffer's NMOST in Domino CMOS circuits are decided by the expert system. We combine analytic tools with a rule-based expert system to take advantage of timely on-line information to administer the rules and to verify the actions.

iJADE is an expert system. By that we mean that it is an program designed to provide expert-level solutions to complex circuit timing problems. The main function of the expert system approach is to speed up paths by widening the transistors of slow blocks and/or subcircuits while maintaining other constraints satisfied. There are two main parts to an expert system: a knowledge base and an inference mechanism.

4.3.1. Knowledge Base

The knowledge base is the program's store of facts and associations it "knows" about a subject area such as electronic circuits. A critical design decision is how such knowledge is to be represented within the program. In the program, the knowledge is represented by conditional statements, or rules, of the following form [Buc 84]:

IF: There is evidence that A and B are true,

THEN: Conclude there is evidence that C is true.

We refer to the antecedent of a rule as the premise or left-hand-side (LHS) and to the consequent as the action or right-hand-side (RHS).

4.3.2. Inference Mechanism

The decision as to which particular representation or control technique to use depends highly on the type of the task. For example, the forward chain reasoning technique is one of the general techniques that is commonly used in constructive tasks whereas backward chain

reasoning is mainly used for analysis tasks. In the case of constructive tasks the starting conditions are the givens and one searches for a solution, forward chain reasoning is very appropriate. In the case of analysis tasks, one starts with a solution and wants to find the initial conditions which attributed to the solution, backward chain reasoning is very appropriate. Because of the situation-action nature in the circuit design environment, forward chaining is used in the inference mechanism. The application of operators to those structures in the data base that describe the task-domain situation - to produce a modified situation - is called reasoning forward. The object is to bring the situation, or problem state, forward from its initial configuration to one satisfying a goal condition.

4.3.3. Topological Pattern Recognition for Functionality

Typical VLSI circuits are very complex systems. Accordingly, their design is directed by a wide variety of considerations. The standard approach to mastering that complexity is to discipline the design process by adopting a number of rules that, taken together, are conductive to correct designs. Such a set of *a priori* selected rules, of which geometrical design rules are one aspect, is commonly referred to as a "methodology". Different methodologies are better suited to achieving different goals, but all share a basic characteristic: they help isolate the designer from the details, allowing the design effort to be concentrated on higher level abstractions. The feedback node of a static latch in Figure 4.1, for example, could be instantiated using the following expressions:

```
(defun feedback-node (node)
    (AND
        (input node SCC)
        (output node block)
        (member block SCC)
    )
)
```

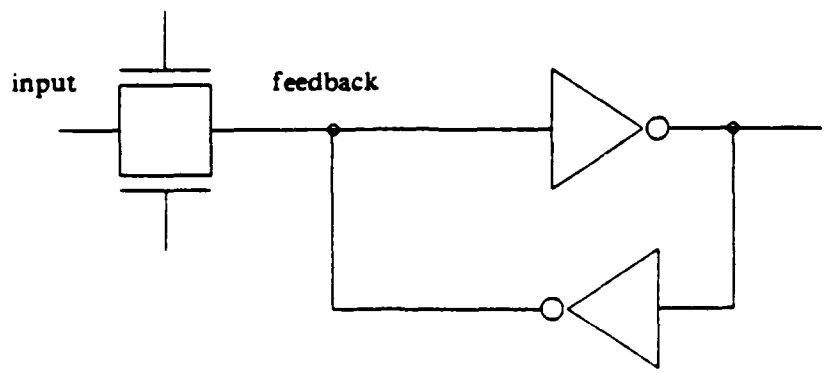


Figure 4.1(a). Static latch.

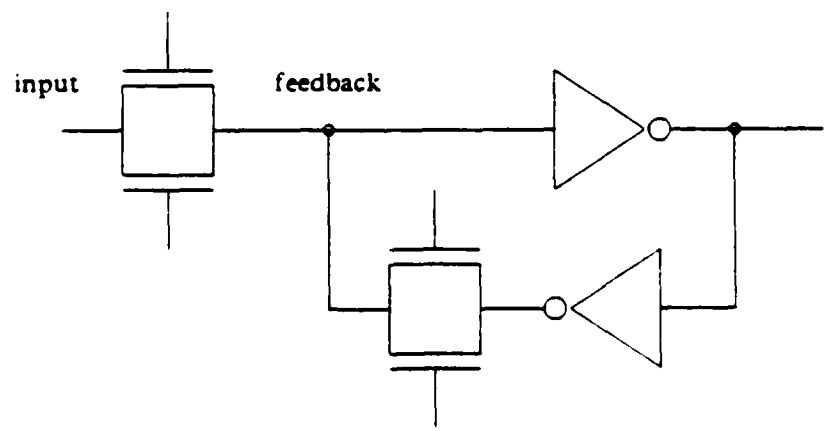


Figure 4.1(b). 2-phase static latch.

4.3.4. When are rules useful

Since production systems support a different model of computation than the model supported by imperative programming languages, the question of when it is appropriate to use this new model naturally arises. The general answer to this question is that production systems

should be chosen when it is desirable to have the interpreter determine the order in which subroutines are invoked. Thus production systems are appropriate when the task to be solved has some property that makes it difficult to write explicit subroutine calls into the code. The best example of tasks for which this is the case are knowledge-based expert systems. Expert systems are programs that solve problems that are ordinarily solved by human experts.

4.4. Optimization Rules

Knowledge bases are generally developed in several stages. First, "book knowledge" of the problem is codified as a set of situation-action rules: interviews with experts then fill in knowledge gaps and refine current knowledge. Then many example problems are given to the expert system, and experts closely examine and validate the results. Often, errors are found through the examples, and new rules are added to the system to correct the error situations. The circuit optimization rules implemented in the expert system are listed as follows:

(R1). Density and Power dissipation:

Let W_p equal W_n when similar structures are cascaded [Wes 85].

(R2). Noise-margin:

By setting $W_p / W_n = \mu_n / \mu_p$, we can have good noise margin.

(R3). Clock skew:

If the clock signals are delayed equally before arriving at the subblocks, they will be perfectly synchronous. Use identical clock drivers to equalize the load so as to minimize the clock skew.

(R4) Capacitance effect:

In a driver block the transistor closest to the output is the smallest, with transistors increasing in size the nearer they are to V_{ss}. The decreasing switching times are attributed to

the dominance of the capacitance term in the RC time constant of the gate.

(R5). Input rise time:

a. If a block is driving a large load, or has a small transistors then only very slow input rise times will affect the block's delay.

b. If a block is driving a small load, or has very large transistors, its delay will be more sensitive to the rise time of its input.

c. Fast inputs put the gate in an RC response mode where the output waveform's switching time is governed by the gate's effective output resistance and capacitive load.

d. Slow inputs place the gate in a limited mode where the gate's gain increases the sharpness of the waveform's transition.

(R6). Pass transistor:

The PMOS transistor transmits a logic "1" well, and the NMOS transistor transmits a logic "0" well.

(R7). Symmetrical VTC:

If $V_{tp} = -V_{tn}$ and $\mu_n \frac{W_n}{L_n} = \mu_p \frac{W_p}{L_p}$, we can have a symmetrical voltage transfer characteristic.

(R8). Speed:

The minimum propagation delay occurs when $W_p / W_n = \sqrt{\mu_n / \mu_p}$ [Kan 81].

(R9). Delay:

The optimized design requires that the rise-time of the input voltage be approximately equal to the fall-time of the input voltage [Lee 84].

(R10). TTL interface input buffer:

While a logic TTL "0" can be considered low enough to switch the CMOS gate, a TTL logic "1" (say 3.2V) is too close to the inverter threshold voltage - which is 2.5V for a balanced inverter when V_{dd} = 5V - to guarantee reliable operation. Thus, when a CMOS gate is driven by a TTL gate the channel width of NMOS transistor should be about five times greater than the channel width of the PMOS transistor [Ann 86].

(R11). TTL interface output buffer:

Because the output swings from V_{dd} to V_{ss}, there is no problem in reaching the TTL "high" and "low" logic levels. The only constraint calls for the driver to sink and source the necessary amount of current in both logic states. We can express the channel resistance of the MOS transistor in the linear region as:

$$R_{channel} = \frac{L}{W \mu_{n/p} C_{ox} (V_{gs} - V_{Tn/p})}.$$

By combining channel resistance with the current constraints we have:

$$\frac{L}{W} = \frac{V_{\circ}}{I_{sink}} \mu_{n/p} C_{ox} (V_{gs} - V_{Tn/p}).$$

(R12). Pass transistor:

a. If a gate is driven by PMOS pass transistors only, its PMOS block should be larger than usual.

b. If a gate is driven by NMOS pass transistors only, its NMOS block should be larger than usual.

(R13). Buffer:

The buffer is to be inserted when the sizes of the first stage logic gates have been increased. The number of buffer stages is determined by the formula $n = \left| \ln \frac{C_{gate-first-stage}}{C_{gate-minimum-size}} \right|$. The merit of this operation is to reduce the capacitance loading of the driving stage and to increase the speed of the circuit. If the logic gates of the output stage have many transistors, we could

place the buffer pairs in the output stage to reduce the transistor active area. A even number of buffers are inserted such that there is no change in the signal polarity.

(R14). Latch:

An alternate 2-phase static latch is depicted in Figure 4.1(b). When the pass transistor controlling signal ϕ is high and ϕ overlaps it due to skew, the D input and feedback signal will "fight" to determine the new value on the input of the latch. One method of reducing this effect is to make the feedback inverter a weak "trickle" inverter so that the input signal will override the effect of this signal. A "trickle" inverter is constructed by using transistors with a lower β than a regular "minimum" sized inverter.

(R15). Latch:

A nand gate latch is shown in Figure 4.2. We can obtain a sharp and symmetrical transfer characteristic curve if the two nand gates have the same transistor sizes.

(R16). Transmission line:

To avoid reflections at the fan-out points, the current supplied by the driver and into the transmission line must be equal [Bak 86]. The current into the transmission line is given by

$$I = 0.5\frac{V_{dd}}{Z}. \tag{4.1}$$

Since the transistor is operating at the linear region, it can supply a current of

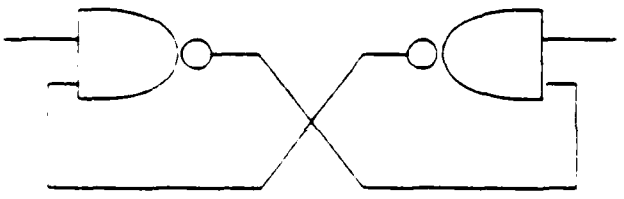


Figure 4.2. 2-nand gate latch.

$$I = \mu C_{ox} \frac{W}{L}[(V_{dd} - V_T)\frac{V_{dd}}{2} - \frac{V_{dd}^2}{8}]. \quad (4.2)$$

The optimal transistor sizes are obtained by setting the currents in Equations 4.1 and 4.2 equal.

$$\frac{W}{L} = [\mu C_{ox}(0.75 V_{dd} - V_{dd})]^{-1}.$$

(R17). Special gate:

The exclusive-nor circuit shown in Figure 4.3 should have the same width for all the transistors. and always has one output buffer to restore voltage levels.

The rule interpreter searches through all the rules one after another. This search continues until either the condition of one of the rules becomes true or a match is unavailable. at

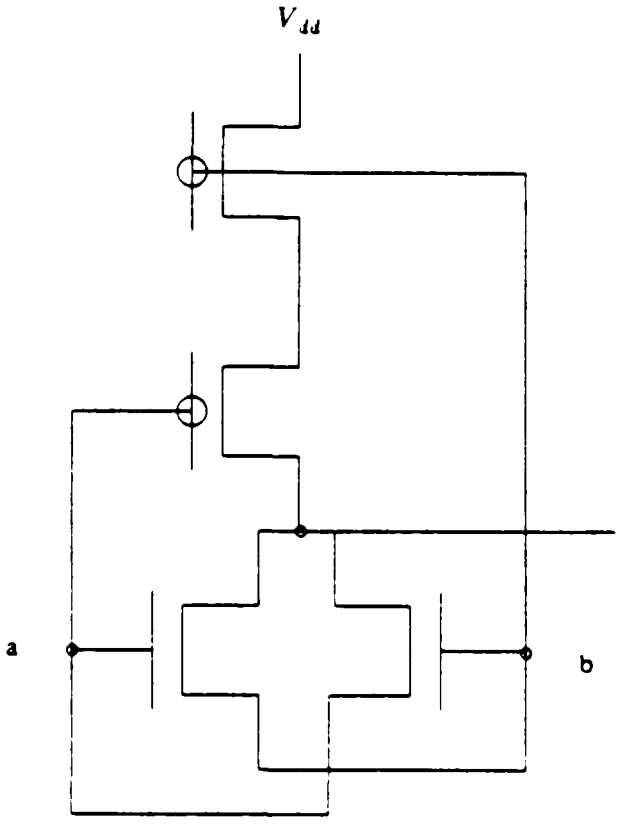


Figure 4.3. XNOR circuit.

which point backtracking occurs and the alternatives of the last choice are tried. The backtracking operation is implicitly achieved by reducing the loading contributed from the blocks not in the critical paths. The size of the transistors may be decreased if they are driven by the blocks not in the critical paths. Two side benefits of this operation are to reduce the power consumption in the non-critical paths and to equalize input arrival times.

4.5. Experimental Results

Several circuits have been optimized by running iJADE. Two examples are presented below. The first example is a chain of four inverters. The optimum scaling ratio between two neighboring stages for the minimum delay is e [Mea 80], [Gla 85]. The output loading capacitance C_L is set such that the size ratio between the two succeeding inverters in the chain is e: $C_L = C_g * e^N$, where C_g is the gate capacitance of the first inverter and N is the number of stages in the inverter chain. In 2 μm CMOS technology the minimum size of the first stage transistor is set to be 4 μm. The rise and fall delay of the inverter chain with the scaling factor of e are 3.6ns. respectively. The total active area calculated by summing transistor areas in each stage of the inverter chain is $575\mu m^2$. A comparison with the optimization result from iJADE shows that the delay time can be kept the same by consuming only $292\mu m^2$ of active area. The rise delay of the inverter chain sized by iJADE is 3.6ns and the fall delay is 3.8ns. The PMOST sizes in the inverter chain are (4, 8, 20, 51) μm and NMOST sizes are (4, 6, 15, 38) μm, respectively.

The second example is a Domino CMOS 6-input AND gate shown in Figure 4.4. Dynamic CMOS logic circuits are now an integral part of CMOS VLSI technology. The reason for this wide acceptance is that the circuit is compatible with any CMOS processing technology and the circuit is fast since only NMOS transistors determine the delay. When many dynamic CMOS gates are cascaded to make a Domino CMOS configuration the circuit is capable of executing

Figure 4.4. Four stage inverter chain.

many data path operations of a microprocessor. We apply the scaling technique presented in [Sho 85] to speed up the circuit and to save some of the active device area. The size of the transistors from ground to output are (69, 60, 53, 46, 41, 36, 31) μm , respectively. The size of the PMOS precharge transistor is 60 μm , while the sizes for the output buffer being 53 μm for PMOST and 8 μm for the NMOST. The propagation delay t_{plh} is 2.2 ns and t_{phl} which restricted by the clock period is 8.2 ns .

The third example is a 4-bit ALU with 142 transistors. The performance of the circuit as manufactured in industry is compared with that optimized by iJADE. The output waveform of the circuit optimized by iJADE shows fewer glitches in the SLATE simulation [Yan 80]. The propagation delay times as determined by the falling edge of the waveforms are smaller in the iJADE-produced circuit configuration. The active area used in iJADE's circuit is only 1750 μm^2 as compared with 3118 μm^2 in the industry circuit. The output waveforms with three different transistor sizes for the 4-bit ALU circuit (dot: minimum size, dash: sized in industry, solid: sized by iJADE) are shown in Figure 4.6(c). It should be noted that the circuit with minimum transistor width will produce a large glitch that causes incorrect logic operations. The execution time performance of iJADE for the above examples is listed in Table 4.1.

The results show that iJADE performs CMOS circuit optimization competitively or even better than the design from industry in terms of glitch avoidance, propagation delay time, and chip area. Also the hierarchical approach saves a lot of memory space and computation time. The overall execution time appears to grow linearly with the number of transistors.

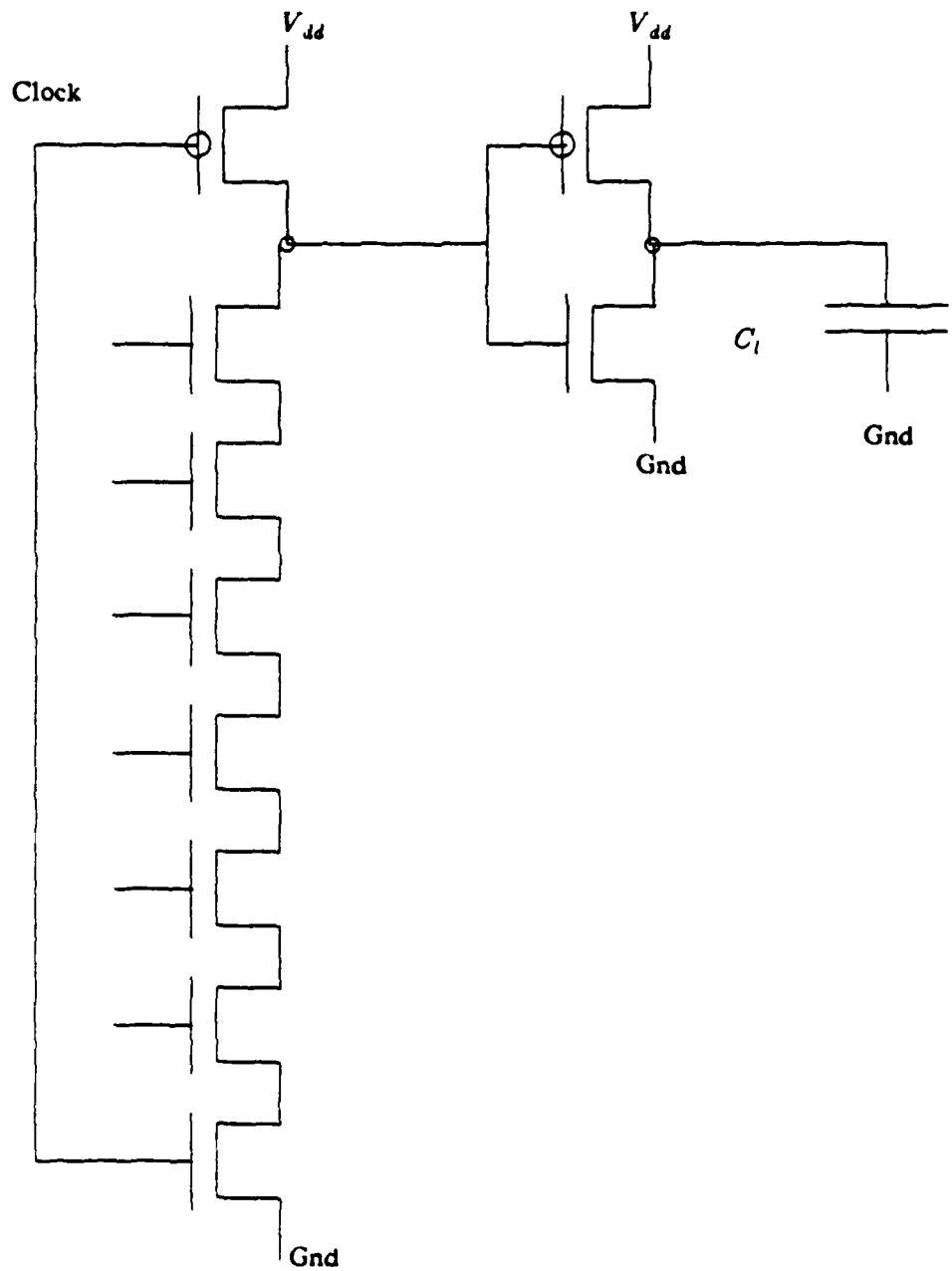


Figure 4.5. Dynamic domino CMOS circuit.

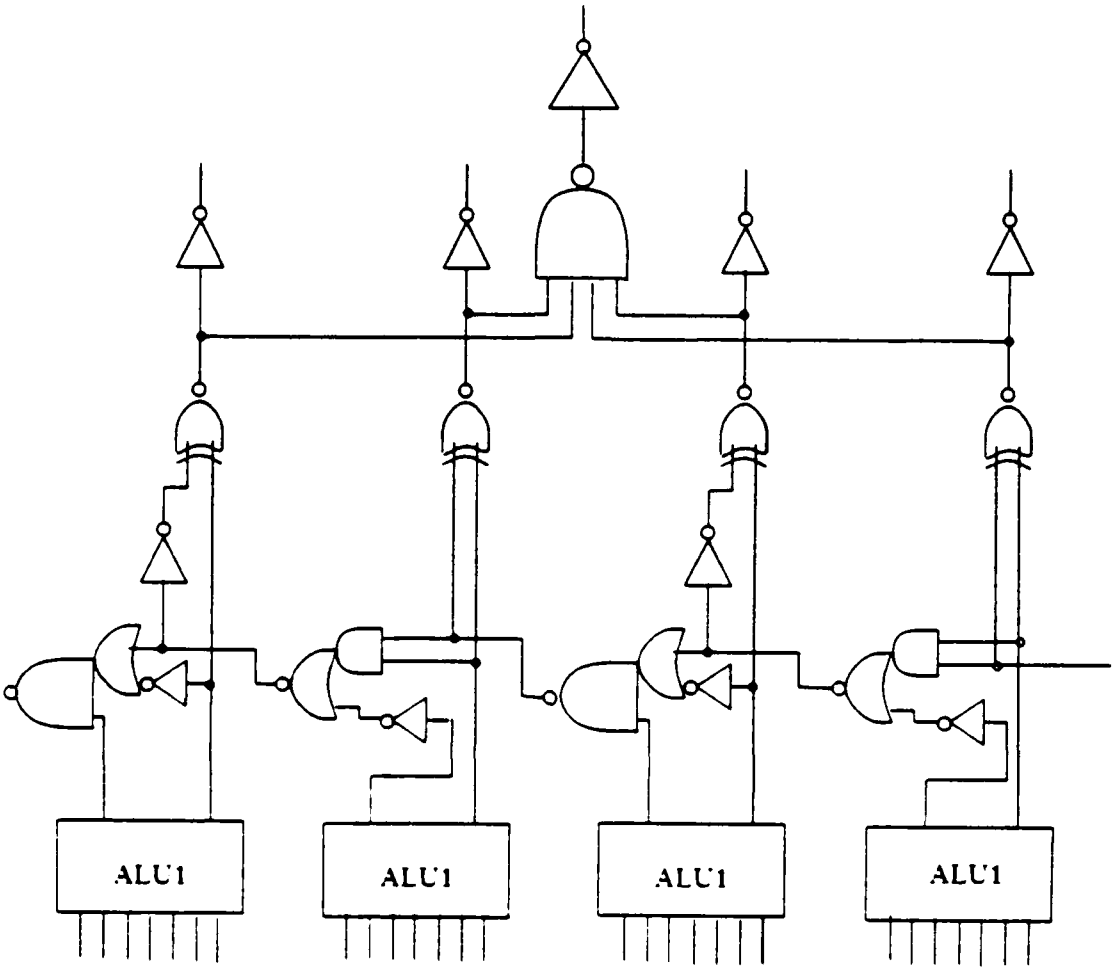


Figure 4.6(a). 4-bit ALU circuit.

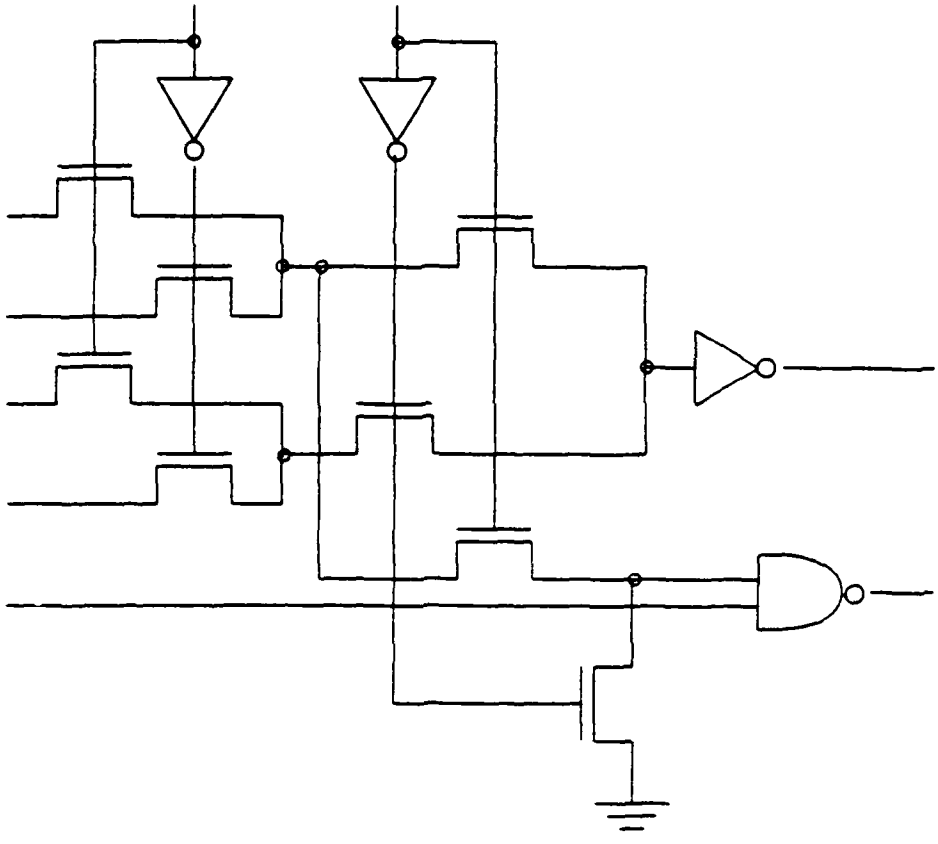


Figure 4.6(b). ALU1 circuit.

	Inverter Chain	Domino CMOS	4-bit ALU
transistors	8	10	142
iterations	2	2	3
time(seconds)	157	177	823

Table 4.1. The execution time performance of iJADE.

4.6. Summary

A design aid system iJADE has been developed for high performance CMOS VLSI circuit generation. It has four novel features. First, it combines analytic tools with a rule-based expert system to exploit timely on-line information to administer the rules and to verify the actions. iJADE is a closed loop performance evaluation and design parameter refinement system. Secondly, iJADE contains an accurate switch-level timing simulator that detects the latches, traces the clock paths and simulates the accurate signal waveform of each node in the circuit for tuning a synchronous circuit to satisfy the clock timing constraints. Thirdly, iJADE optimizes the circuit performance by tuning the critical paths found by the worst-case timing analyzer. Lastly, the hierarchical structure for both circuit and programming saves a lot of memory, speeds up the calculation and can define the potential problem subcircuit clearly. Experimental results on a 4-bit ALU and register paths show that iJADE-optimized CMOS circuits perform competitively or even better than hand crafted designs from industry in terms of glitch avoidance, propagation delay time, and chip area.

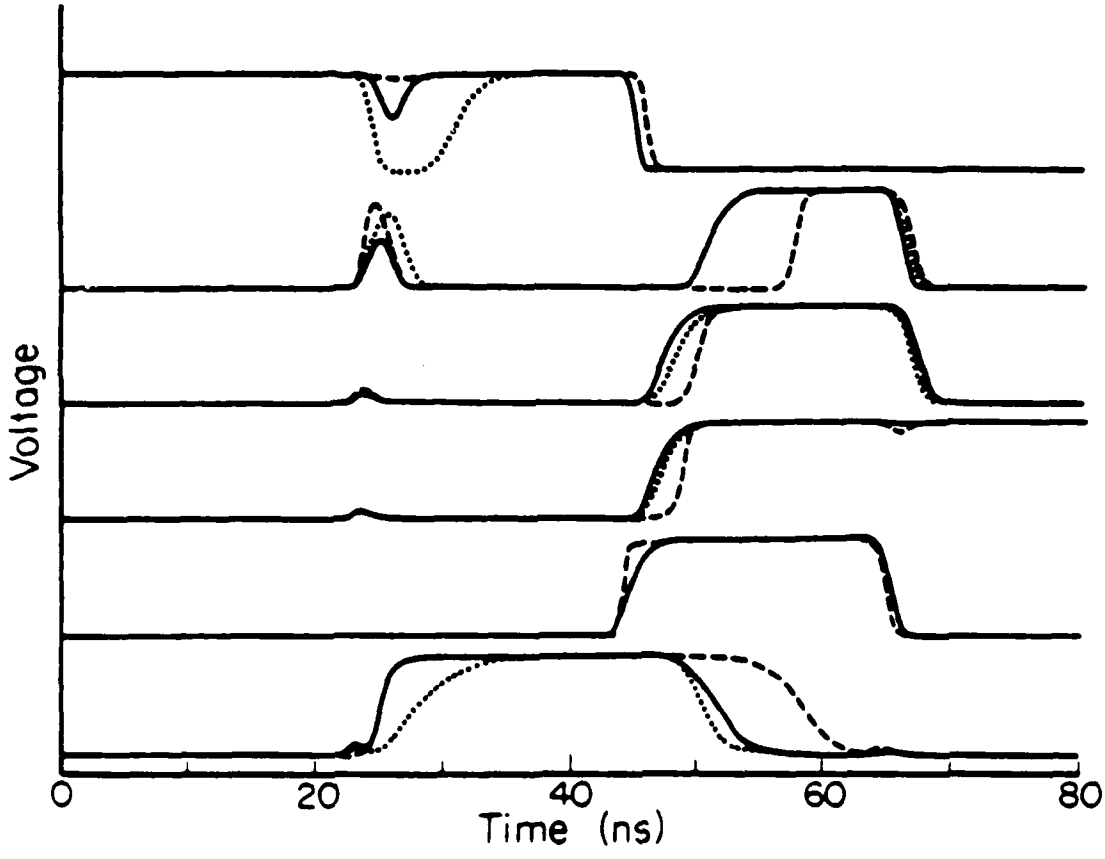


Figure 4.6(c). 4-bit ALU output waveforms.

CHAPTER 5.

TIMING ERROR CORRECTOR FOR VLSI SYNCHRONOUS CIRCUITS

5.1. Introduction

For one signal to influence another they must intersect, not only in space, but also in time. Choreographing the thousands of signals in a VLSI system so that each part of the data arrives where it is supposed to, and when it is supposed to, is an extremely complex task. This is particularly true for high performance designs. One way to cope with this complexity is to employ a clocking methodology. Its role is to synchronize the orderly and the controlled flow of information in the digital system, as well as to reduce the complexity of the circuit design by restricting the valid logic forms. In a digital circuit, we must restore both logical values and time. This means, we must make sure when it is valid to interpret an analog voltage waveform as representing a logic value. As signals propagate through the network, they undergo delays that cannot be always precisely predicted. Since we need to be able to bring signals together in time, we must develop a technique for dealing with delay uncertainties. One method, "clocking," equalizes all of the delays by making them all equally bad. Clocks are used in a digital system to hold up a signal until it is time for it to begin to move through the next stages of logic. Registers are used in conjunction with the clocks so that a signal can be restored at a location until it is needed.

The clock pulse distribution has been a serious problem in VLSI design when logic elements are physically separated but need to be gated simultaneously [Gla 77]. Clock skews have caused major timing errors, and therefore, the use of precise clock waveforms has been [illegible] clock signals are traced out to get the real arrival time at each node. The latch

pairs are then identified to calculate the path delays between each latch pair. Therefore, the optimization of synchronous circuit performance is equivalent to the minimization of the critical path delay through combinational circuits.

Previous approaches [Hit 82], [Agr 82], [Ben 82], [Jou 83], [Cha 85], [Szy 86] have only detected the timing errors between latch pairs often using ideal clock waveforms. The simplest form of timing verification is *dynamic timing analysis* which consists of applying a timing simulation model of the circuit to a large set of test vectors at a variety of clock frequencies. Although in principle this approach can catch every timing problem, it is very expensive to perform. Moreover, it is difficult to construct a set of test data which is guaranteed to cover all relevant cases. Finally, a failed simulation provides little information as to why the circuit failed.

Static timing analysis attempts to avoid these problems by examining the circuit for a single clock cycle in a way which is independent of the logic values at the nodes in the circuit. Most systems of this type operate by enumerating all paths between the storage elements of the circuit. In many cases, the number of such paths is enormous and leads to long run times. TV [Jou 83] and TA [Hit 82] avoid these long run times by taking a PERT-like approach but are restricted to circuits that are expressed at a high enough level that their circuit graphs are acyclic within each clock phase. Crystal [Ous 85] allows circuits with cycles and provides a method of determining when each node in a circuit will respond to an external stimulus. Unfortunely, no mechanism is provided for checking that the signals arrive at their destinations "soon enough." Moreover, since Crystal deals with clock phases independently, it provides little help in dealing with signals that are in transit across phase boundaries.

In addition to the deficiencies listed above, none of these previous approaches is particularly well suited to repeated analyses with changed circuit or clocking parameters.

In this chapter, we introduce a new CAD tool [Lai 87c] for detecting the timing errors and correcting them through circuit optimization techniques. The tool has been developed to speed up the circuit and to meet the multiphase clock timing requirements. The main algorithm is:

1. Read the timing specifications and circuit file in SPICE format.

2. Calculate the parasitic capacitance contributed by the devices.

3. Trace clock paths and identify latches.

WHILE (data-base is modified) DO

4. Trace critical paths and shortest paths between latches.

5. Detect timing errors

5.1. Set-up time

5.2. Hold time

6. Correct timing errors

6.1. Clock paths.

6.2. Signal critical paths between latches

6.3. Signal shortest paths between latches

7. Update the data base

UNTIL (all specifications are satisfied)

8. IF success THEN (Print out results) ELSE (print out suggestions)

To detect timing errors, the hold time and the set-up time of each latch are checked with accurate signal waveforms. To correct the timing errors, two procedures are used: (1) resizing transistors in clock paths and (2) resizing transistors in critical and shortest signal paths.

5.2. Path Analysis

Simulation of logic networks using logic states and accurate timing information has long been one of the standard tools for design verification. In a detailed simulation of this type,

functional and timing specifications of the devices are calculated, and delays through interconnections are used to verify both correct functional behavior and correct timing behavior. The relative importance of function and timing varies with the type of system being designed. In high performance systems, timing verification is very important and must be done throughout most of the design process to guide design decisions. Detailed simulation can be used for this purpose, but it suffers from one major disadvantage - incompleteness. Simulation must be driven by specific test cases. Due to the large number of input combinations, however, it is impossible to simulate all cases. Any problem that is not exercised by the chosen test cases is likely to remain hidden until the late stages of the development process where a fix can be very expensive or might lead to reduced system performance.

We use the timing analysis approach discussed in Section 3.5, instead of simulation, to overcome the incompleteness problem. The delay modeling is based on the time-scaling method which depends on the input slew rate, loading capacitance, and the circuit structure. The timing analyzer uses the slowest slew rate of all the input signals and the worst-case block resistance to get the worst-case propagation delay times. The strongly connected components (SCC) are detected and treated specially in the timing analysis. The block with the earlist arrival input from outside of the SCC is analyzed first. The rest are analyzed in descending fanout order.

The previously described techniques can be grouped into two main categories: path enumeration techniques and block oriented analysis techniques. Path enumeration techniques are characterized by programs which start at certain "start points", and trace back through the blocks feeding the start points until a terminal point is reached. At this stage, the information is complete and the data for the path can be accumulated. Data accumulation can be done by simply adding up the delays. But path enumeration techniques tend to have long running times since the number of paths through a graph grows exponentially with the size of the graph. Block oriented analysis techniques are characterized by programs which start with signals at

storage element outputs. The blocks, which these signals feed, are then processed to find the latest and/or the earliest time at which the signal could propagate through them. The blocks are processed so that the times at the output of each block is only be calculated once. Block oriented analysis techniques tend to run much faster than enumerative path analysis techniques.

Here, we adopt the critical path approach of the block analysis techniques, because of the efficiency and the simplicity of the method. The critical path algorithm is executed in the corresponding acyclic graph with the SCC shrunk to a single node. The complexity of the algorithm is $O(E + V)$, where E and V are the number of edges and vertices of the corresponding acyclic graph, respectively. Since the delay analysis for the most part ignores function, some paths may be flagged as having timing problems when they, in fact, do not. We suggest that timing analysis only be used to find the worst-case critical paths. The real timing information should be found using a timing simulator with the appropriate input vectors.

5.2.1. Clock Path Tracing and Latch Identification

The clock paths are traced hierarchically, and switch-level timing simulation is done for each node along the paths. The clock paths can contain resistors and inverter gates. Only the initial nodes of the clock paths need to be specified, since the program can automatically trace the clock paths. A latch is detected when an SCC is connected to a clock node either directly or indirectly through logic gates. The function that defines a latch is as follows:

Whether an SCC with a triggering clock node is a latch or a flip-flop, depends on the triggering signal.

There are two types of latches normally used in digital circuits: (1) edge-triggered and (2) level-sensitive. An edge-triggered latch is only sensitive to a particular edge (rising or falling)

```
(defun latch (module)
    (AND
        (member module SCC)
        (OR
            (input clock module)
            (AND
                (input node module)
                (output node block)
                (input clock block)
            )
        )
    )
).
```

of the clock signal. The signal at the input must be stable before this clock edge arrives at the latch. Irrespective of when the data signal arrives at the input, it propagates to the output at the triggering edge. Every edge-triggered latch is considered a destination of a path leading to it. On the other hand, a level-sensitive latch is sensitive to both rising and falling edges of the clock. One edge opens the latch so that the data can propagate through it, while the other edge closes the latch. The content of a level-sensitive latch, at the closing edge, is retained in the latch while it is closed. The signal can appear at the output of a level-sensitive latch at any time while the latch is open. The delay of a path leading into a level-sensitive latch can be analyzed as follows:

(1) If the signal arrives at the latch after the closing edge, then it must wait until the next opening edge.

(2) If the signal arrives before the opening edge, then it must wait until the opening edge.

(3) If the signal arrives between the opening and the closing edges, then it continues to propagate as if the latch were only a flip-flop.

It is worth noting that an edge-triggered latch can be analyzed in exactly the same manner, if we assume that its opening and closing edges occur simultaneously

5.2.2. Critical Paths between Latches

For simplification of the path search and tuning algorithms, only the critical paths between latches are traced. The critical path algorithm is the same as in [Lai 87a], except here we start at the slowest arrival input node of each latch and end at a latch. Critical paths are found between a latch pair. Timing edges and clock pulse widths are found through simulation. The sequential circuit clock timing relation is depicted in Figure 5.1.

5.3. Timing Error Detection

Generally, there are two kinds of timing errors, the set-up time violation and the hold time violation. The user can define these two values to check timing errors. The default hold time value is set to the internal propagation delay of the latch.

5.3.1. Clock Waveform

The distribution of clock signals is a difficult problem and often gets complicated further when multiphase clock signals must be derived from the master clock with precise timing relationships. To detect timing errors due to the clock skews, timing simulation is used for clock path circuit analysis.

5.3.2. Set-up and Hold Time

The *set-up time* T_{su} is the time interval allowed for the inputs to stabilize before they are clocked into the latch. The *hold time* T_h is the time interval after the clock transition has occurred during which the inputs must be held stable. The timing error algorithms in [Ben 82] [Cha 85] were modified to check the set-up time and the hold time explicitly. The program also checks to insure that the clock pulse duration is greater than the internal propagation delay of

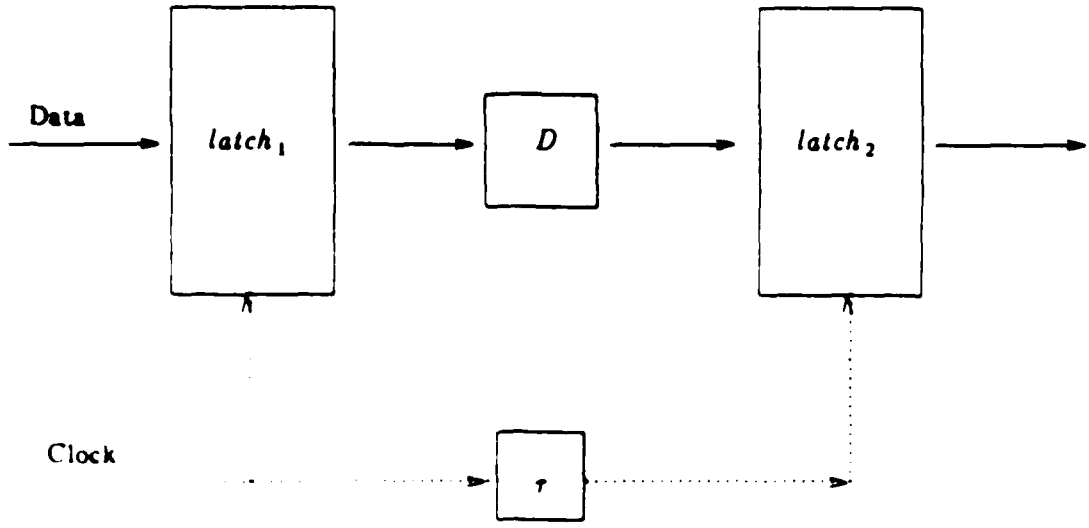


Figure 5 1 Sequential circuit clock timing relation.

the latch using the following algorithm.

```
FOR each latch DO BEGIN
    WHILE T1 < TMAX DO BEGIN
        TX := T1+TPD1;
        WHILE T2 < TX DO T2 := T2+Tperiod2;
        (* perform paths check *)
        IF (T1 + Cri_Path_Delay + Set_Up_Time) > T2 THEN Set_Up_Time_Error;
        IF (T1 + Short_Path_Delay + Tperiod1) < (T2 + Hold_Time) THEN
        Hold_Time_Error;
        (* when signal stream through latch2 in mid-phase (used as Flip-Flop) *)
        IF (T1+TPD1 + Short_Path_Delay) < (T2 + Hold_Time) THEN
        Hold_Time_Error;
        T1:=T1+Tperiod1;
    END;
END;
```

Here T_1 and T_2 are the triggered times of $latch_1$ and $latch_2$, respectively. TMAX is a user-defined maximum time boundary. TPD_1 is the propagation delay of $latch_1$. Cri_Path_Delay is the critical path delay from $latch_1$ to $latch_2$. Short_Path_Delay is the shortest path delay from $latch_1$ to $latch_2$. When the signals stream through the latches in mid-phase, the latches are acting as flip-flops. In the case of flip-flops, we only have to check the hold time condition for the shortest paths. The criterion is that the hold time plus the clock skew must be less than the sum of TPD_1 and Short_Path_Delay. A 10% error margin from timing analysis has been included to detect a possible hidden timing error coming from the deviation of the delay estimation. The clock and data timing relation is depicted in Figure 5.2.

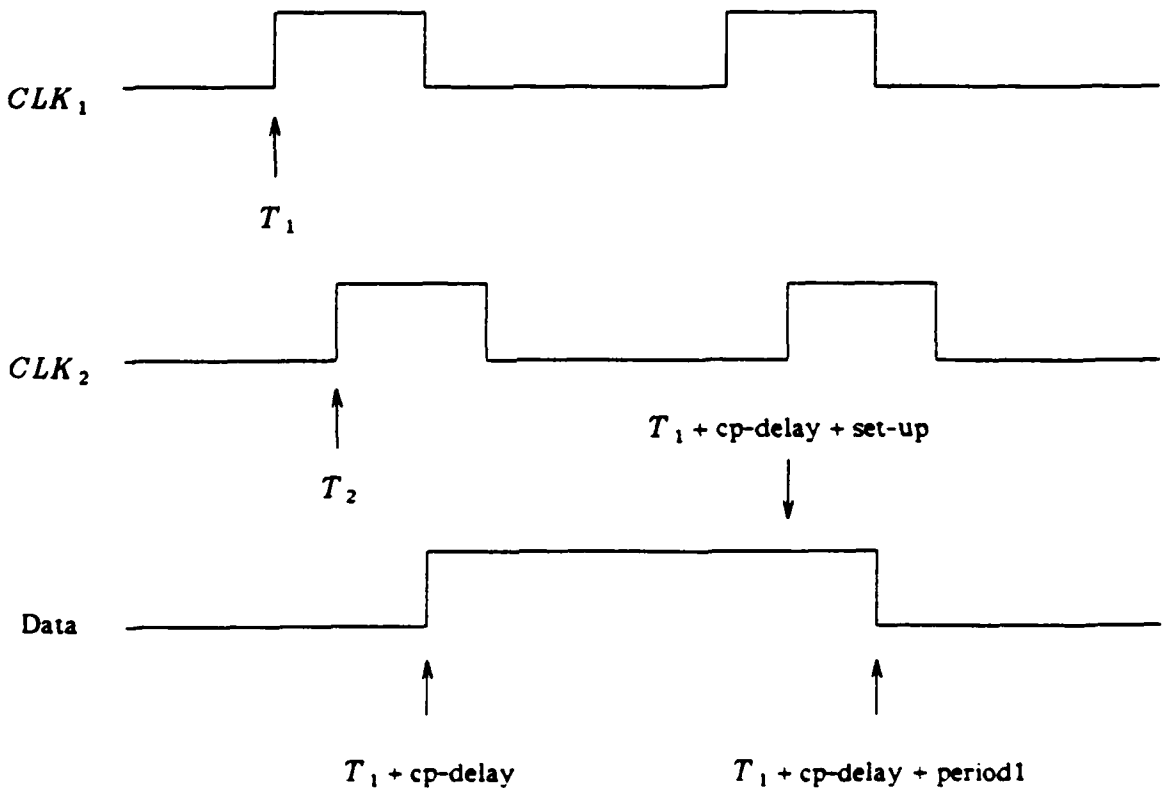


Figure 5.2. Clock and data timing relation.

5.4. Timing Error Correction

As mentioned earlier, it is important to minimize the clock skew [Bak 86]. Also, all critical paths should be optimized to solve timing problems. Here, we derive the τ boundary range from the timing error checking algorithm.

Case 1: $\tau < TPD_1$

We want $T_2 = T_1 + Tperiod_1 + \tau$, such that, the following two inequalities must be satisfied:

$$Cri\_Path\_Delay + Set\_Up\_Time < Tperiod_1 + \tau \quad (5.1)$$

$$\tau < Short\_Path\_Delay - Hold\_Time. \quad (5.2)$$

From Equations 5.1 and 5.2 we get

$$Cri\_Path\_Delay + Set\_Up - Tperiod_1 < \tau < Short\_Path\_Delay - Hold\_Time. \quad (5.3)$$

Case 2: $\tau > TPD_1$

We want $T_2 = T_1 + \tau$, such that, the following two inequalities must be satisfied:

$$Cri\_Path\_Delay + Set\_Up\_Time < \tau \tag{5.4}$$

$$\tau < Short\_Path\_Delay - Hold\_Time + Tperiod_1. \tag{5.5}$$

From Equations 5.4 and 5.5 we get

$$Cri\_Path\_Delay + Set\_Up < \tau < Short\_Path\_Delay - Hold\_Time + Tperiod_1. \tag{5.6}$$

The τ value must be within a boundary to satisfy the timing requirement. Let a represent the expression $Cri\_Path\_Delay + Set\_Up$ and b represent the expression $Short\_Path\_Delay - Hold\_Time + Tperiod_1$. We construct a table to describe the 6 possibilities of the permutations and appropriate adjusting sequences to get the relationship in Table 5.1.

inequality	first action	goal	second action	goal
$a < \tau < b$	null		null	
$a < b < \tau$	$b\uparrow$	$2\tau - a$	$\tau\downarrow$	$\frac{a+b}{2}$
$\tau < a < b$	$a\downarrow$	$2\tau - b$	$\tau\uparrow$	$\frac{a+b}{2}$
$\tau < b < a$	$a\downarrow$	$2\tau - a$	$b\uparrow$	$2a - \tau$
$b < a < \tau$	$b\uparrow$	$2\tau - a$	$a\downarrow$	$2b - \tau$
$b < \tau < a$	$a\downarrow$	$2b - \tau$	$b\uparrow$	$2a - \tau$

Table 5.1. Clock and path delay relationship.

5.4.1. Clock Path Speedup

The most effective method for correcting timing errors is to speed up the clock path while equalizing the loads of the various clock drivers, such that, the clock skew is reduced. The clock paths are tuned backwards starting with the sizing of the last-stage transistors. The changes in capacitance loading after resizing are propagated to the driving blocks for further tuning [Lai 87b]. Only the first clock of the latch pair is tuned each time, so that the latter stage tuning will not affect the previous results, see Figure 5.3, or it would have the same amount of influence on the clock delays but the clock skew kept the same.

5.4.2. Critical Paths and Shortest Paths

When the clock paths have been sped up, but still fail to remove the timing error, the propagation delay times of critical paths and shortest paths are then adjusted according to the type of the errors. In this section, we consider two frequently occurring problems having to do with paths between vertices. In what follows, let G be a directed graph. The graph, G^*, which has the same vertex set as G, but has an edge from v to w if and only if there is a path from v to w in G, is called the *transitive closure* of G [Aho 74]. A problem closely related to finding the transitive closure of a graph is the *shortest path problem*. Associated with each edge e of G is a nonnegative delay $d(e)$. The *path delay* is defined to be the sum of the delays of the

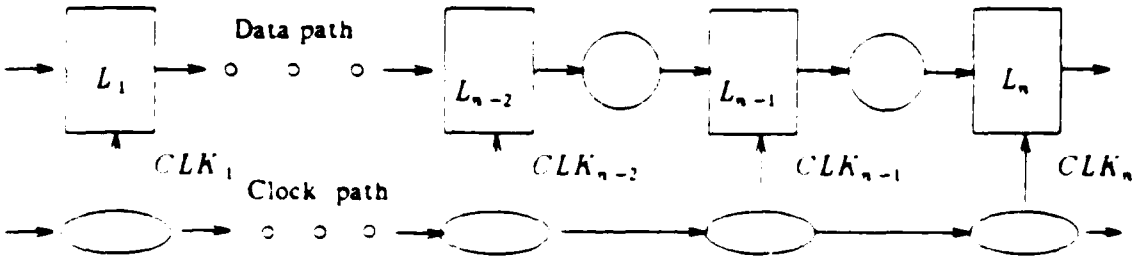


Figure 5.3 Clock tuning strategy

edges in the path. The shortest path problem is to find for each ordered pair of vertices (v , w), the lowest delay of any path from v to w. Consider a directed graph $G=(V, E)$, where $V=(v_1, v_2, \ldots, v_n)$. Let $d(v_i, v_j)$ be the delay of edge (v_i, v_j), if one exists, and ∞ otherwise. Compute D_{ij}^k for all $1 \leq i \leq n$, $1 \leq j \leq n$, and $0 \leq k \leq n$. The intention is that C_{ij}^k should be the shortest path from v_i to v_j which does not pass through any vertex higher than v_k. The algorithm is as follows:

```
begin
    for 1≤i, j ≤n do C_ij^0 ← d(v_i, v_j);
    for k ←1 until n do
        for 1≤i, j ≤n do
            C_ij^k ← MIN(C_ij^(k-1), C_ik^(k-1) + C_kj^(k-1));
end.
```

The sizes of the transistors in the critical paths are flagged when there is a Setup_Timing_Error. The shortest path is considered instead for Hold_Timing_Error. Reducing the propagation delay between latches can reduce the clock period and hence increase the circuit speed performance. All the blocks that belong to critical paths, but not to shortest paths, are tuned for speedup. The sizes of transistors are adjusted according to their loading capacitance. The side effect of this backward tuning can reduce the variation in the gate propagation delay for the combinational circuit between latches which decreases the probability of *critical races*. In the first step, tuning may increase the propagation delay of the shortest path. If the Hold_Timing_Error still exists, then the driven block of the shortest path is flagged to increase the delay times of the shortest path.

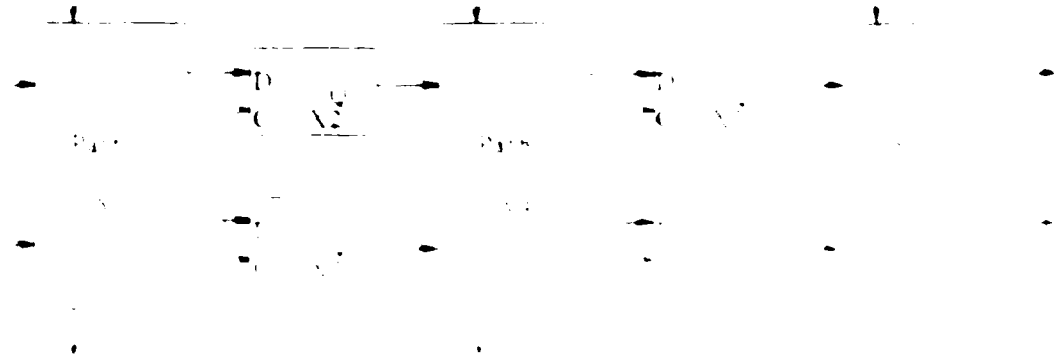
5.4.3. Clock Pulse Width

The clock period must be long enough to permit the effect of any input change to propagate to, and stabilize at, the inputs of the latch before the occurrence of the clock pulse. The maximum value of the propagation delay time (often equal to twice the typical value) must be used for each gate in the logic chain when calculating the minimum clock period. A last resort to overcome the timing problem is to increase the clock pulse width and the clock period. The program prints out the minimum clock period required if it can not successfully tune the circuit under the current clock timing condition.

5.5. Experimental Results and Summary

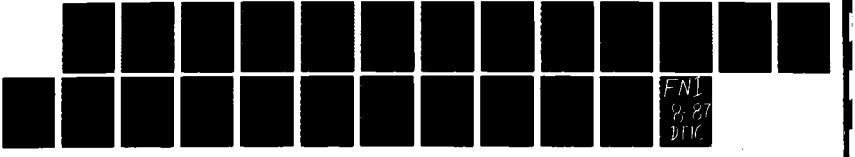
A CMOS data path circuit with 160 transistors, shown in Figure 5.4, was tuned by the program.

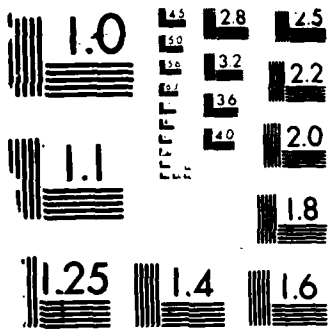
The original worst-case delay time between input and output was 13.8 ns, and the total



AD-A182 869 RULE-BASED CIRCUIT OPTIMIZATION FOR CMOS VLSI(U) 2/2
ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB F LAI
JUL 87 UILU-ENG-87-2244 N00014-84-C-0149

UNCLASSIFIED F/G 9/1 NL





MICROCOPY RESOLUTION TEST CHART

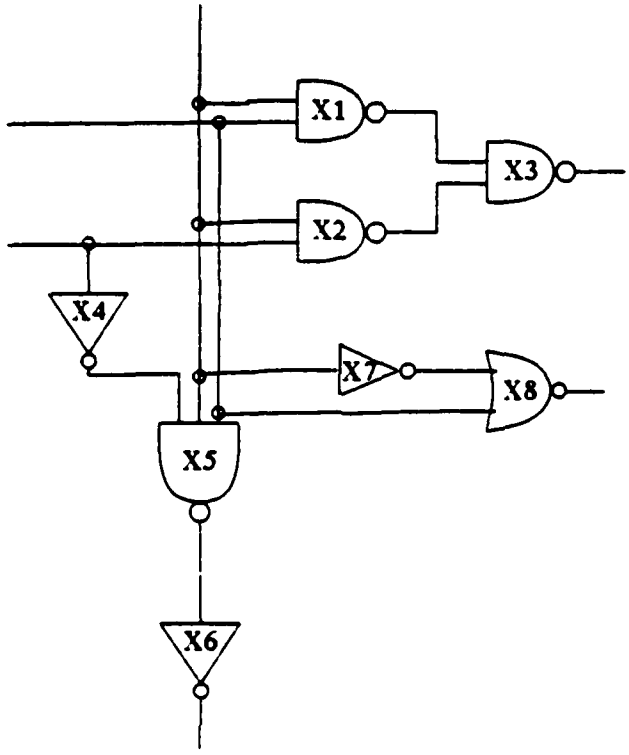


Figure 5.4(b). Path circuit.

transistor width was 640 μm. The program took 7 minutes to reduce the worst-case delay time to 8.5 ns, with a new total transistor width of 845 μm. The maximum propagation delay time between latches was reduced from 5.1 ns to 3.8 ns. The program is truly autonomous and does not require any human intervention. In comparison with a manual approach, the design turn around time can be shortened by a few orders of magnitude.

In this chapter we have presented a timing error correction method through transistor resizing techniques. By minimizing the clock skew, the propagation delay time of a path decreases, permitting a higher frequency of operation and improving the data throughput of the overall system. Critical paths and shortest paths between each latch pair are tuned to meet the set-up time and the hold time requirements.

CHAPTER 6.

CONCLUSIONS AND FUTURE WORK

A design aid system, iJADE, has been developed for high performance CMOS VLSI circuit generation. It has four novel features. First, it combines the analytic tools with a rule-based expert system to take advantage of timely on-line information to administer the rules and to verify the actions. iJADE is a closed loop performance evaluation and design parameter refinement system. The second novelty is that iJADE can detect the latches, trace the clock paths, and accurately simulate the voltage waveforms in the circuit and tune synchronous circuits to satisfy the clock timing constraints. The third novelty is that iJADE optimizes the circuit performance by tuning the critical paths found by the worst-case timing analyzer. The fourth novelty is that the hierarchical structure for both circuit and programming saves a lot of memory, speeds up calculations, and can define the potential problem subcircuit clearly. Experimental results on a 4-bit ALU and register paths show that iJADE-optimized CMOS circuits perform competitively, or even better, in terms of glitch avoidance, propagation delay time, and chip area.

Power dissipation is another important concern with the performance of very large-scale circuits. It becomes increasingly more important to reduce the power dissipation as the number of devices in VLSI circuits increases [Kan 86]. Beginning early with the chip floor plan, it would be helpful to consider laying the power and ground wires on a VLSI chip. This will provide the designers the chance to estimate the power distribution at the pre-layout stage. Observation of the hot spots on the whole chip will be helpful for the designers to achieve better thermal immunity for the circuits. There are two components that establish the amount of power dissipated in a CMOS circuit. They are:

1 Static dissipation - due to leakage current.

2 Dynamic dissipation - due to:

a. switching transient current

b. charging and discharging of load capacitances.

There is some small static dissipation due to reverse bias leakage current between diffusion regions and the substrate. The static power dissipation is the product of the device leakage current and the supply voltage. The dynamic dissipation can be modeled by assuming the rise and the fall time of the step input is much less than the repetition period.

We are also studying methods of applying the optimization data to other aspects of circuit design. For example, designers frequently face situations where a speed specification on a circuit path cannot be met, no matter how the circuit's transistors are sized. In such cases, designers often try to rearrange interconnect wires and module placements to reduce the capacitive loading on critical nodes, hoping to thereby meet the speed requirement. The design optimization techniques can be extended to the logical-level. In the logical-level, the designer has to make intelligent judgements for each logical path about the proper logic pattern and the proper number of stages.

APPENDIX A.

iJADE VERSION 1A USER'S GUIDE

iJADE is a multiple functional program that optimizes VLSI CMOS digital circuits. Circuits may contain resistors, capacitors, independent voltage sources, and MOSFET's. iJADE has built-in models for the device capacitance. iJADE can simulate or analyze the circuit at the switch-level to identify the critical paths. For optimization iJADE uses both algorithms and a rule-based expert system. The primitive cells delay data are stored in the file *data.l*. The technology dependent data are stored in the file *teck.l*.

To run the program, type

iJADE input-file output-file {option}

where input-file is the input file name, and output-file is the output file name.

There are several options available:

s: simulation

a: analysis

m: combinational

c: sequential clocked

o: optimization

p: critical path generation.

"s" and "a" are mutually exclusive and so are "m" and "c".

"o" also generates the list of critical paths.

A.1. Frame Data-base

Schematic frames are those frames which hold the information about the circuit elements as they are initially entered and further classified by the program. Just below the root "whole" there are eleven frames: transistor, capacitor, resistor, input, model, node, subcircuit, subcircuit-call, block, latch, and strongly connected component(SCC).

Transistor frame stores the structure and property of a transistor. It includes the following slots: ako (a-kind-of), drain, gate, source, substrate, model-card-name, length, width, drain-area, source-area, drain-perimeter, source-perimeter, and beta (W/L).

Node frame includes the following slots: ako, status, neighbor, gate-of, lump-capacitance, external-capacitance, load-capacitance, capacitor, resistor, fanin-block, fanout-block, initial-condition, block-p, and block-n. Where, *status* represents the strength of the node such as *input*, *pull-up*, *pre-charge*, *normal*. *Neighbor* is a DC connected relation. Its data structure is as follows: (connected-through-device, neighboring-node). The connected-through-device can be a resistor or a transistor. *Gate-of* is the transistor list with the node as its gate. *Lump-capacitance* is the total lumped capacitance between the node and ground. *External-capacitance* is the total lumped capacitance between the node and ground excluding the parasitic capacitance contributed from the transistor. *Load-capacitance* is the loading capacitance from upper level. *Block-p* is the PMOS driving block and *block-n* is the NMOS driving block.

Circuit frame includes the following slots: ako, model-card, delay-specification, area-specification, maximum-area, mos, nodes, gate-set, capacitor, resistor, port, input-node, input-name, output-node, pull-up, pass-transistor, subcircuit-call, called-by, block, scc, latch, and ordering.

Capacitor frame includes the following slots: ako, c+, c-, and farad.

Resistor frame includes the following slots: ako, r+, r-, and ohm.

Subcircuit-call frame includes the following slots: ako, port, call, edge-to, edge-from, number, lowlink, and label. Where, *number* and *lowlink* are used in the SCC detecting algorithm. *Label* value represents the topological ordering of the device.

Block frame includes the following slots: ako, node-set, input-node, output-node, mos, resistor, capacitor, block-type, edge-to, edge-from, number, lowlink, visited, and label.

Strongly connected component frame includes the following slots: ako, component, input-node, output-node, edge-to, edge-from, visited, label, and status.

Latch frame includes the following slots: ako, clock-node, critical-f, critical-r, shortest-r, shortest-f, internal-delay.

Input frame includes the following slots: ako, v+, v-, source-type, waveform, sequence, and interval.

Model frame includes the following slots: ako, type, ld, vto, xj, tox, nsub, and mu.

A.2. Input Format

The circuit format is generally the same as in SPICE. But the following information must be provided as well. iJADE accepts most of the SPICE format and ignores the others that it can not recognize. Design specification token always starts with a "*" such that it will be processed by iJADE but ignored by SPICE. Subcircuit must be previously defined before it can be called. Several tokens starting with "*" have special meanings in iJADE so that the same file can be run in both SPICE and iJADE.

1. Input nodes must be specified.

   *input i_1 i_2, ..., i_n

2. Output nodes must be specified.

*output o_1, o_2, ... o_n

3. A subcircuit must be defined before it can be called.

4. Subcircuit's port function must be specified: i (input) / o (output).

.subckt inverter 1 2 99

*i/o i o i

5. Current sources can not be handled at present. The current card is ignored in iJADE.

6. All the clock nodes must be defined.

*clock 99

7. The setup time, T_{su}, is the time interval allowed for the inputs to stablize before they are clocked into the flip-flop.

*set-up 3ns

8. The hold time, T_h, is the time interval after the clock transition has occurred during which the inputs must be held stable.

*hold 3ns

9. For the worst-case timing analysis, the input slew rate may be specified by the user. The default value is 1 ns.

*input-slew 1.5ns

10. The characteristic impedance of the transmission line can be specified so that iJADE can determine the transistor size of the output driver.

*z 100

11. No floating capacitors are allowed.

12. The maximum transistor can be specified.

*max-width 200u

13. The upper bound on total power consumption is set in terms of the maximum total channel area allowed.

*total-width 3000u

14. TTL level signal input nodes should be specified to set the width ratio between P and N MOSFET.

*TTL-input 3 4

15. The maximum delay time can be specified.

*delay 10ns

16. The character ":" is reserved for iJADE to build names for the flattened circuit from the hierarchical one. Please avoid using the character in SPICE input file.

17. The OPTIONS card can be used to set the default width and length of the transistors. but it must placed before any transistor definition.

.OPTIONS defw=4u defl=2u

A.2.1. Input Example

```
cmos register
 .temp 95
 .subckt not 2 3 1
*i/o i o i
mp1 3 2 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn1 3 2 0 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
 .ends not

 .subckt nor 2 3 4 1
*i/o i i o i
mp1 5 2 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mp2 4 3 5 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
```

```
mn1 4 2 0 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn2 4 3 0 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
.ends nor

.subckt nand 2 3 4 1
*i/o i i i o i
mp1 4 2 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mp2 4 3 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn1 4 2 5 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn2 5 3 0 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
.ends nand

.subckt nand3 2 3 4 5 1
*i/o i i i o i
mp1 5 2 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mp2 5 3 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mp3 5 4 1 1 modp l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn1 5 2 6 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn2 6 3 7 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
mn3 7 4 0 0 modn l=2u w=4u ad=16p as=16p ps=16u pd=16u
.ends nand3

.subckt latch 8 10 6 4 1
*i/o i i o o i
x1 8 5 1 not
x2 5 10 7 1 nand
x3 8 10 2 1 nand
x4 2 6 4 1 nand
x6 7 4 6 1 nand
.ends latch

.subckt between 11 2 3 4 5 6 1
*i/o i i i o o o i
x1 11 2 7 1 nand
x2 11 3 8 1 nand
x3 7 8 4 1 nand
x4 3 10 1 not
x5 10 11 2 12 1 nand3
x6 12 6 1 not
x7 11 9 1 not
x8 9 2 5 1 nor
.ends between

*input 11 22 33 1 2 3 99
*output 6 12 17 15 16 90 91 92 93
c6 6 0 0.1pf
c12 12 0 0.1pf
c17 17 0 0.1pf
c15 15 0 0.1pf
c16 16 0 0.1pf

x1 11 2 3 4 5 6 1 between
```

```
x2 4 99 90 7 1 latch
x3 5 99 91 8 1 latch

x4 22 7 8 9 10 12 1 between
x5 9 199 92 13 1 latch
x6 10 199 93 14 1 latch

x7 33 13 14 15 16 17 1 between
x8 99 199 1 not
x9 199 299 1 not

*clock 99
 .print tran v(15) v(16) v(17)
vclock 99 0 pulse(0 5 1ns 1ns 1ns 14ns 30ns)
v2 2 0 pulse(0 5 1ns 1ns 1ns 20ns 50ns)
v3 3 0 pulse(0 5 11ns 1ns 1ns 20ns 50ns)
v11 11 0 dc 5
v22 22 0 dc 5
v33 33 0 dc 5
vdc 1 0 dc 5
 .tran 0.2ns 60ns

 .model modn nmos vto=1.25 uo=525 kappa=2.0e-2 nsub=1.77e+17 theta=0.073
+ vmax=1.74e+5 eta=0.96 tox=3.15e-8 xj=3.0e-7 tpg=1.0 js=1.0e-7 cj=1.23e-3
+ pb=9.71062e-1 rsh=17

 .model modp pmos vto=-0.95 uo=298 kappa=2.0e-2 nsub=7.66e+15 theta=0.138
+ vmax=2.92e-5 eta=1.4 tox=3.15e-8 xj=3.0e-7 tpg=-1.0 js=1.0e-7 cj=2.66e-4
+ pb=8.99323e-1 rsh=56
 .end
```

A.3. Output Format

The switch-level timing simulator and analyzer generate discrete waveform sequences for the output nodes and critical paths described by blocks. The PRINT card is used as in SPICE, but PRTYPE is ignored.

.PRINT PRTYPE V(1)

The general discrete voltage waveform is as follows:

$$([0/1], u, t_1), ([1\ 0], u, t_2)$$

In the output the default capacitance unit is femto farad, and the default time unit is pico second.

A.3.1. Output Example

```
whole15        : node name
(u 1 -1)       : signal waveform

whole16
(0 u 1.1185e-08)
(u 1 1.4842e-08)
(1 u 3.7617e-08)
(u 0 3.8887e-08)

whole17
(0 u 8.512e-09)
(u 0 9.347e-09)

delay-r : 9950
delay-f : 9522

critical-path-f
wholex7:betweenx6:blocknot3.not0
wholex7:betweenx5:blocknand31.nand35
wholex7:betweenx4:blocknot3.not0
wholex6:latchx4:blocknand1.nand4
wholex6:latchx3:blocknand5
wholex4:betweenx8:blocknor5
wholex2:latchx4:blocknand5
wholex2:latchx3:blocknand1.nand4
wholex1:betweenx3:blocknand5
wholex1:betweenx2:blocknand1.nand4

critical-path-r
wholex7:betweenx8:blocknor5
wholex5:latchx4:blocknand5
wholex5:latchx3:blocknand1.nand4
wholex4:betweenx3:blocknand5
wholex4:betweenx1:blocknand1.nand4
wholex2:latchx4:blocknand5
wholex2:latchx3:blocknand1.nand4
wholex1:betweenx3:blocknand5
wholex1:betweenx2:blocknand1.nand4


wholex7:betweenx6:blocknot3.not0
scale :(2.0)
wholex7:betweenx5.blocknand31.nand35
scale :(1.0)
wholex7:betweenx4:blocknot3.not0
scale :(1.0)
wholex7:betweenx8:blocknor5
scale :(5.309229648742579)
wholex5:latchx4:blocknand5
scale :(2.56)
```

wholex5:latchx3:blocknand1.nand4
scale :(1.0)
wholex6:latchx4:blocknand1.nand4
scale :(1.274215115698219)
wholex6:latchx3:blocknand5
scale :(1.0)
wholex4:betweenx3:blocknand5
scale :(1.28)
wholex4:betweenx1:blocknand1.nand4
scale :(1.0)
wholex4:betweenx8:blocknor5
scale :(1.698953487597625)
wholex2:latchx4:blocknand5
scale :(2.56)
wholex2:latchx3:blocknand1.nand4
scale :(1.0)
wholex1:betweenx3:blocknand5
scale :(1.28)
wholex1:betweenx2:blocknand1.nand4
scale :(1.0)

"hold time not long enough"

"set-up time not long enough"

"clock pulse width not long enough"

list: (75413 1470 64)
symbol: (3098 126 25)
string: (61 62 1)
fixnum: (621 11 128)
flonum: (451 21 64)
total width= 0.072
total time= 72 seconds

A.4. Delay File

*table-r : (slew-rate t1 t2) of RC circuit (unit: pico seconds)

R: 10k, C: 10ff

((6 28 138) (30 38 140) (60 51 144) (120 65 175) (180 74 214) (300 83 316)

(600 95 605) (1200 100 1200) (1800 100 1800) (2400 100 2400) (3000 100 3000)

(3600 100 3600) (4200 100 4200) (4800 100 4800) (5400 100 5400) (6000 100 6000)

(7200 100 7200) (9000 100 9000) (10800 100 10800) (12000 100 12000)

(15000 100 15000))

```
*table1.1 : type 1 primitive N-drive
((6 130 256) (30 152 267) (60 166 269) (120 215 272) (180 259 274) (300 330 282)
 (600 529 345) (1200 843 529) (1800 1131 679) (2400 1357 926) (3000 1636 1012)
 (3600 1906 1093) (4200 2189 1206) (4800 2393 1471) (5400 2636 1554)
 (6000 2947 1596) (7200 3466 1844) (9000 4109 2512) (10800 4936 2605)
 (12000 5404 2844) (15000 6781 3170))
*table1.2 : type 1 primitive P-drive
((6 178 375) (30 205 397) (60 220 400) (120 255 400) (180 296 400) (300 382 409)
 (600 532 487) (1200 866 636) (1800 1172 744) (2400 1469 837) (3000 1738 968)
 (3600 2021 1068) (4200 2324 1185) (4800 2601 1328) (5400 2831 1449)
 (6000 3111 1500) (7200 3691 1679) (9000 4304 2265) (10800 5275 2328)
 (12000 5817 2533) (15000 7147 2829))
*table2.1 : type 2 primitive pass-1
((6 121 728) (30 143 738) (60 155 738) (120 189 740) (180 211 742) (300 249 773)
 (600 301 944) (1200 338 1428) (1800 353 1990) (2400 364 2575) (3000 368 3164)
 (3600 369 3752) (4200 369 4342) (4800 372 4932) (5400 372 5528) (6000 372 6122)
 (7200 372 7316) (9000 374 9108) (10800 374 10900) (12000 374 12103)
 (15000 374 15100))
*table2.2 : type 2 primitive pass-0
((6 87 498) (30 113 498) (60 133 498) (120 173 498) (180 210 508) (300 249 576)
 (600 346 720) (1200 405 1233) (1800 431 1822) (2400 446 2400) (3000 454 2996)
 (3600 460 3583) (4200 463 4178) (4800 466 4772) (5400 468 5366) (6000 468 5958)
 (7200 470 7158) (9000 470 8958) (10800 471 10734) (12000 471 11932)
 (15000 472 14928))
*table3.1 : type 3 primitive pass-1
((6 14 555) (30 45 555) (60 71 559) (120 120 564) (180 164 566) (300 226 589)
```

(600 370 691) (1200 563 963) (1800 700 1138) (2400 732 1418) (3000 837 1577)
(3600 863 1763) (4200 937 1854) (4800 947 2129) (5400 959 2263) (6000 969 2433)
(7200 902 2793) (9000 878 3396) (10800 839 3616) (12000 785 4016)
(15000 650 4424))

*table3.2 : type 3 primitive pass-0

((6 25 370) (30 82 370) (60 100 371) (120 149 380) (180 190 383) (300 245 407)
(600 380 527) (1200 580 789) (1800 735 997) (2400 830 1288) (3000 955 1439)
(3600 1051 1587) (4200 1151 1714) (4800 1243 1939) (5400 1291 2080)
(6000 1376 2231) (7200 1508 2480) (9000 1635 3016) (10800 1709 3382)
(12000 1802 3555) (15000 2004 4049))

*slew-rate (unit: pico seconds)

(6 30 60 120 180 300 600 1200 1800 2400 3000 3600 4200 4800 5400 6000 7200 9000
10800 12000 15000)

(slew-rate, equivalent-resistance) (unit: pico second, kilo ohm)

*r-eq1.1 : type 1 pass transistor equivalent R when conducting 1

((6 15) (30 18) (60 19) (120 20) (180 22) (300 25) (600 32) (1200 49) (1800 64)
(2400 80) (3000 93) (3600 106) (4200 119) (4800 134) (5400 147) (6000 159)
(7200 183) (9000 225) (10800 259) (12000 281) (15000 339))

*r-eq1.2 : type 1 pass transistor equivalent R when conducting 0

((6 23) (30 25) (60 26) (120 27) (180 28) (300 32) (600 39) (1200 54) (1800 69)
(2400 83) (300 98) (3600 110) (4200 123) (4800 139) (5400 152) (6000 162)
(7200 187) (9000 258) (10800 263) (12000 286) (15000 342))

*r-eq2.1 : type 2 pass transistor equivalent R when conducting 1

16

*r-eq2.2 : type 2 pass transistor equivalent R when conducting 0

16

*r-eq3.1 : type 3 pass transistor equivalent R when conducting 1

((6 17) (30 19) (60 20) (120 22) (180 23) (300 26) (600 34) (1200 49) (1800 59)
(2400 70) (3000 78) (3600 85) (4200 91) (4800 100) (5400 105) (6000 111)
(7200 121) (9000 140) (10800 146) (12000 157) (15000 166))

*r-eq3.2 : type 3 pass transistor equivalent R when conducting 0

((6 12) (30 14) (60 15) (120 16) (180 18) (300 20) (600 29) (1200 44) (1800 56)
(2400 69) (3000 78) (3600 86) (4200 93) (4800 104) (5400 110) (6000 118)
(7200 130) (9000 152) (10800 166) (12000 175) (15000 198))

A.5. Technology File

ad-coef :: ad= l*w + ad-coef

as-coef :: as= l*w + as-coef

pd-coef :: pd= 2*w + pd-coef

ps-coef :: ps= 2*w + ps-coef

max-load-c :: the maximum output loading capacitance, to check if buffer needed

V_{th} :: upper threshold voltage (= 4 volt)

V_{tl} :: lower threshold voltage (= 1 volt)

V_{TTL} :: TTL threshold voltage

C_{cell-1} :: Primitive cell 1 equivalent loading capacitance

C_{cell-2} :: Primitive cell 2 equivalent loading capacitance

RC :: RC time constant for the RC delay time table

A.5.1. Physical, Materials, and Default Values

(1) Intrinsic carrier concentration of silicon: n_i^2=1.375e 24/cm^6

(2) Dielectric constant of vacuum: $\epsilon_0 = 8.85e-14\ farad/cm$

(3) Dielectric constant of silicon: $\epsilon_{si} = 11.7\epsilon_0$

(4) Dielectric constant of SiO_2: $\epsilon_{ox} = 3.9\epsilon_0$

(5) Gate oxide capacitance per unit area: $C_{ox} = \frac{\epsilon_{ox}}{t_{ox}}$

(6) Body-effect coefficient: $\gamma = \frac{1}{C_{ox}}\sqrt{2q\,\epsilon_{si}\,N_A}$

(7) built-in junction potential: $\phi_p = 2V_T \ln\frac{N_D}{n_i}$ $\phi_n = 2V_T \ln\frac{N_A}{n_i}$

(8) Junction capacitance: $CJ_p = (\frac{q\,\epsilon_{si}\,N_D}{2\phi_p})^{1/2}$ $CJ_n = (\frac{q\,\epsilon_{si}\,N_A}{2\phi_n})^{1/2}$

(9) Sidewall capacitance: $CJSW_p = CJ_p\,X_j$ $CJSW_n = CJ_n\,X_j$

(10) $C_{dbp} = CJ_p\,AD_p + CJSW_p\,PD_p$ $C_{dbn} = CJ_n\,AD_n + CJSW_n\,PD_n$

(11) $K_{eqp} = \frac{\phi_p}{2.5}((1+\frac{5}{\phi_p})^{1/2}-1)$ $K_{eqn} = \frac{\phi_n}{2.5}((1+\frac{5}{\phi_n})^{1/2}-1)$

(12) $C_{jdp} = C_{ox}\,LDW_p$ $C_{jdn} = C_{ox}\,LDW_n$

(13) Lumped device capacitance: $C_{dev} = K_{eqp}\,C_{dbp} + K_{eqn}\,C_{dbn} + C_{jdp} + C_{jdn}$

(14) Gate capacitance: $C_j = C_{ox}\,LW$

REFERENCES

[Agr 82] V. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," *IEEE 19th Design Automation Conference*, pp.629-635, June 1982.

[Aho 74] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., 1974.

[Al-H 85] H. Al-Hussein, "Path-Delay Computation Algorithms for VLSI Systems," *VLSI Design*, pp.86-91, February 1985.

[Ann 86] M. Annaratone, *Digital CMOS Circuit Design*, Kluwer Academic Publishers, 1986.

[Bak 86] H. Bakoglu, J. Walker, and J. Meindl, "A Symmetric Clock-Distribution Tree and Optimized High-Speed Interconnections for Reduced clock skew in ULSI and WSI Circuits," *IEEE ICCAD'86*, pp.118-122, November 1986.

[Ben 82] L. Bening, T. Lane, and J. Smith, "Developments in Logic Network Path Delay Analysis," *IEEE 19th Design Automation Conference*, pp.605-615, June 1982.

[Bir 86] W. Birmingham, R. Joobbani, and J. Kim, "Knowledge-Based Expert Systems and Their Application," *IEEE 23rd Design Automation Conference*, pp.531-539, June 1986.

[Bry 81] R. Bryant, "MOSSIM: A Switch-Level Simulator for MOS LSI," *IEEE 18th Design Automation Conference*, pp.786-790, June 1981.

[Bra 81] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated-Circuit Design," *Proceedings of the IEEE*, Vol. 69, No. 10, pp.1334-1362, October 1981.

[Buc 84] B. Buchanan, and E. Shortliffe. *Rule-Based Expert Systems*. Addison-Wesley Publishing Co., 1984.

[Cha 85] E. Chan. "Development of a Timing Analysis Program for Multiple Clocked Network." *IEEE 22nd Design Automation Conference*, pp.816-819, June 1985.

[Coh 85] W. Cohen, K. Bartlett, and A. Geus. "Impact of Metarules in a Rule Based Expert System for Gate Level Optimization." *IEEE Proceedings of ISCAS'85*, pp.873-876, May 1985.

[Fis 85] J. Fishburn, and A. Dunlop. "TILOS: A Posynomial Programming Approach to Transistor Sizing." *IEEE ICCAD'85*, pp.326-328, November 1985.

[Gla 77] A. Glaser, and G. Subak-Sharpe. *Integrated Circuit Engineering*. Addison-Wesley Publishing Co., 1977.

[Gla 84] L. Glasser, and L. Hoyte. "Delay and Power Optimization in VLSI Circuits." *IEEE 21st Design Automation Conference*, pp.529-535, June 1984.

[Gla 85] L. Glasser, and D. Dobberpuhl. *The Design and Analysis of VLSI Circuits*. Addison-Wesley Publishing Co., 1985.

[Hit 82] R. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program." *IEEE 19th Design Automation Conference*, pp.594-604, June 1982.

[Hod 83] D. Hodges, and H. Jackson. *Analysis and Design of Digital Integrated Circuits*. McGraw-Hill Book Co., 1983.

[Hwa 86] S. Hwang, Y. Kim, and A. Newton. "An Accurate Delay Modeling Technique for Switch-Level Timing Verification." *IEEE 23rd Design Automation Conference*, pp.227-233, June 1986.

[Jou 83] N. Jouppi. "Timing Analysis for nMOS VLSI." *IEEE 20th Design Automation Conference*. pp.411-418. June 1983.

[Kan 81] S. Kang. "A Design of CMOS Polycells for LSI Circuits." "*IEEE Trans. Circuits and Systems*." Vol. CAS-28. No. 8. pp.838-843. August 1981.

[Kan 86] S. Kang. "Accurate Simulation of Power Dissipation in VLSI Circuits." *IEEE J. of Solid-State Circuits*. Vol. SC-21. No. 5. pp.889-891. October 1986.

[Kan 83] A. Kanuma. "CMOS Circuit Optimization." *Solid State Electronics*. vol. 26. No. 1. pp.47-58. January 1983.

[Kao 85] W. Kao. N. Fathi. and C. Lee. "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits." *IEEE 22nd Design Automation Conference*. pp.781-784. June 1985.

[Kow 85] T. Kowalski. and D. Thomas. "The VLSI Design Automation Assistant: What's in a Knowledge Base." *IEEE 22nd Design Automation Conference*. pp.252-258. June 1985.

[Lai 87a] F. Lai. V. Rao. and T. Trick. "JADE: A Hierarchical Switch-Level Timing Simulator." *IEEE Proceedings of ISCAS'87*. pp.592-595. May 1987.

[Lai 87b] F. Lai. S. Kang. T. Trick. and V. Rao. "iJADE: A Rule-based Hierarchical VLSI CMOS Circuit Optimizer." *IEEE Proceedings of ICCD'87*. October 1987.

[Lai 87c] F. Lai. S. Kang. and T. Trick. "A Timing Error Corrector for VLSI Synchronous Path Circuit." *IEEE Proceedings of ICCAD'87*. November 1987.

[Lee 84] C. Lee. and H. Soukup. "An Algorithm for CMOS Timing and Area Optimization." *IEEE J. of Solid-State Circuits*. Vol. SC-19. No. 5. pp.781-787. October 1984.

[Lob 84] C. Lob. "RUBBIC: A Rule-Based Expert System for VLSI Integrated Circuit Critique."

Mem. No. UCB/ERL M84/80, December 1984.

[DeM 85] H. J. De Man, I. Bolsens, E. vanden Meersch, and J. van Cleynenbreugel, "DIALOG: An Expert System for MOS VLSI Design," *IEEE Trans. Computer-Aided Design*, Vol. CAD-4, pp.303-311, July 1985.

[Mat 85] M. Matson, "Optimization of Digital MOS VLSI Circuits," *Chapel Hill Conference on Very Large Scale Integration*, pp.109-126, 1985.

[Mea 80] C. Mead, and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., 1980.

[DeM 86] G. De Micheli, "Performance-Oriented Synthesis in the Yorktown Silicon Compiler," *IEEE ICCAD'86*, pp.138-141, November 1986.

[Mur 79] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons Publishing Co., 1979.

[Mur 82] S. Muroga, *VLSI System Design*, John Wiley & Sons Publishing Co., 1982.

[Nag 75] L. Nagel, "SPICE2: A Computer Program for Simulate Semiconductor Circuits," *ERL Memo ERL-M520*, University of California, Berkeley, CA, May 1975.

[Nye 81] W. Nye, E. Polak, A. Sangiovanni-Vincentelli, and A. Tits, "DELIGHT: An Optimization-Based Computer-Aided Design System," *IEEE Proceedings of ISCAS'81*, pp.851-855, April 1981.

[Ous 85] J. Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI," *IEEE Trans. Computer_Aided Design*, Vol. CAD-4, pp.336-349, July 1985.

[Pre 77] F. Preparata, and R. Yeh, *Introduction to Discrete Structure for Computer and Engineer-*

ing. Prentice Hall. 1977.

[Rao 85] V. Rao. "Switch-Level Timing Simulation of MOS VLSI Circuits." *Report R-1032, UILU-ENG 85-2207*. University of Illinois. Urbana-Champaign. March 1985.

[Rei 77] E. Reingold. J. Nievergelt. and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice Hall. 1977.

[Rue 77] A. Ruehli. P. Wolff. Sr.. and G. Goerzel. "Analytical Power/Timing Optimization Techniques for Digital Circuits." *IEEE 14th Design Automation Conference*. pp.403-410. June 1977.

[Sho 82] M. Shoji. "Electrical Design of BELLMAC-32A Microprocessor." *IEEE ICCC'82*. pp.112-115. 1982.

[Sho 85] M. Shoji. "FET Scaling in Domino CMOS Gates." *IEEE J. of Solid-State Circuits*. Vol. SC-20. No. 5. pp.1067-1071. October 1985.

[Szy 86] T. Szymanski. "LEADOUT: A Static Timing Analyzer for MOS Circuits." *IEEE ICCAD'86*. pp.130-133. November 1986.

[Tam 83] E. Tamura. K. Ogawa. and T. Nakano. "Path Delay Analysis for Hierarchical Building Block Layout System." *IEEE 20th Design Automation Conference*. pp.403-410. June 1983.

[Ter 83] C. Terman. "RSIM - A Logic-Level Timing Simulator." *IEEE ICCD'83*. pp.437-440. November 1983.

[Wes 85] N. Weste. and K. Eshraghian. *Principles of CMOS VLSI Design a System Perspective*. Addison-Wesley Publishing Co.. 1985.

[Wit 84] J. White. and A. Sangiovanni-Vincentelli "RELAX2.1: A Waveform Relaxation Based Circuit Simulation Program." *Proceedings IEEE Custom Integrated Circuits Conference*.

pp.232-236. May 1984.

[Wil 85] A. Wilkinson. "MIND: An Inside Look at an Expert System for Electronic Diagnosis." *IEEE Design and Test*. pp.69-77. August 1985.

[Win 81] P. Winston. and B. Horn. *LISP*. Addison-Wesley Publishing Co.. 1981.

[Yan 80] P. Yang. I. Hajj. and T. Trick. "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing." *IEEE Proceedings of ICCC'80*. pp.353-355. October 1980.

[Zip 83] R. Zippel. "An Expert System for VLSI Design." *VLSI memo No. 83-134. MIT*. February 1983.

VITA

Feipei Lai was born in Changhua, Taiwan, Republic of China on May 17, 1958. He received a B.S. degree in Electrical Engineering from National Taiwan University, Taipei, R. O. C. in June 1980. He came to the University of Illinois in January 1983 and received his M.S. degree in Computer Science in May 1984. From January 1983 to May 1984 he worked as a research assistant at the Digital Computing Laboratory, Urbana. From May 1984 to May 1987 he worked as a research assistant at the Coordinate Science Laboratory and a teaching assistant with the Department of Computer Science at the University of Illinois. He has accepted a joint position as an Associate Professor in Department of Electrical Engineering and Department of Computer Science at National Taiwan University. His research interests include the area of simulation, optimization of VLSI circuits, and computer-aided design algorithms.

END

8-87

DTIC