ACCURATE TIMING VERIFICATION FOR DIGITAL VLSI DESIGNS

by

Young Hwan Kim

Memorandum No. UCB/ERL M89/2

17 January 1989

ACCURATE TIMING VERIFICATION FOR DIGITAL VLSI DESIGNS

by

Young Hwan Kim

Memorandum No. UCB/ERL M89/2

17 January 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ACCURATE TIMING VERIFICATION FOR DIGITAL VLSI DESIGNS

by

Young Hwan Kim

Memorandum No. UCB/ERL M89/2

17 January 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ACKNOWLEDGEMENTS

I thank Prof. A. Richard Newton, my research advisor for both the Master's and Ph.D. degrees, for his effective guidance and consistent encouragement during the course of this work. His door was always open to me for discussion and he was a constant source of inspiration. Had it not been for his patience and the encouragement he provided during difficult times in my life, much of this work would not have been possible.

Several Berkeley professors have graciously contributed their time on my behalf and I would like to express my gratitude. I thank Prof. Donald Pederson for consistently offering me good advice during the course of my graduate studies and gladly agreeing to serve on my qualifying examination as chairman. Thanks are due to Prof. Alberto Sangiovanni-Vincentelli and Prof. Robert Brayton for their continuing interest in my research projects. I also wish to additionally express appreciation to Prof. Arthur Bergen, Prof. Randy Katz, and Prof. Albert Pisano, who served on my qualifying examination which started at 7 o'clock in the morning, along with Prof. Donald Pederson and Prof. Richard Newton.

I thank all CAD group members at the University of California, Berkeley, for the help they provided me. Among them, I wish to express appreciation to Donald Webber for his critical reading of the manuscript of this dissertation. Thanks to Seung Hwang for the many long hours of engaging in useful discussion on verification problems. Rick Spickelmier, Peter Moore, and Tom Quarles are some of the Berkeley CAD group members who are experts in Unix[1] as well as in their research area, and were always more than happy to help me whenever I needed it. Nick Weiner, Jyuo-Min Shyu, Dr. Dave Riley, Linda Milor were

[1] Unix is a trademark of Bell Laboratories.

good enough to welcome me as a new member in their cubicle. I enjoyed discussions with Jeff Burns, Dr. Hormoz Yaghutiel, and Tammy Huang on relevant subjects, including class assignments. Karti Mayaram, Fabio Romeo, Theo Kelessoglou, Ken Kundert, Wayne Christopher, Chuck Kring, Tom Laidig, Dave Harrison, Dr. G. Casinovi, Dr. Srinivas Devadas, Dr. Hyun Shin, Dr. George Jacob, Dr. Ron Gyurcsik, Dr. Carl Sechen, Dr. Jacob White, Dr. Rwei-sing Wei, and Dr. Mark Hoffman are some of brightest people I've been lucky enough to meet over years after joining the Berkeley CAD group. I also want to offer my gratitude to Dr. Resve Saleh for things too numerous to enumerate. He has been, and will always be, a good friend.

I thank all former and current Korean EECS graduate students who I have been acquainted with for those wonderful time I had. I especially thank Dr. Bong-Koo Kang for his moral support during my early years at Berkeley.

Finally, I wish to thank my wife, Young, my children, Kyo-Jin and Sonia, and my parents, Eui B. and Choon O. Kim and Kyoo R. and Ok R. Kim, for their moral support during my graduate studies.

I feel that it is extremely important for me to offer my thankfulness for the financial support that made this research possible. Funding for this project was provided by the Semiconductor Research Corporation and computer resources were provided by Digital Equipment Corporation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. ANALYSIS OF VERY-LARGE-SCALE-INTEGRATED (VLSI) CIRCUITS

A circuit is a complex maze of signal paths that combine to effect an overall function. In general, there are two analysis methods to ensure that the circuit design is correct: *dynamic analysis* and *static analysis*, as illustrated in Figure 1.1. Dynamic analysis is the study of a circuit's behavior as a function of time. One might, for example, be interested in the voltages at a set of the output nodes of a circuit some time t after certain voltage sources are applied to

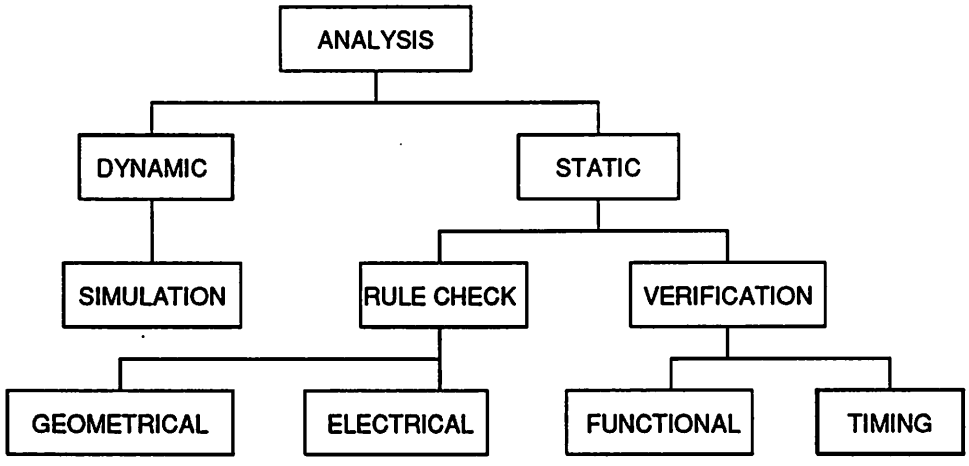


Figure 1.1 Analysis Methods of Circuit Designs

the input nodes. However, in general, it is not possible to obtain the analytical expressions that represent the behavior of a circuit. Thus, in dynamic analysis, a circuit's response to a particular set of inputs called a test vector is computed. This process is referred to as *simulation*. While computer simulations are also performed at process [1] and device levels [2,3,4], they are performed mostly for the development of fabrication process, devices, or device models. The most accurate simulator that is used by the circuit designers is probably a circuit simulator such as SPICE [5] or ASTAP [6], which provides a fine-grain detailed analog waveforms at particular nodes in the design. Unfortunately, circuit simulation is a computationally expensive process and is, therefore, impractical for circuits consisting of more than a few thousand transistors. There are two approaches for simulating VLSI designs of a few hundred thousand transistors. The first approach is to break the design into a number of smaller pieces and to simulate each piece at the circuit level. The second is to use higher (coarser) level simulators such as switch-level simulators [7, 8], logic (or gate) simulators [9, 10], and functional/behavioral simulators [11, 12, 13]. Switch-level/logic simulators replace the analog waveforms of circuit simulators with logic levels 0, 1, and X (undefined). Some simulators are augmented by a larger set of discrete levels in order to obtain more detailed information. Switch-level simulators model MOS transistors as switches and logic simulators model circuit elements at the gate level rather than the transistor level. Functional/behavioral simulators allow the use of data and control representations.

The other type of analysis method, namely static analysis, examines a circuit that is independent of input data. There are two kinds of static analysis. The first one is based on *rules checking* which examines whether a circuit obeys *geometric* or *electrical design rules*. Geometrical design rule checkers [14, 15, 16] ensure a margin of safety among wires and contacts of the layout for process errors so that the circuit will be manufactured correctly. Electr-

ical design rule checkers [17] detect the violations of electrical rules such as incorrect transistor ratios, short circuits, isolated parts of a circuit, and bad design practice. The second static analysis method is verification. There are two types of verification: *functional* and *timing*. A functional verifier [18, 19, 20] derives a circuit behavior by combining the behavior of each of the primitive components. Then, it compares the derived circuit behavior with designer-specified behavior. On the other hand, timing verification [21, 22, 23, 24] is used for the management of signal timing in a digital system. For example, in a synchronous system, if a delay through a piece of combinational logic circuit is either too long (outside the required clock range) or too short (which could cause a race condition), the system may not function correctly. This is referred to as a *timing error*. Timing verifiers detect the possibility of such timing errors. In addition, the maximum operation speed of a digital system is determined by the slowest of its all possible signal paths, called the *critical path*. Timing verifiers are also used to determine the critical paths to optimize the performance of the system.

1.2. SIMULATION VERSUS TIMING VERIFICATION

Although not impossible, it is computationally impractical to use detailed simulation for the detection of critical paths or timing errors in a VLSI design, since the number of necessary simulations would be exponential in the number of circuit nodes. In order to detect critical paths or timing errors, it is necessary to locate the longest path first. Note that simulation requires an input vector in order to determine a system response. Simulation is not guaranteed to locate the correct longest delay of a system unless the system is simulated with all possible input vectors. This is because a particular input vector is necessary to compute a pathological path delay, hence exhaustive simulation is inevitable. Additionally, even after finding the input vector causing the longest delay, it may not be easy to locate the

corresponding path, because there may be millions of signal paths between input and output nodes. On the other hand, as will be described in Chapter 3, timing verifiers carry out only one analysis (or a few analyses) using fast delay approximation techniques and report much of the information necessary to improve or correct the design. Therefore, timing verification is preferred for the management of signal timing in a VLSI system due to its computational efficiency and test completeness.

1.3. ORGANIZATION OF THE THESIS

In this dissertation, a complete and consistent study of timing verification techniques is presented. Then, new approaches for the accurate timing verification of VLSI MOS designs are described.

The main issue of Chapter 2 is the timing constraints that n-phase clocking synchronous systems must satisfy in order to function correctly. First, two types of clocked storage elements (edge-triggered and transparent) that are used for the global synchronization of system signals are described. For an effective description of timing constraints, the terms and notation that are not standardized in the literature are defined. Then, the timing constraints that IC designers have used for the design of synchronous systems including those with precharged busses and dynamic logic gates are described. Even though some of these timing constraints can be found in the literature, they are usually presented only for single-phase or two-phase clocking synchronous system and with only one of two types of clocked storage elements. In addition, this chapter presents the algorithms that examine the satisfaction of timing constraints of the combinational logic circuit between clocked storage elements.

In Chapter 3, timing verification techniques are introduced. First, to illustrate the difference between simulation and timing verification, value-dependent and value-

independent signal propagations are compared. Two path analysis methods, *path enumeration* and *critical-path analysis*, are presented. Issues associated with path delays are described, including delay computations and representations. Since a critical-path analyzer can be implemented based on modified depth-first search, level-based search, or topological-order based search, the complexity and loop-handling of different traversal approaches for the critical-path analysis method are compared. Finally, problems specific to switch-level critical-path analyzers are described.

The focus of Chapter 4 is the development of an accurate delay modeling technique, referred to as *Electrical-Logic (ELogic)* modeling technique. The delay models of existing switch-level timing verifiers are introduced, along with their respective strengths and limitations. Then, the ELogic delay model is introduced and its accuracy and stability properties are investigated. Some experimental results that compare the accuracy of the existing switch-level delay model and the ELogic delay model are included at the end of the chapter.

Chapter 5 presents the details of the concept and approaches developed for an accurate timing verifier for MOS digital VLSI systems, referred to as *Electrical-logic based Timing Verifier (E-TV)*. First, a rule-based algorithm which determines the directions of signal flow through MOS transistors is presented. The effect of value-dependent and value-independent events in determining the worst delay predecessor of a path is investigated. Then, the models of synchronous systems and combinational circuits and algorithms for their verification are presented. Like other timing verifiers, *E-TV* breaks feedback loops in a circuit. A novel idea which reduces the number of forward paths blocked by loop-breaking is presented. Following a description of a simple but efficient switch-level simulation algorithm which is used to propagate the effect of nodes that the user sets at logic "1" or "0", the way in which *E-TV* views clocked storage elements along with how it actually verifies combinational logic paths

are presented. Algorithms for the extraction of transistor chains and for determining the worst-case predecessor of an and-or-inverter are presented. In addition, the way *E-TV* deals with dynamic circuits using precharging is described.

In Chapter 6, the performance of *E-TV* is evaluated using two microprocessor designs as test circuits. First, the performance of the path analysis section of *E-TV* and the analysis method (ELogic) used for delay computations are evaluated. Then, the MOS model used in *E-TV* is described and its accuracy is compared to the SPICE MOS2 model.

Conclusions and directions for future work are provided in Chapter 7.

CHAPTER 2

TIMING CONSTRAINTS OF SYNCHRONOUS SYSTEMS

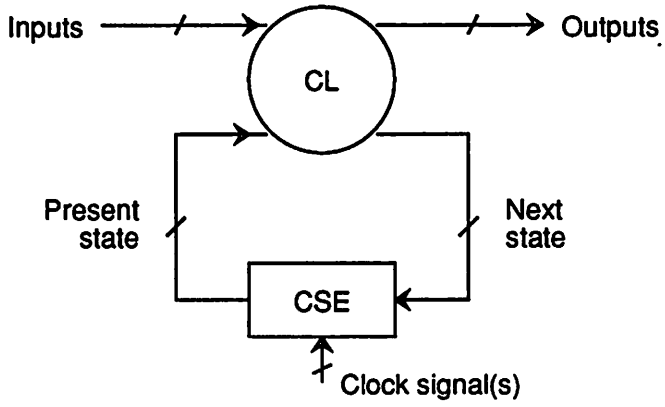
Proper management of signal timing in a digital system is essential for the successful operation of the system. All signals in the system must arrive at intended places in space, and in the intended sequence at the times intended by the designer. Otherwise, the system, while possibly functionally correct, will have timing problems or will fail to achieve its performance objectives. However, as the system size grows larger and the design gets more complex, the precise prediction of propagation delays of signals through the system becomes more difficult and the timing management of all signals is extremely complex. Two design disciplines that alleviate the complex task of managing signals are *self-timed systems* and *synchronous systems* [25,26,27,28]. In self-timed systems, all signals are held up until the slowest one arrives. Sequence and time are connected in the interior of parts called *elements* and all system events are assured to occur in proper sequence rather than at particular times. In synchronous systems, which are most widely used, system-wide clock signals are generated by a timing device called a *clock generator*. These clock signals are used for synchronization by holding up all signals periodically to equalize the delays. Sequence and time are connected through the clock signal.

While designers tend to cope with the complexity of managing signal timing in a systematic way by employing the clocking methodology of a synchronous system, there are still strict timing constraints that synchronous systems must satisfy in order to function correctly. Even though these timing constraints have been described in the literature [25,26,27,28], the description is not sufficiently organized to be used for timing verification. In addition, it is

not complete in the sense that a formulation for general *n*-phase clocking using both edge-triggered and transparent synchronization elements is not available. For example, Glasser [28] presents only the timing constraints of single-phase or two-phase clocking circuits using pass transistor registers whose delay can be ignored. In this chapter, the timing constraints of synchronous systems are reorganized in general forms for timing verification, with a particular emphasis on MOS circuit design techniques. The algorithms for the timing verification of synchronous systems are also presented. The timing constraints that are described are used in the ELogic-based timing verifier (*E-TV*) for the detection of possible timing errors in synchronous MOS systems.

2.1. CLOCKED STORAGE ELEMENTS

The Mealy finite state machine model of a synchronous system is shown in Figure 2.1 [25]. This synchronous system uses storage elements (usually registers or latches) in conjunction with clock signals to hold up the movement of signals to the next stages of the combinational logic until an intended time is reached. The clock pulses generated by a clock generator are distributed throughout the system. These clock signals affect the storage elements in such a way that the elements can latch and store signals only at discrete instants or for the period of time associated with the clock. Storage elements that are used for the global synchronization of system signals, with clock signals, will be referred to as *clocked storage elements (CSE's)* [25] throughout this dissertation. The topological requirement of synchronous systems is that all closed paths of the systems pass through one or more clocked storage elements. The timing constraints for the system depend on the type of clocked storage elements used in addition to the clocking scheme. One common clocked storage elements used in TTL are "edge-triggered" flip-flops. Even though edge-triggered types are sometimes used in



CL: Combinational Logic
CSE: Clocked Storage Element

Figure 2.1 Clocking Scheme of Synchronous Systems

MOS VLSI designs, the "transparent" type is much more common. There are "pulse-width sensitive" elements such as JK master-slave flip-flop, but they are rarely used due to the so-called "glitch-catching" problem [29].

The edge-triggered clocked storage element samples and latches input data value during a short time period (*sampling interval*) around a rising or falling clock edge (*activating clock edge*), and changes its output state on the clock edge. If a rising clock edge causes the element to change the output state, the element is positive edge-triggered type. Otherwise, the element is negative edge-triggered type. Examples of the edge-triggered type are the JK edge-triggered flip-flop and the D edge-triggered flip-flop [29].

The clock node of a transparent clocked storage element works as an ENABLE node, which, when active, makes the element transparent (the output follows the input) [29]. If an element becomes transparent while the ENABLE signal is high, the element is active-high transparent type. Otherwise, the element is active-low transparent type. The transparent type is also an example of the "level-sensitive" type, because the element becomes transparent when the ENABLE signal level is active. The beginning and the end of ENABLE period will be referred to as *activating* and *inactivating* clock edges respectively. Note that, unlike the edge-triggered element, the output of the transparent element follows the input data as soon as it becomes available during ENABLE period. The examples of transparent type are pass transistor registers and D transparent flip-flops [29]. The most common clocked storage elements used in MOS VLSI circuits are pass transistor registers [28].

2.2. DEFINITIONS AND NOTATION

Possibly because the subject of timing constraints has not been dealt with often in the literature, terms and notation are not yet standardized. Therefore, in this section terms and notation are defined so that the timing constraints of synchronous systems can be described clearly and unambiguously. They will be used throughout this dissertation.

[Notation 2.1] $\phi(R)$ and $\phi(F)$ clock edges

For a given clock, ϕ, $\phi(R)$ and $\phi(F)$ clock edges denote the rising (R) and falling (F) edges of ϕ, respectively. □

[Definition 2.1] Clocking by clock phase

Storage elements are said to be *clocked* by $\phi(R)$ or $\phi(F)$, or said to be *clocked* by ϕ or $\bar{\phi}$, if the elements are activated by the rising or falling edge of ϕ. Clocked storage elements

having the same activating clock edge are said to be clocked by *the same clock phase.* □

[Definition 2.2] Clocked path in synchronous systems

A *clocked path* is a signal path which begins and ends at clocked storage elements, referred to as *preceding* clocked storage element (*P_CSE*) and *succeeding* clocked storage element (*S_CSE*). □

The purpose of timing verification of a system is to examine whether or not a signal, which begins to propagate from a specific point in space at a specific point in time, arrives at another specific point in space by a specific point in time. In this dissertation, the start and end time points for timing verification are referred to as a *preceding* and *succeeding time references* (t_{PR} and t_{SR}). The corresponding space points are referred to as *preceding* and *succeeding space references* (s_{PR} and s_{SR}). When a clocked path is subject to timing verification, P_CSE and S_CSE of the clocked path will be referred to as *preceding* and *succeeding reference* clocked storage elements (*PR_CSE* and *SR_CSE*), if necessary, to avoid confusion.

[Definition 2.3] Synchronous loop of clocked paths

A clocked path forms a *synchronous loop* if the preceding and the succeeding clocked storage elements are clocked by the same clock phase. □

Delays through clocked paths with a synchronous loop constrain a clock period.

[Definition 2.4] Single-stage and multistage clocked paths

A *single-stage clocked path* is a clocked path which has only two clocked storage elements in the path: preceding and succeeding elements. A *multistage clocked path* is a clocked path which consists of more than one contiguous single-stage clocked path, without

synchronous loops unless the whole path forms a synchronous loop. An N-stage clocked path refers to a multistage clocked path consisting of N single-stage clocked paths. □

Single-stage clocked paths are basic units to consider for timing verification. Multistage clocked paths exists only in multiphase clocking systems. No more than one storage element in a multistage clocked path is clocked by the same clock phase, except when the whole path forms a synchronous loop or transparent clocked storage elements are consecutive. The clocked path of ϕ_i and $\bar{\phi}_j$ represents a clocked path which begins logic evaluation at $\phi_i(R)$ and finishes it at $\phi_j(F)$, when P_CSE and S_CSE are ideal clocked storage elements without delays. Figures 2.2 and 2.3 illustrate ϕ_i - ϕ_j single-stage and multistage clocked paths, where P_SCE and S_CSE are positive edge-triggered type.

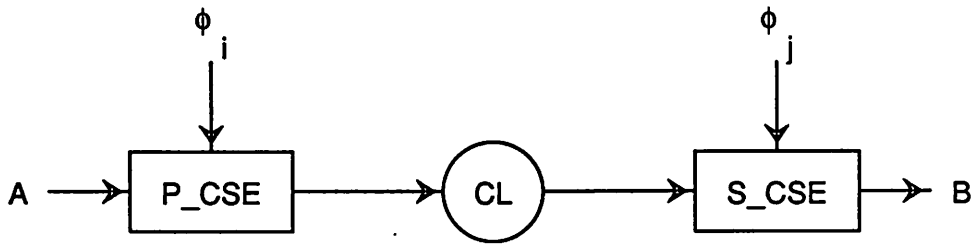
[Definition 2.5] Clocked logic segments (single-stage and multistage)

A ϕ_i - ϕ_j single-stage (multistage) clocked logic segment in a digital system is a collection of all ϕ_i - ϕ_j single-stage (multistage) clocked paths in the system. □

[Definition 2.6] Logic evaluation sections

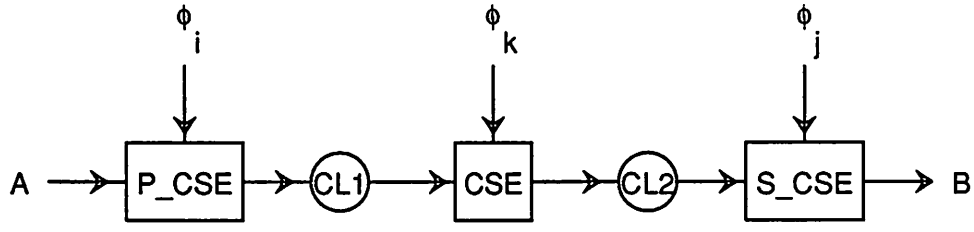
A ϕ_i-*logic evaluation section* in a digital system is a collection of all single-stage logic segments in the system, whose preceding storage elements are clocked by ϕ_i. □

There may be as many as N^2 single-stage clocked logic segments and N logic evaluation sections in a circuit under analysis, where N is the number of clock edges which are used for synchronization. Assuming that no single-stage logic segments form a synchronous loop, there may be $N(N-1)$ single-stage clocked logic segments. Even though there are $2n$ clock edges available for synchronization in n-phase clocking systems, system designers usually use n rising or n falling edges only.



CL: Combinational Logic Path
P_CSE: Preceding Clocked Storage Element
S_CSE: Succeeding Clocked Storage Element

Figure 2.2 Single-Stage Clocked Path



CL: Combinational Logic Path, CSE: Clocked Storage Element
P_CSE: Preceding CSE, S_CSE: Succeeding CSE

Figure 2.3 Multistage Clocked Path ($i \neq k$, $j \neq k$)

[Notation 2.2] Propagation delay

For a given logic path A, the propagation delay through A is denoted by D_A. □

[Notation 2.3] Activating and inactivating clock edge

For a given clocked storage element, CSE, $t_{AE}(CSE)$ denotes times at which the activating clock edges of CSE occur. For a given transparent clocked storage element, CSE, $t_{IAE}(CSE)$ denote the times at which the inactivating clock edges of CSE occur. □

2.3. CLOCKED PATHS USING EDGE-TRIGGERED CLOCKED STORAGE ELEMENTS

When edge-triggered clocked storage elements are used, it is important to keep the input data unchanged during the sampling interval around an activating clock edge to guarantee that the correct input values are latched. The part of the error-free sampling interval before the activating clock edge is called a *set-up time*, and the one after the activating clock edge is called a *hold time* [26, 29]. If the slowest arriving signals settle at the input nodes of clocked storage elements the set-up time before an activating clock edge, they will remain stable until the end of the hold time, unless logic paths are extremely fast. Since the hold time is usually negligibly smaller than propagation delays through logic paths, many timing verifiers check only whether or not the input data of clocked storage elements become stable before the set-up time.

[Notation 2.4] Setup time of clocked storage element

For a given clocked storage element, CSE, $T_{SETUP}(CSE)$ denotes the set-up time of CSE. □

Consider a single-stage clocked path, illustrated in Figure 2.2. Suppose that P_CSE and S_CSE are edge-triggered type. Then, the propagation delay of the clocked path, D_{CL}, must satisfy following constraints to ensure that input data to S_CSE settles before the set-up time:

$$D_{CL} \leq t_{SR} - t_{PR} = [\, t_{AE}(S\_CSE) - T_{SETUP}(S\_CSE) \,] - t_{AE}(P\_CSE) \qquad (2.1)$$

Note that $t_{AE}(S\_CSE)$, a timepoint at which the activating clock edge of S_CSE occurs, is the next closest one to $t_{AE}(P\_CSE)$ on timing chart. If clock signals shown in Figure 2.4 are used for synchronization and the two clocked storage elements are positive edge-triggered type, following constraints must be satisfied:

$$D_{CL} \leq t3 - t1 - T_{SETUP}(S\_CSE)$$

A multistage clocked path also must satisfy the timing constraints of Equation (2.1). In this case, D_{CL} represents a propagation delay through the multistage clocked path; D_{CL} is the sum of delays through single-stage clocked paths which constitute the multistage clocked

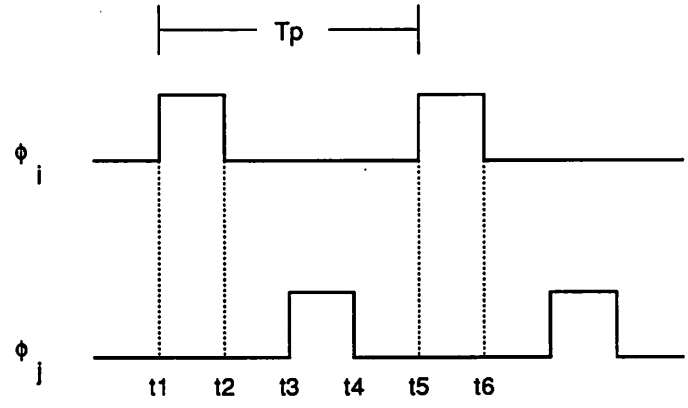


Figure 2.4 Clock Signals

path. However, fortunately, any multistage clocked path using edge-triggered clocked storage elements, including the one which constrains a clock period, satisfies timing constraints automatically if each of its constituent single-stage clocked paths satisfies timing constraints. Thus, when verifying a system which uses edge-triggered clocked storage elements, timing verifiers obtain the maximum propagation delay through each single-stage logic segment and apply Equation (2.1) for the timing verification of the system.

2.4. CLOCKED PATHS USING TRANSPARENT CLOCKED STORAGE ELEMENTS

In a system using transparent clocked storage elements, input signals of clocked storage elements must remain stable during a set-up time for correct values to be latched. The set-up time of transparent type is a short time interval just before the inactivating clock edge.

Consider a single-stage clocked path, illustrated in Figure 2.2, once more. This time, clocked storage elements used are assumed to be transparent type. To ensure the correct input data to be latched, the propagation path delay, D_{CL}, must satisfy:

$$D_{CL} \leq t_{SR} - t_{PR} = [\, t_{END}(S\_CSE) - T_{SETUP}(S\_CSE) \,] - t_{AE}(P\_CSE) \qquad (2.2)$$

where

$t_{END}(S\_CSE)$: $t_{AE}(S\_CSE)$, if a clocked path forms a synchronous loop.
$t_{IAE}(S\_CSE)$, otherwise.

In the above constraints, $t_{END}(S\_CSE)$ is the next closest one to $t_{AE}(P\_CSE)$ on the timing chart. If a particular single-stage clocked path forms a synchronous loop (i.e., $i = j$ in Figure 2.2), the path employs a single-phase clocking scheme. In this case, D_{CL} constrains a clock period. Otherwise, D_{CL} constrains a *clock separation*, which is a separation between clock

edges of different phase. Note that a single-stage clocked path using transparent elements may use both preceding and succeeding clock phases for logic evaluation, unless it forms a synchronous loop. In Figure 2.2, if P_CSE and S_CSE are positive-active transparent type and clock signals illustrated in Figure 2.4 are used for synchronization, Equation (2.2) yields following constraints:

$$D_{CL} \leq t4 - t1 - T_{SETUP}(S\_CSE)$$

[Definition 2.7] Order of clock edges

For a given clocked path, clocked logic segment, or logic evaluation section for timing verification, the *order of clock edges* is the one that clock edges occur on a timing chart during one clock period, beginning from a preceding activating clock edge. □

Note that an order of clock edges is not an order of clock signals specified by the user. Suppose that a single-stage clocked path of *CSE2-CL2-CSE3* in Figure 2.5 is verified. If CSE2 is positive-active transparent type, a preceding activating clock edge is $\phi3(R)$. Thus, the increasing order of clock edges for the path is $\phi3(R)$ - $\phi3(F)$ - $\phi4(R)$ - $\phi4(F)$ - $\phi1(R)$ - $\phi1(F)$ - $\phi2(R)$ - $\phi2(F)$.

[Definition 2.8] Clocking delay of clocked storage elements

When an input signal arrives at a clocked storage element, CSE, before an activating clock edge does, it can not be latched until an activating clock edge arrives. A *clocking delay (CSE)* is defined as a time interval the input signal must wait to be latched as follows:

$$clocking\ delay\ (CSE) = \min\left(t_{AE}(CSE) - t_{READY}(CSE),\ 0\right) \tag{2.3}$$

where $t_{READY}(CSE)$ is a timepoint at which an input data of CSE is ready. □

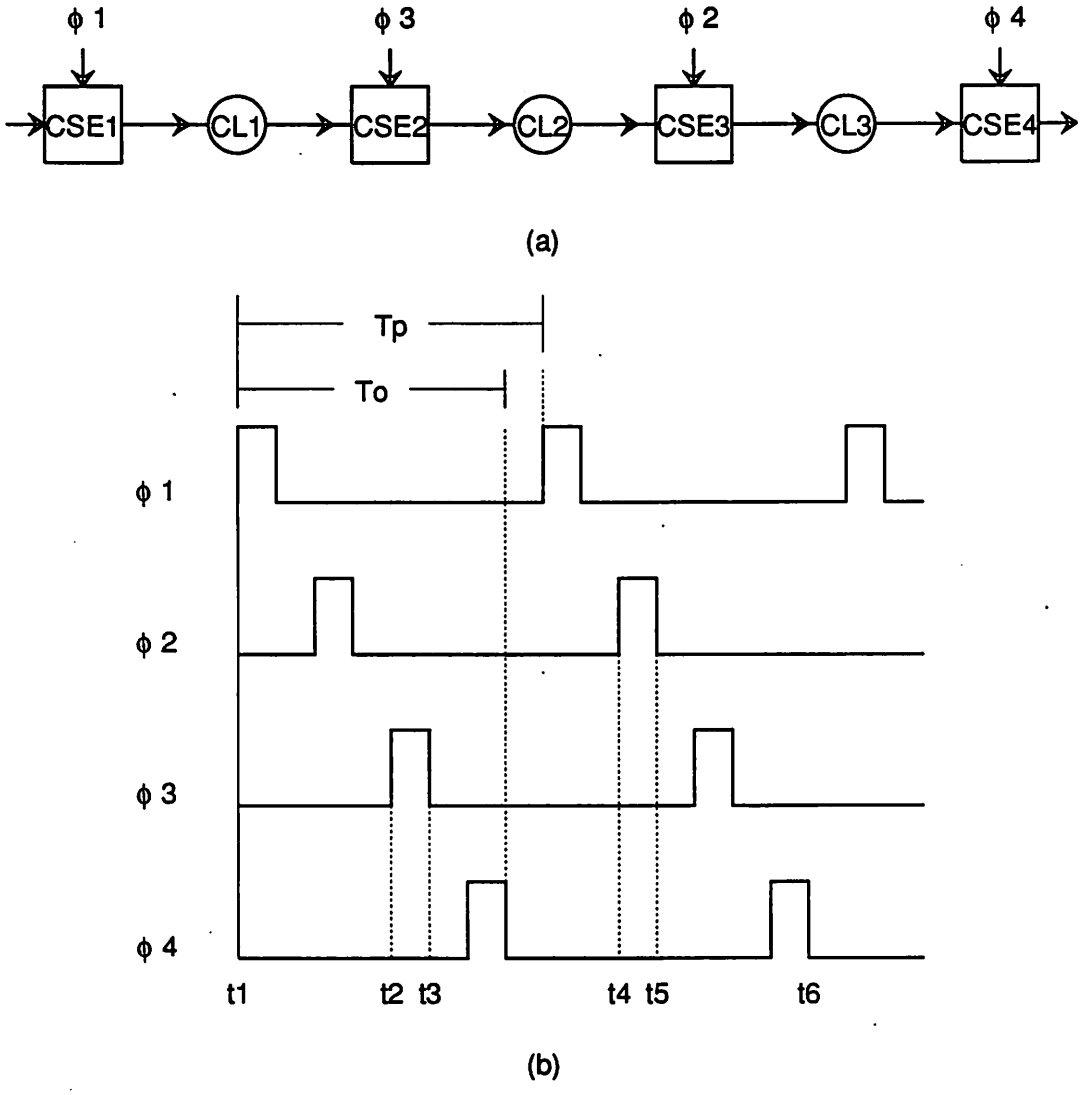


Figure 2.5 3-Step Clocked Path in A 4-Phase Clocking Scheme

If an input signal arrives at a clocked storage element on or after an activating clock edge, the clocking delay is zero. The concept of the clocking delay will be used for the description of the algorithms for the verification of multistage clocked paths which use transparent clocked storage elements.

The timing constraints of Equation (2.2) can be applied to examine multistage clocked paths as well as single-stage clocked paths, if the meaning of D_{CL} is modified as follows:

$$D_{CL} \leq t_{SR} - t_{PR} = [\, t_{END}(SR\_CSE) - T_{SETUP}(SR\_CSE) \,] - t_{AE}(PR\_CSE) \qquad (2.4)$$

where

D_{CL}	:	Propagation time through a clocked path, excluding clocking delays of interposed clocked storage elements.
$t_{END}(SR\_CSE)$	:	$t_{AE}(SR\_CSE)$, if a clocked path forms a synchronous loop. $t_{IAE}(SR\_CSE)$, otherwise.

Note that D_{CL} is a sum of delays through successive single-stage clocked paths and it is not a sum of the worst-case delays in successive single-stage clocked logic segments. Like a single-stage clocked path, if a multistage clocked path forms a synchronous loop, it constrains a clock period. If not, it constrains a clock separation. However, $t_{END}(SR\_CSE)$ for a multistage clocked path may not be the next closest one to $t_{AE}(PR\_CSE)$ as it is for a single-stage clocked path. A time difference between $t_{END}(SR\_CSE)$ and $t_{AE}(PR\_CSE)$ in Equation (2.4) is equal to or larger than a clock period, if clock edges activating storage elements between PR_CSE and SR_CSE are not in monotonically increasing order. Therefore, when $t_{AE}(PR\_CSE)$ is given for a multistage clocked path, $t_{END}(SR\_CSE)$ must be computed by considering the evaluation period of each of its constituent single-stage clocked paths on the timing chart, one by one from the first to the last single-stage clocked paths. As an example, let all the clocked storage elements in Figure 2.5 be positive-active transparent type. Assume

that $t_{AE}(CSE\,1)$ is $t1$ on timing chart. Then, the evaluation periods for CL1, CL2, and CL3 are from $t1$ to $t3$, $t2$ to $t5$, and $t4$ to $t6$, respectively. Since the path does not form a synchronous loop, the proper $t_{END}(CSE\,4)$ is $t6$, which is $T_P + T_0$ apart from $t_{AE}(CSE\,1)$.

When a system uses transparent clocked storage elements, a multistage clocked path may not satisfy timing constraints, even when all of its constituent single-stage clocked paths satisfy their timing constraints. Therefore, when a multistage clocked path is examined, all of its nested clocked paths must be examined and assessed using Equation (2.4). If a multistage clocked path has N constituent single-stage clocked paths, it has $\sum_{j=1}^{N} j$ nested clocked paths. Since the path of Figure 2.5 has 3 constituent single-stage paths, the following 6 nested clocked paths must satisfy their timing constraints as follows:

(1) Single-stage nested clocked paths

$$D_{CL1} \leq t3 - t1 - T_{SETUP}(CSE\,2)$$
$$D_{CL2} \leq t5 - t2 - T_{SETUP}(CSE\,3)$$
$$D_{CL3} \leq t6 - t4 - T_{SETUP}(CSE\,4)$$

(2) 2-stage nested clocked paths

$$D_{CL1} + D_{CL2} \leq t5 - t1 - T_{SETUP}(CSE\,3)$$
$$D_{CL2} + D_{CL3} \leq t6 - t2 - T_{SETUP}(CSE\,4)$$

(3) 3-stage nested clocked path

$$D_{CL1} + D_{CL2} + D_{CL3} \leq t6 - t1 - T_{SETUP}(CSE\,4) = T_P + T_0 - T_{SETUP}(CSE\,4)$$

The algorithm for the verification of *N-stage* clocked paths are shown in Algorithm 2.1, in a high-level Pidgin-C description.

As mentioned earlier, when transparent clocked storage elements are used, the fact that all constituent single-stage clocked paths satisfy their timing constraints does not guarantee that the corresponding multistage clocked path satisfies its timing constraints. However,

```
/* START: Constraint check for a N-stage clocked path, MCP , using transparent type */
foreach ( j ∈ {1, ... , N} ) {
   extract all j-stage nested clocked paths from MCP ;
   foreach ( Nest_CP_i ∈ {set of j-stage nested clocked paths} ) {
      check Nest_CP_i using Equation (2.4);
   }
}
/* END: Constraint check for MCP */
/* Note: D_CL in Equation (2.4) is the sum of delays through the constituent
   single-stage paths of CL, which does not include a clocking delay. */
```

Algorithm 2.1 Constraint Check For A *N-stage* Clocked Path Using Transparent Type

there has been an attempt to examine a multistage path on the fly, while verifying constituent single-stage clocked paths [24, 30]. A key idea of this approach is to examine and evaluate a path from a preceding reference clocked storage element, which changes during verification, to the end of a constituent single-stage clocked path which is most recently examined and evaluated. The approach is described in Algorithm 2.2. In the description, the boxed part is used later in this chapter to describe the algorithm for the verification of clocked paths that use both types of storage elements. Suppose that a multistage clocked path *MCP*, which uses transparent clocked storage elements, is subject to timing verification using Equation (2.2). Let a preceding reference clocked storage element (*PR_CSE*) be the preceding clocked storage element of *MCP* initially. A signal starts to propagate from *PR_CSE* at $t_{AE}(PR\_CSE)$. Algorithm 2.2 examines each single-stage clocked path, one by one from the first one to the last one. Suppose that SCP_i denotes the i th constituent single-stage clocked

```
/* START: Constraint check for a N-stage clocked path, MCP , using transparent type */

PR_CSE = P_CSE of MCP ;
Propagate a signal at t_AE(PR_CSE);

foreach (SCP_i = i th constituent single-stage clocked path of MCP , i ∈ {1, ... , N}) {
    Check SCP_i using Equation (2.2);
    if ( PR_CSE ≠ P_CSE of SCP_i ) {
      if (P_CSE of MCP to S_CSE_i forms a synchronous loop) {
        /* S_CSE_i must be the last single-stage clocked path of MCP */
        if ( t_READY(S_CSE_i) > (t_AE(S_CSE_i) − t_SETUP(S_CSE_i)) )
          report a clock period violation;
      }

      S_CSE_i = S_CSE of SCP_i ;

      if ( t_READY(S_CSE_i) ≤ t_AE(S_CSE_i) ) {
        /* CASE 1: Path from PR_CSE to S_CSE_i satisfies constraints */
        Delay a signal propagation until S_CSE_i becomes active;
        PR_CSE = S_CSE_i ;
      }
      else {

        if ( t_READY(S_CSE_i) > (t_IAE(S_CSE_i) − T_SETUP(S_CSE_i)))
          /* CASE 2: Path from current PR_CSE to S_CSE_i violates constraints */
          report a clock separation violation;

        else if (SCP_i is the last one in MCP) ; /* CASE 3: Constraints satisfied */
        else   /* CASE 4: */
          /* Path from PR_CSE to S_CSE_i is still subject to verification */
      }
    }
}       /* END: Constraint check for MCP */
```

where

PR_CSE	:	Preceding reference clocked storage element
$P\_CSE\ (S\_CSE)$	:	Preceding (Succeeding) clocked storage element
$t_{READY}(S\_CSE_i)$	:	Timepoint at which an input of S_CSE_i is ready

Algorithm 2.2 Constraint Check For A *N-stage* Clocked Path Using Transparent Type

path of MCP and S_CSE_i denotes its succeeding clocked storage element. In general, after examining SCP_i, Algorithm 2.2 checks if a path from the preceding clocked storage element of MCP to S_CSE_i forms a synchronous loop. If the path forms a synchronous loop, SCP_i must be the last constituent single-stage clocked path of MCP from the definition. Hence a signal propagation through the path must finish by $t_{SR}(S\_CSE_i)$ to satisfy the following inequality:

$$t_{READY}(S\_CSE_i) > t_{SR}(S\_CSE_i) = t_{AE}(S\_CSE_i) - t_{SETUP}(S\_CSE_i)$$

where $t_{READY}(S\_CSE_i)$ is an input signal arrival time at S_CSE_i. If the path does not form a synchronous loop, there are two cases to consider:

(1) An input signal arrives before an activating clock edge at S_CSE of SCP_i.

(2) An input signal arrives after an activating clock edge at S_CSE of SCP_i.

If an input signal is ready before an activating clock edge to arrive at S_CSE of SCP_i, a pending path from the current PR_CSE to the S_CSE of SCP_i satisfies maximum delay constraints (CASE 1 in Algorithm 2.2). In this case, a signal propagation is delayed until S_CSE of SCP_i is activated; i.e., a clocking delay is added to the path delay. The signal starts to propagate through the path again when S_CSE of SCP_i becomes active, as it starts to propagate at P_CSE of MCP when it becomes active. Therefore, PR_CSE is updated by the S_CSE of SCP_i for the examination of the remaining path of MCP. On the other hand, when an input signal arrives after an activating clock edge at S_CSE of SCP_i, one of following two cases happens:

(2.1) An input signal does not arrive by a set-up time before an inactivating clock edge.

(2.2) An input signal arrives by a set-up time before an inactivating clock edge.

If an input signal does not arrive at S_CSE of SCP_i by a set-up time before an inactivating clock edge, a pending path for timing verification, which is a path from the current PR_CSE

to S_CSE of SCP_i, violates timing constraints (CASE 2 in Algorithm 2.2). When an input signal arrives by a set-up time before an inactivating clock edge, if SCP_i is the last single-stage clocked path in *MCP*, the pending path is regarded as satisfying timing constraints (CASE 3 in Algorithm 2.2). If it is not the last one, the pending path is still subject to timing verification (CASE 4 in Algorithm 2.2). The algorithm moves to the next constituent single-stage clocked path to examine while PR_CSE is kept the same.

Figure 2.6 illustrates CASE 1 to CASE 4 in Algorithm 2.2, assuming that the set-up times of NMOS pass transistor registers are negligible. In CASE 1, the input of CSE2 settles before $t2$ which is $t_{AE}(CSE2)$. Thus, CL1 satisfies timing constraints and $d2$ represents a clocking delay through CSE2. In CASE 2, the input of CSE2 settles after $t3$ which is $t_{IAE}(CSE2)$ and CL1 violates timing constraints. In CASE 4, the input of CSE2 settles between $t2$ and $t3$ and CL1 is pending for verification. CASE 1' to CASE 4' in Figure 2.6 also illustrate CASE 1 to CASE 4 in Algorithm 2.2, respectively, involving more than one single-stage clocked path. In CASE 1, pending paths CL1 and CL2 satisfy timing constraints. while they fail in CASE 2. From CASE 2, notice that a multistage path may violate timing constraints, even though its constituent single-stage paths satisfy timing constraints individually. In CASE 3, the early part illustrates how a signal propagates after CASE 1 has occurred. After CL1 is examined, PR_CSE changes from CSE1 to CSE2. Because CL3 is the last single-stage path in this particular example, a multistage path of CL2 and CL3 satisfies timing constraints. In CASE 4, a multistage path of CL1 and CL2 is still subject to verification.

Clock pulse width (ENABLE period) must be wide enough so that the output nodes of transparent clocked storage elements can settle. However, it is easily satisfied and, therefore, the designers are not concerned about the clock pulse width unless they use a single-phase

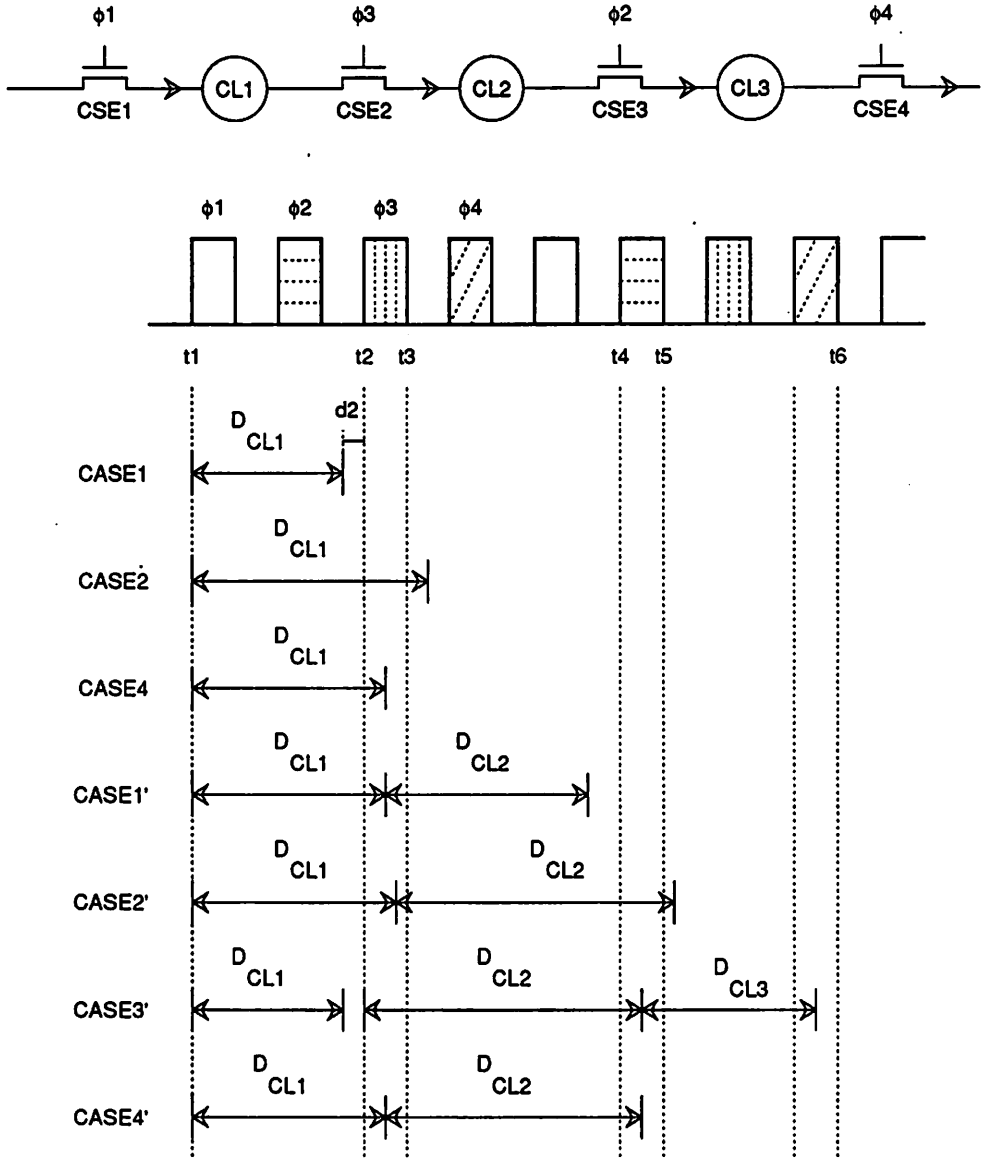


Figure 2.6 Checking Timing Constraints For A Multistage Clocked Path

clocking scheme which limits the clock pulse width, as will be described later.

2.5. CLOCKED PATHS USING EDGE-TRIGGERED AND TRANSPARENT CLOCKED STORAGE ELEMENTS

Circuit designers may use both edge-triggered and transparent clocked storage elements in one design. Thus, the types of preceding and succeeding clocked storage elements may be different and so it is helpful to derive timing constraints and algorithms that are applicable regardless of the element types used. A clocked path takes an activating clock edge as a preceding time reference (t_{PR}), whatever the type of a preceding clocked storage element is. However, a succeeding time reference (t_{SR}) depends on the type of a succeeding clocked storage element (S_CSE). If S_CSE is edge-triggered type or the path forms a synchronous loop, an activating clock edge will be t_{SR}. Otherwise an inactivating clock edge will be t_{SR}. Therefore, the timing constraints and algorithms presented earlier can be modified to reflect above facts for the timing verification of the clocked paths using either type of elements.

From Equation (2.1) and (2.2), timing constraints for a single-stage clocked path illustrated in Figure 2.2 are given as follows:

$$D_{CL} \leq t_{SR} - t_{PR} = [\, t_{END}(S\_CSE) - T_{SETUP}(S\_CSE) \,] - t_{AE}(P\_CSE) \tag{2.5}$$

where

$t_{END}(S\_CSE)$: $t_{AE}(S\_CSE)$, if S_CSE is edge-triggered type or the path forms a synchronous loop.
$t_{IAE}(S\_CSE)$, otherwise.

and $t_{END}(S\_CSE)$ is the next closest one to $t_{AE}(P\_CSE)$.

As mentioned earlier, edge-triggered clocked storage elements partition a multistage clocked path into sub-paths whose timing verification can be carried out independently of

each other. If all the partitioned sub-paths satisfy their timing constraints, the multistage clocked path is also guaranteed to satisfy its timing constraints. Therefore, the timing verification of a multistage clocked path using both types of clocked storage elements can be translated into a task of examining each sub-path separately, provided that they are partitioned by interposed edge-triggered clocked storage elements. If a multistage path has N interposed edge-triggered clocked storage elements between its preceding and succeeding clocked storage elements, there are $N+1$ sub-paths to verify independently. The sub-path is a single-stage or multistage clocked path. If a sub-path is multistage, all of its interposed clocked storage elements are transparent type. There are four types of sub-paths as listed in Table 2.1.

Sub-Path type	CSE Type	
	Preceding CSE	Succeeding CSE
Type 1	Edge-triggered	Edge-triggered
Type 2	Edge-triggered	Transparent
Type 3	Transparent	Edge-triggered
Type 4	Transparent	Transparent

Table 2.1 Types Of Sub-Paths Which Can Be Verified Individually After Partitioning A Multistage Clocked Path

Algorithms 2.1 and 2.2 can be modified as follows to check if each sub-path satisfies timing constraints.

Modified Algorithm MA2.1:

$t_{END}(S\_CSE_i)$ for Equation 2.4 is redefined as follows:

$t_{END}(SR\_CSE)$: $t_{AE}(SR\_CSE)$, if SR_CSE is edge-triggered type or
a clocked path forms a synchronous loop.
$t_{IAE}(SR\_CSE)$, otherwise.

Modified Algorithm MA2.2:

The boxed part in Algorithm 2.2 is modified to followings:

Check SCP_i using Equation (2.5);
if ($PR\_CSE \neq P\_CSE$ of SCP_i) {
  if (P_CSE of MCP to S_CSE_i forms a synchronous loop or
    S_CSE_i is edge-triggered type) {

2.6. SINGLE-PHASE CLOCKING

The simplest clocking scheme, even though its timing constraints are not simple to satisfy, is a single-phase scheme which uses only one clock phase [25,28]. Since the same clock phase is applied to all clocked storage elements, all clocked paths form synchronous loops. Thus, clocked path delays must be smaller than a clock period. More precisely, from Equation (2.5), the maximum delay D^{Max} of the combinational logic in a single-phase clocking system is constrained as follows, irrespective of clocked storage element types:

$$D^{Max} \leq T_P - T_{SETUP}(S\_CSE) \tag{2.6}$$

where T_P is a clock period and S_CSE is a succeeding clocked storage element.

Suppose a single-phase clocking system uses transparent clocked storage elements. Since only one clock phase is used, all clocked storage elements in the system are active during an ENABLE period. If a combinational logic has path delays that are smaller than the ENABLE period, the newly generated next states are latched as inputs during the same ENABLE period. If such *race* conditions exist, the system will not function correctly. Therefore, the minimum path delay of the combinational logic, D^{Min}, must satisfy following

constraints:

$$T_{ENABLE} \leq D^{Min} + T_{OH}(S\_CSE) \tag{2.7}$$

where T_{ENABLE} is an ENABLE period for which clocked storage elements in the system become transparent, and T_{OH} is a hold time. Equations (2.6) and (2.7) mean that not only the slowest path but also the fastest path in the system must be observed.

The single-phase clocking is cheap and fast. However, if transparent clocked storage elements are used for synchronization, it is difficult to implement combinational logic block which satisfies the *two-sided constraints* of Equations (2.6) and (2.7) under all conditions of process variation in manufacture. This clocking scheme was employed in many of the early TTL systems, but is not popular in MOS VLSI circuits where transparent pass transistor registers are commonly used.

2.7. MULTIPHASE CLOCKING

The advantage of using multiphase clocking is that the system can be free of two-sided constraints and the combinational logic may have *single-sided* relations which constrain the maximum path delay, even when transparent clocked storage elements are used. As in single-phase clocking system, race conditions may occur in multiphase clocking systems from synchronous-looped paths whose constituent clocked storage elements are all transparent type. However, race conditions can be avoided by choosing clock signals properly for a given clocking scheme and, thus, the painful minimum delay constraints can be eliminated. One of the most common ways to avoid race conditions is to use nonoverlapping clock signals so that only one transparent clocked storage element in any synchronous-looped path can be active at any time. In fact, even if overlapping clock signals are used, race conditions do not occur as long as not all transparent clocked storage elements of the synchronous

looped-path are active at the same time. One of the examples is the canonical three-phase overlapping clocking scheme, where only two phases, not three phases, simultaneously overlap [28]. If an overlapping clocking scheme presents two-sided constraints due to simultaneous overlapping of all clock signals, the precise constraints on the minimum delay to avoid a race depends on the particular scheme used.

Examination of the maximum delays through the combinational logic of a multiphase clocking system is partitioned into two tasks:

(1) Identifying all single-stage and multistage clocked paths in a system that make a synchronous loop or terminate prematurely before making a synchronous loop.

(2) Verifying identified clocked paths against timing constraints, using Modified Algorithm MA2.1 or MA2.2.

The above tasks can be performed simultaneously. Figure 2.7 illustrates a system which has two isolated closed paths: *Path 1* and *Path 2*. If a timing verifier starts a clocked-path search from only ϕ_1 clock nodes, it detects *Path 1* but misses *Path 2*. In general, when a timing verifier looks for clocked paths in a system, the search must start from all clock nodes in order to find all clocked paths.

2.8. PRECHARGING

Precharging is a quite useful technique in MOS circuit designs, because it provides the advantages of faster operation, reduced power consumption, and greater circuit density, depending on the application [29, 28]. However, once the stored charge is lost, it cannot be recovered until the next precharging time. Thus, circuits become more sensitive to glitches and timing errors and have tight timing constraints to satisfy. Figure 2.8 illustrates a part of circuit which uses a precharged bus. In the figure, CSE1 and CSE2 are NMOS transmission

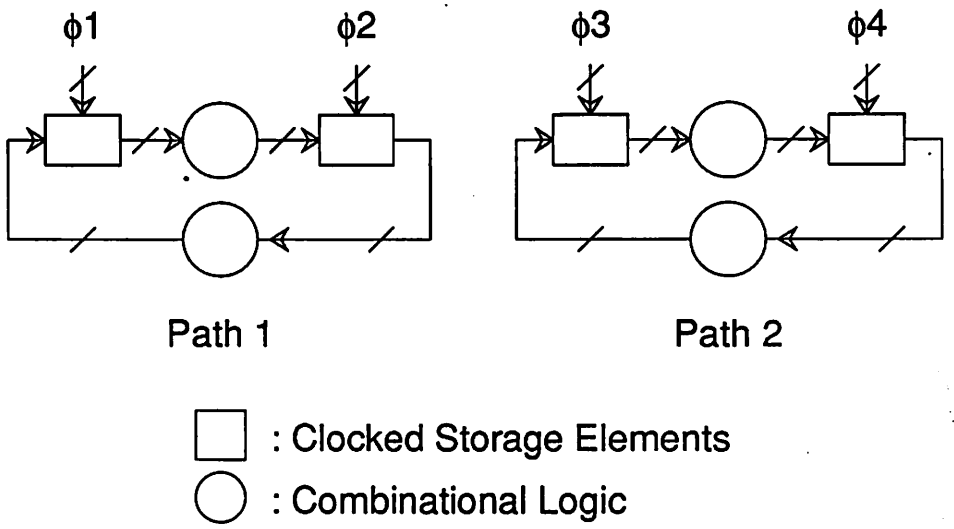


Figure 2.7 A Synchronous System with Two Separate Closed Paths

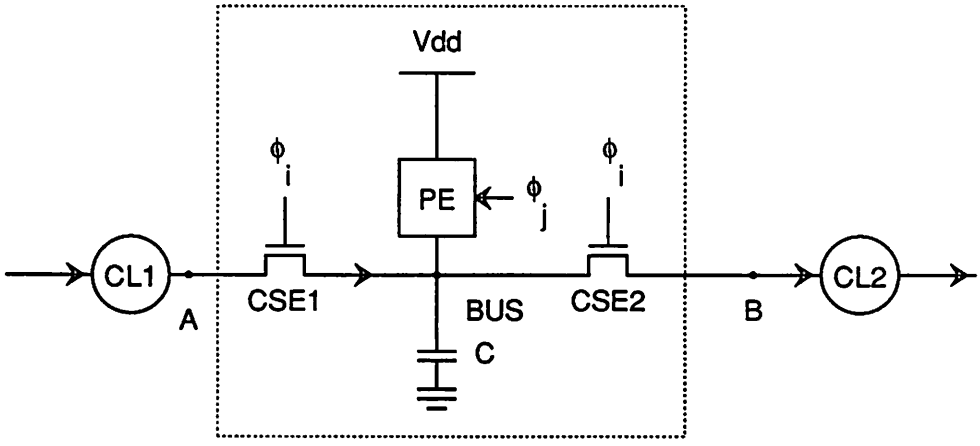


Figure 2.8 Precharged Module

gates. A precharging element (*PE*) may be a PMOS or an NMOS transistor. During a precharging period, the precharging element is on and *BUS* precharges. Then it discharges through CSE1 depending on the logic signal at *A* during an evaluation period (ϕ_i). Since *BUS* takes its input signal from Node *A* while ϕ_i is high, it belongs to $\phi_i(R)$ logic evaluation section.

If a precharging element is an NMOS transistor, *BUS* is precharged while ϕ_j is high, where ϕ_j and ϕ_i are usually nonoverlapping, as illustrated in Figure 2.4. Obviously, a precharging time through an NMOS precharging element, T_{PRECHG}, must satisfy followings:

$$T_{PRECHG} \leq t_{OFF}(PE) - t_{ON}(PE) = \phi_j(F) - \phi_j(R) \tag{2.8.a}$$

When a precharging element is a PMOS transistor, ϕ_j is usually ϕ_i. Thus, the precharging time through a PMOS precharging element, T_{PRECHG}, must satisfy the following:

$$T_{PRECHG} \leq t_{OFF}(PE) - t_{ON}(PE) = \phi_j(R) - \phi_j(F) = \phi_i(R) - \phi_i(F) \tag{2.8.b}$$

[Notation 2.5] Rising and falling transition times

For a given node *A*, $t_A(R)$ and $t_A(F)$ are times at which rising and falling transitions occur, respectively. □

Note that precharging uses dynamic charge storage. If Node *A* does not complete its rising transition before CSE1 is on, the charge stored on *BUS* will be lost. Therefore, Node *A* must complete its rising transition by the time that CSE1 is clocked on:

$$t_A(R) \leq \phi_i(R) \tag{2.9}$$

Provided that Equation (2.9) is satisfied, a logic "1" signal is available at *BUS* as soon as the precharging is done. Hence when a timing verifier needs to propagate the worst-case rising transition at *BUS* to the next logic gates, it propagates a rising transition through a precharg-

ing path, and all other rising transitions are ignored.

Since an output transmission gate CSE2 is also gated by ϕ_i, the rising and falling transitions at Node B through CSE2 must finish before CSE2 turns off:

$$t_B(F) \leq \phi_i(F) \tag{2.10.a}$$

$$t_B(R) \leq \phi_i(F) \tag{2.10.b}$$

Since a logic "0" signal at Node A starts to pass through CSE1 at $\phi_i(R)$, Equation (2.10.a) means that a falling delay through CSE1 and CSE2, D^F_{PC}, must not be larger than the pulse width of ϕ_i for which ϕ_i is high:

$$D^F_{PC} \leq \phi_i(F) - \phi_i(R) \tag{2.11.a}$$

Similarly, a logic "1" signal at *BUS* starts to pass through CSE2 at $\phi_i(R)$. Thus, Equation (2.10.b) means that a rising delay through CSE2, D^R_{PC}, must not be larger than the pulse width of ϕ_i:

$$D^R_{PC} \leq \phi_i(F) - \phi_i(R) \tag{2.11.b}$$

One should notice that, while circuits using precharged modules are faster than static circuits, they have to satisfy tight timing constraints. A circuit example using precharged modules is illustrated in Figure 2.9. Suppose that CSE3 is an active-high transparent type with a negligible set-up time. Then Equations (2.4), (2.8), (2.9), (2.11.a) and (2.11.b) yield the following constraints to satisfy:

From Equation (2.4): $D + D_{CL2} \leq \phi_j(F) - \phi_i(R) = t4 - t1$

$$D_{CL1} + D < \phi_i(F) - \phi_j(R) = t6 - t3$$

$$D + D_{CL1} + D_{CL2} \leq T_P \tag{2.12}$$

where D is the path delay through CSE 1 and *CSE* 2

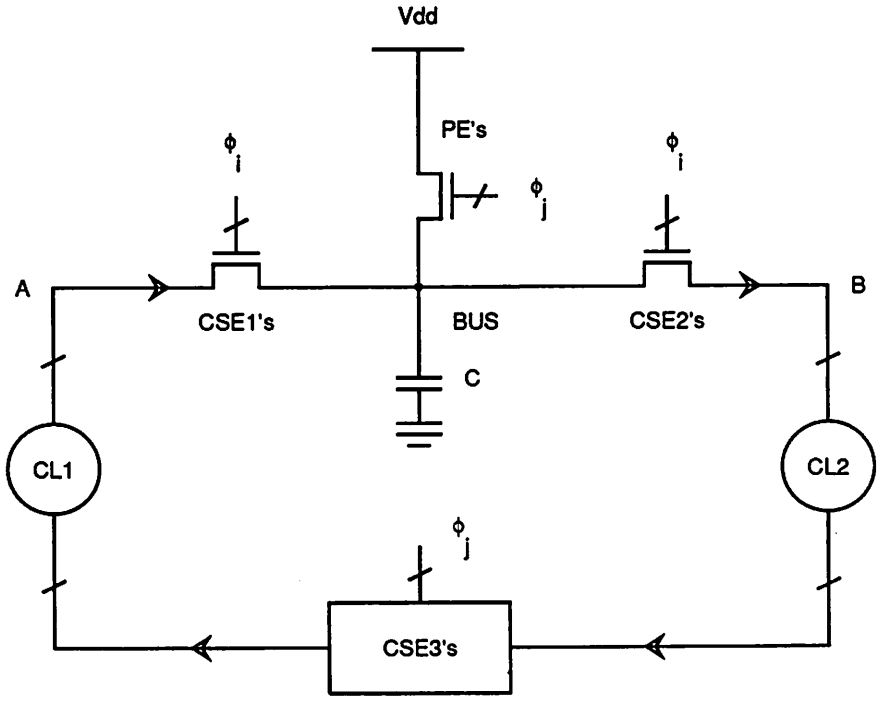


Figure 2.9 A Circuit Example Which Uses A Precharged Module

From Equation (2.8): $T_{PRECHG} \leq \phi_j(F) - \phi_{(R)} = t4 - t3$

From Equation (2.9): $t_A(R) \leq t1$, for *an evaluation period of* t_1 and t_2 (2.13)

From Equation (2.11.*a*): $D_{FC} \leq t2 - t1$

From Equation (2.11.*b*): $D_{BC} \leq t2 - t1$

As mentioned earlier, the first three constraints, derived from Equation (2.4), apply to successive clocked paths rather than successive clocked logic segments in the system. As an illustration, $(D + D_{CL1} + D_{CL2})$ in Equation (2.12) is a sum of delays through one of precharged modules, a path in CL2, and a path in CL1, that are successive. It is not the sum of the

worst-case delay through precharged modules, the worst-case delay in CL2, and the worst-case delay in CL1.

Two other circuit modules that use precharging and predischarging are illustrated in Figure 2.10: N-type and P-type dynamic logic gates [31, 32]. The N-type dynamic logic gate is used in both NMOS and CMOS designs, while the P-type dynamic logic gate is available only in CMOS designs. In NMOS design, *M1* is an NMOS transistor, and ϕ_i is $\bar{\phi}_j$ or another clock signal which is nonoverlapping with ϕ_j. The output node, *Output*, is precharged when ϕ_i is high and is conditionally discharged through N-type logic block while ϕ_j is high. These N-type dynamic logic gates are used as precharged busses in NMOS design. In CMOS design, *M1* of the N-type dynamic logic gate is a PMOS transistor, and ϕ_i is ϕ_j. *Output* is precharged when ϕ_j is low and is conditionally discharged while ϕ_j is high. N-type dynamic

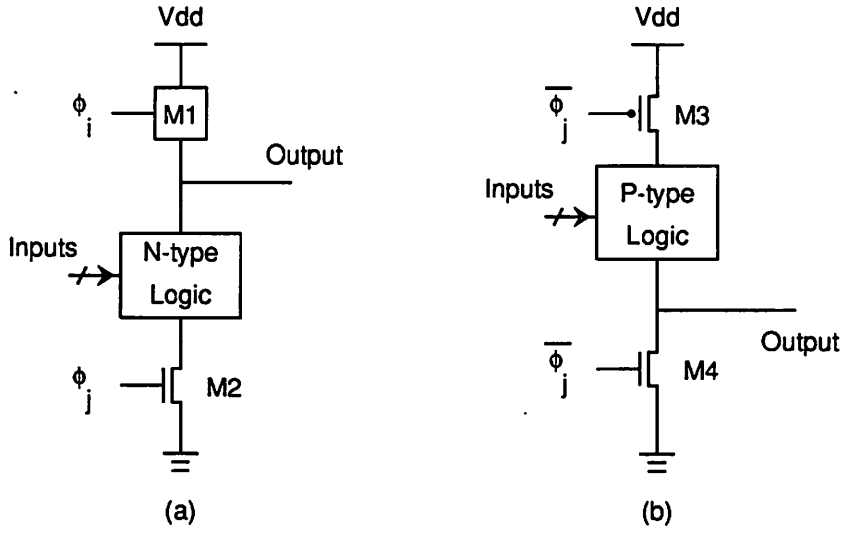


Figure 2.10 (a) N-Type Dynamic Logic Gate (b) P-Type Dynamic Logic Gate

logic gates are used as precharged busses in CMOS design or they are used to construct CMOS domino logic [31], where each N-type dynamic logic gate is followed by an inverter to ensure a single transition from logic "0" to logic "1" at the gates of NMOS transistors in the next N-type dynamic logic gates. Thus, the domino logic precharges the output nodes of N-type dynamic logic gates and evaluates multiple logic levels during the same clock phase, where the outputs fall sequentially, like a row of dominos. The domino logic consisting of the N-type of Figure 2.10(a) will precharge when ϕ_j is low and evaluates when ϕ_j is high. Since an N-type dynamic logic gate utilizes precharging, its timing constraints are similar to those for precharged modules illustrated in Figure 2.8. If *M1* is an NMOS transistor (NMOS design), a precharging time must satisfy the following:

$$T_{PRECHG} \leq t_{OFF}(M1) - t_{ON}(M1) = \phi_i(F) - \phi_i(R) \tag{2.14.a}$$

Similarly, if *M1* is a PMOS transistor (CMOS design), the following constraints are given for precharging an output node:

$$T_{PRECHG} \leq t_{OFF}(M1) - t_{ON}(M1) = \phi_i(R) - \phi_i(F) = \phi_j(R) - \phi_j(F) \tag{2.14.b}$$

The falling output transition of an N-type dynamic logic gate, which starts after $\phi_j(R)$, must finish by the next $\phi_j(F)$ at which *M2* turns off.

$$t_{output}(F) \leq \phi_j(F) \tag{2.15}$$

When falling input transitions of an N-type dynamic logic gate do not finish until *M1* turns off, *Output* may lose its charge by charge sharing or discharging through NMOS transistors of N-type logic block as *M1* turns off. Therefore, most circuit designers prefer the falling input transitions to finish by the time *M1* turns off as follows. If *M1* is an NMOS transistor (NMOS design),

$$t_{INPUTS}(F) \leq t_{OFF}(M1) = \phi_i(F) \tag{2.16.a}$$

Similarly, if *M1* is a PMOS transistor (CMOS design),

$$t_{INPUTS}(F) \leq t_{OFF}(M1) = \phi_i(R) = \phi_j(R) \tag{2.16.b}$$

Another circuit design technique which uses precharging and predischarging is a NORA logic [32], which consists of alternating N-type and P-type dynamic logic gates. *M1* of N-type dynamic logic gate is a PMOS transistor and hence ϕ_i is ϕ_j. *Output* of a P-type dynamic logic gate of Figure 2.10(b) is predischarged when ϕ_j is high and is evaluated when ϕ_j is low. In a NORA logic, no inverters follow N-type or P-type dynamic logic gates. Like a domino logic, the NORA logic precharge and predischarge the output nodes of N-type and P-type dynamic logic gates, respectively. Then, it evaluates multiple logic levels during the same clock phase. The NORA logic consisting of the N-type and P-type logic gates of Figure 2.10 precharges and predischarges when ϕ_j is low and, then, evaluates when ϕ_j is high. Notice that a P-type dynamic logic gate is the dual of an N-type dynamic logic gate. Hence, the predischarging time of P-type dynamic logic gate has following timing constraints to satisfy:

$$T_{PREDISCHG} \leq t_{OFF}(M4) - t_{ON}(M4) = \bar{\phi}_j(F) - \bar{\phi}_j(R) \tag{2.17}$$

The rising output transition of a P-type dynamic logic gate, which starts after $\phi_j(R)$, must finish by the next $\bar{\phi}_j(R)$ at which *M3* turns off.

$$t_{Output}(R) \leq \bar{\phi}_j(R) \tag{2.18}$$

Similarly to the falling input transitions of an N-type dynamic logic gate, the rising input transitions of a P-type dynamic logic gate must finish before *M4* turns off and *M3* turns on:

$$t_{INPUTS}(R) \leq t_{OFF}(M4) = t_{ON}(M3) = \bar{\phi}_j(F) \tag{2.19}$$

2.9. CLOCK SKEW

During clock distribution in a synchronous system, there is a variation in the arrival time of clock signals to different clocked storage elements due to different propagation delays. This arrival time variance is called *clock skew*. Clock skew changes the effective clock period and clock separations and, as a result, generally impacts system performance adversely. An extreme clock skew turns nonoverlapping clocks into overlapping clocks and may introduce two-sided timing constraints. Note that clock skew is signal-relative at a logic and not a function of the absolute delay of signals in the circuit. Since it is difficult to predict clock skew and clock timing exactly, the control of clock skew is a problem in large systems, including VLSI systems. Although the clock delay can be reduced by placing re-shaping circuits at intervals along a long line, the overall delay is still fairly long compared to the propagation delay through a single MOS gate.

One way of looking at the influence of clock skew is to transform the clock skew to appear in the clocked paths as an absolute delay from a given reference point (usually the output of the clock generator) [28]. Suppose a clocked storage element, CSE, has a clock skew of d_{SKEW}. The clock skew on the clock signal path to CSE can be eliminated by adding a delay d_{SKEW} to the logic path following CSE, while the same delay is subtracted from the other logic path which precedes CSE. This transformation is shown in Figure 2.11.

Another way of taking the influence of the clock skew into account is to use actual clock arrival times to clocked storage elements, when attempting to verify clocked paths against timing constraints. That is, clock skew affects timing constraints by changing the effective clock separation or clock period. This approach is used in *E–TV*, as will be described in Chapter 5.

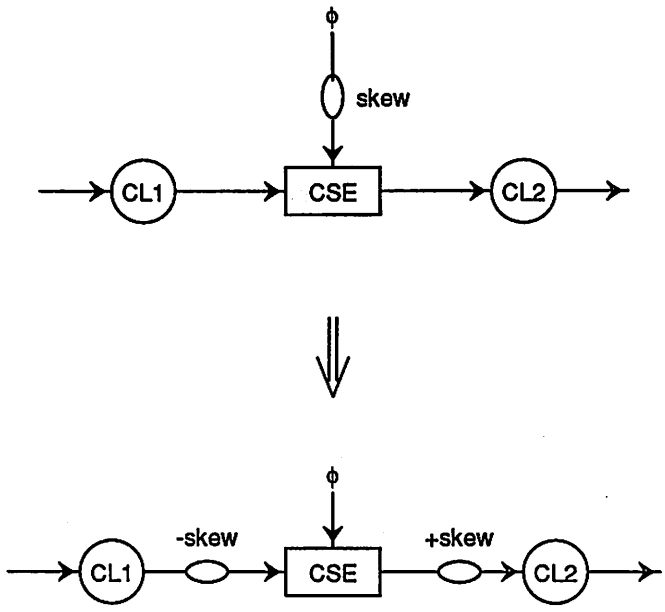


Figure 2.11 Transformation of Clock Skew into Clocked Paths

CHAPTER 3

TIMING VERIFICATION TECHNIQUES

During the design process of a digital system, designers examine the signal timings at various design points. There are two main purposes for the examination of signal timings. First, designers attempt to detect the possibility of timing errors. Needless to say, if there are any timing errors, the system will not function correctly. Therefore, detecting and correcting timing errors are crucial for the successful operation of the design. Second, designers attempt to optimize the performance of the system. In a system, the actual operating speed is determined by the slowest of all possible signal paths. The system speed can sometimes be improved greatly by reducing propagation delays through a few critically slow paths. Thus, designers are interested in identifying critical paths for the speed improvement of a system. At the same time, an unnecessarily fast section, compared to the rest of a system, usually consumes extra silicon area and power. Thus, designers are also interested in identifying the unnecessarily fast paths in the system to save silicon area and power.

To investigate system timings, designers would need to extract appropriate circuit pieces from the system. However, as the size and complexity of a system increase, it is no longer a simple task even for skilled designers to extract the right circuit pieces to analyze. Thus, since the Program Evaluation and Review Technique (PERT) [33, 34] had been applied to logic design [21], many timing verification tools have been developed in order to help designers detect timing errors and optimize system performance [35, 22, 36, 37, 38, 23, 24, 30]. PERT was developed as an aid in estimating end dates and critical paths for the scheduling of large projects.

While it may depend on a detailed implementation, a timing verifier usually needs three functional sections: the *path analyzer*, the *delay modeler*, and the *timing-constraint checker*. The operation of the three functional sections can be summarized as follows. A path analyzer extracts a logic block in a given system, systematically. A delay modeler evaluates a propagation delay through the logic block. Using this delay value, a path analyzer computes a path delay from starting nodes to the output node of the logic block. Then, a timing-constraint checker attempts to ascertain whether or not the path violates any timing constraints, or computes proper clock separations using the path delay. A timing verifier without a delay modeler requires the user to specify propagation delays through logic blocks in a system.

In this chapter, various aspects of timing verification techniques, including signal propagation, delay computations, and path analysis, are described. The path-analysis approach used in *E-TV* is the *critical-path* analysis method which detects only the critical paths in a given system. A number of critical-path analysis methods are described in detail, with emphasis on the switch level.

3.1. SIGNAL PROPAGATION FOR TIMING VERIFICATION

There are two possible approaches in propagating signals in a system for timing verification: *value-dependent* and *value-independent* propagations. Assuming that a signal at an input node of a logic block changes, the value-dependent approach propagates its effect to the output node of the logic block only when the other input conditions support it. Conventional simulators [6, 5, 9, 10, 39, 40, 41] belong to this approach. On the other hand, the value-independent approach always propagates the input change of the logic block to the output node without attempting to check whether the signal conditions at other input nodes support that propagation. Most timing verifiers [21, 35, 22, 37, 23, 24, 30] use value-independent

approach. Figure 3.1 compares two approaches. Suppose that all logic gates in Figure 3.1 have a unit delay for the simplicity of comparison. Also, suppose that the worst (longest) delay from input nodes *A* and *B* to an output node *E* is sought, when both inputs change at time t=0. As one of value-dependent approaches, the simulation results are shown in Table 3.1 for four sets of changing input signals. In the table, "Input States" represents whether an input rises or falls at time t=0. For examples, A(R) or A(F) represents that an input signal at

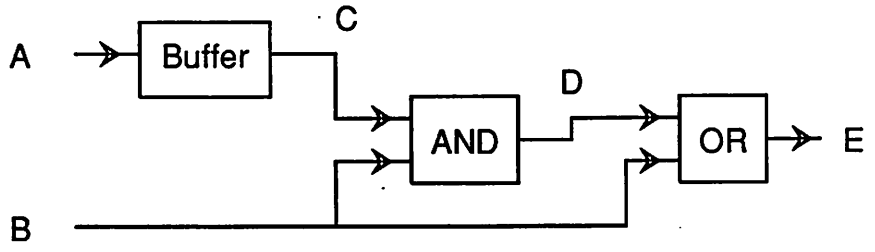


Figure 3.1 A Circuit Example Illustrating Pattern-Independent and Pattern-Dependent Signal Propagations

	Output (Rising)								Output (Falling)							
Input States	A(R), B(R)				A(F), B(R)				A(R), B(F)			A(F), B(F)				
Node	t<0	t>0			t<0	t>0			t<0	t>0		t<0	t>0			
A	0	1	1	1	1	0	0	0	0	1	1	1	0	0	0	
B	0	1	1	1	0	1	1	1	1	0	0	1	0	0	0	
C	0	0	1	1	1	1	0	0	0	0	1	1	1	0	0	
D	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	
E	0	0	1	1	0	0	1	1	1	1	0	1	1	1	0	

Table 3.1 Simulation Results of Figure 3.1

Node *A* rises or falls at time t=0. From the simulation results, the worst rising delay at *E* is 1 unit delay, when "Input States" is {A(R), B(R)} or {A(F), B(R)}. The corresponding worst delay path is found to be Path B-(OR gate)-E from Figure 3.1. Similarly, the worst falling delay at *E* is 2 unit delay, when "Input States" is {A(F), B(F)}. The worst delay path is Path B-(AND gate)-D-(OR gate)-E. On the other hand, when value-independent approach is used, both the rising and the falling worst delays are 3 unit delays through Path A-(Buffer)-C-(AND gate)-D-(OR gate)-E. Because these delays are the worst among all possible cases, they are called the *worst-case delays*. In this example, value-independent worst delays are pessimistic, compared to value-dependent worst delays computed by simulation, because their worst delay path is the one which can not be activated under real operating conditions. In Figure 3.1, in order for the value-independent worst delay path to be active, following two conditions must be satisfied at t>0.

(1) Node *B* must be at logic "1" in order for the signal change at Node *C* to propagate to Node *D* through an AND gate.

(2) Node *B* must be at logic "0", in order for the signal change at Node *D* to propagate to Node *E* through an OR gate.

Because above two conditions conflict with each other, this value-independent worst delay path, A-(Buffer)-C-(AND gate)-D-(OR gate)-E, can not be activated. The paths that are detected by value-independent approach but cannot be activated under real operating conditions are called *false paths*. When value-independent approach is employed, the user needs to perform a *case analysis* [22], which analyzes a number of different cases within a given circuit, one after another, to exclude false paths from consideration. The false paths can be identified by checking the consistency of the signal conditions that are necessary for paths to be activated either manually or automatically by a computer [42, 43]. Even though the value-independent approach may detect false paths, it has an advantage of significantly

reduced work in finding the critical paths in a system and test completeness, when compared to value-dependent approach. Note that value-dependent approach may not examine the critical paths, unless a particular set of input values is fed to the system; e.g., {A(F), B(F)} must be applied for the detection of the worst falling delay of 2 units at Node *E*, in this particular example. Therefore, exhaustive simulation is necessary and the amount of work to locate the critical path is exponential in the number of input nodes. Even though there are many algorithms that find efficient sets of input vectors which can cover most sensitizable paths in a circuit [44,45], the use of such input vectors do not guarantee that the value-dependent approach examines pathological paths. In addition, when simulation is used, it is very difficult to extract the critical paths in a circuit. As a result, the value-independent approach, which provides efficiency and test completeness, is favored for timing verification. In this dissertation, timing verifiers refer to the ones which use value-independent signal propagation.

3.2. PATH ANALYSIS FOR TIMING VERIFICATION

The path analyzers of existing timing verifiers fall into one of two categories in examining paths to locate the critical paths: *path enumeration* and *critical-path analysis* method.

The path enumeration method [35,36] extracts, examines and assesses all possible paths in circuits between preceding and succeeding space references. This method is also referred to as the *path oriented approach.* Because there is usually more than one path between each pair of preceding and succeeding points, the number of paths to examine grows exponentially with the number of circuit branches. Therefore, this method suffers from long computer running times for large circuits but is suitable for small combinational logic circuits. The advantage of having all paths examined is that it is easy to handle false paths. The

individual false path may be blocked before analysis or it can be simply ignored after analysis.

To describe the critical-path analysis method [21,22,23,24], it is convenient to define following terms.

[Definition 3.1] Ancestor, descendant, parent, child, source, and sink nodes in a directed acyclic graph

In a directed acyclic graph, if v and w are two nodes such that v is on the path from r to w, v is an *ancestor* of w and w is a *descendant* of v. If v and w are adjacent, v is the *parent* of w and w is a *child* of v. A *source* node is a node which has no ancestor. A *sink* node is a node which has no descendant. □

When Node A is the parent of Node B on the critical path, A is called a critical parent. Unlike path enumeration methods which examine all paths, the critical-path analysis method locates only the critical paths between preceding and succeeding space references. In contrast with the path oriented approach, this method has been referred to as *block oriented approach* [37], because it identifies the critical path leading up to each block (or each point). While this is a perfectly relevant name to be used at the block level, the critical-path analysis method is a more appropriate name, as the approach can also be used at other levels such as switch level. During critical-path analysis, each node stores information on its critical path only and information on other paths is lost during the search. Therefore, it is difficult for the method to locate the real critical path, when the detected one is a false path. Consider a directed signal graph of Figure 3.2, where edges represent logic blocks. Suppose Node A is the critical parent of Node B. Then, the critical-path analysis method takes A-B-C-F and A-B-C-D as the critical paths for Nodes F and D, respectively. Let Path A-B-C-D, which is the

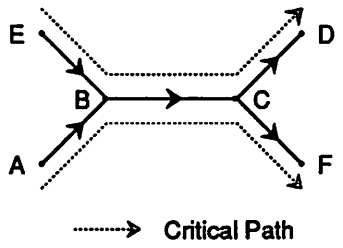


Figure 3.2 A Signal Graph Illustrating The Difficulty of Critical-Path Analysis in Blocking An Individual False Path

critical path for *D*, be a false path. Then, the next critical path (*E-B-C-D*), must be chosen as the real critical path for Node *D*. However, because Node *B* does not store information on the second critical parent (Node *E*), the critical-path analysis method has difficulty finding the right critical path for Node *D* (*A-B-C-D*). Thus, the method must do case analysis, mentioned in previous section, to keep from locating false critical paths. If path enumeration method is used, four paths, *A-B-C-D*, *A-B-C-F*, *E-B-C-D*, and *E-B-C-F*, will be examined. If Path *A-B-C-D* is found to be a false path, it may be ignored and Path *E-B-C-D* can be taken as the critical path for *D*. On the other hand, because the critical-path analysis method examines paths that contains the critical paths for parent nodes only, it runs much more quickly and hence is more appropriate for large circuits than the path-enumeration method.

3.3. DELAYS FOR TIMING VERIFICATION

3.3.1. DELAY COMPUTATION METHODS

Delays through groups of transistors or logic blocks (hereafter, logic blocks) in a system can be computed statically or dynamically for timing verification. The static delay

computation method [22, 46, 30] computes a propagation delay through each functional logic block before a verification starts. The propagation delays can be computed using the delay modeler of a timing verifier or other methods such as simulation tools. These delays are used when computing total path delays through functional logic blocks during path analysis. For a multi-input logic block, the worst-case delay among the delays between pairs of input and output nodes is commonly chosen as a delay for the logic block. Therefore, this method tends to overestimate path delays. Although some timing verifiers that use static delays allow a delay for each pair of input and output nodes, factors such as the information about the input waveform and load condition are not taken into account. On the other hand, the dynamic delay computation method [23, 24] computes a delay through an actual path extracted during path analysis. Timing verifiers using this method usually extract a logic block by tracing back along the path they traversed until a voltage source or ground is reached. Then, an external input signal is applied to the gate of a driver on the logic block to compute a path delay. Thus, two paths with the same schematic but different layouts are evaluated separately using their actual parasitics. Even though dynamic delay computation is more time consuming than static delay computation, it is preferred because of its superior accuracy.

3.3.2. DELAY REPRESENTATIONS

There are three methods for representing path delays or signal arrival times at nodes: *nominal value*, *min-max*, and *statistical representations*.

In the nominal value representation [23, 24], each node in a circuit has a single nominal arrival time for each signal. The nominal arrival time at the output of a logic block is the sum of the nominal input arrival time and a nominal delay through the logic block.

In the min-max representation [38], signal arrival times at nodes and signal delays through logic blocks have the upper and lower bounds. For this method to be useful for designers, the bounds must be tight enough.

In the statistical delay representation [21,37,35,47], a standard deviation and a variance are used to represent the delay distribution. As an example, in the Timing-Analysis program [37], the delay through each logic block and the signal arrival time at each node consist of a mean value (μ) and a delay standard deviation (σ). The mean arrival time at a node is the sum of a mean arrival time at the input of a logic block and a mean delay through the logic block. The arrival time standard deviations are computed by applying standard convolutions. The latest (or earliest) arriving signal path of a node is obtained by comparing the worst-case arrival time of the node through incoming paths, $\mu + \beta\,\sigma$ (or $\mu - \beta\,\sigma$), where β is a confidence level defined by the user. This statistical representation is useful in analyzing systems built using discrete components from different wafer lots, since the device characteristics from different wafers may vary widely. However, timing verifiers based on this representation are only as accurate as the user-defined parameters such as correlation coefficients and confidence levels. The selection of these parameters is both technology and system dependent, and is error-prone. In the case of one-chip VLSI MOS systems, device characteristics are usually very similar. Therefore, for such one-chip systems, a better approach is multiple analyses of the system, using different sets of transistor parameters, for example, one set obtained from typical process and another from the worst case process.

Some approach (e.g., [48]) allows the user to select one of the three representations for delays and time in the analysis, or to extend the program by developing new delay models to meet special needs. Independent of the delay representation, the concept of "slack" is used to provide a measure of the severity of the timing problem in some timing verifiers

[21, 37, 24, 30].

3.4. TIMING VERIFICATION LEVELS

Like simulation, timing verification can span several levels, such as behavioral, block, and switch levels. The timing verification level, here, refers to the level of timing primitives used for path analysis and constraint verification. The timing verification level may be different from the circuit primitive level used for delay computations, because timing verifiers may compute path delays from the lower level circuit description, especially when they have a hierarchical design data base [36, 38].

The example of the behavioral timing verifier is SLTV [49], which calls a general behavioral simulator *Helix* [50] to perform behavioral analyses. In SLTV, the primitives are described using one of hardware description languages. It has an advantage that existing primitives can be easily adapted to a particular technology and new primitives can be easily added.

The block-level timing verifiers [22, 37] provide models for functional logic blocks such as clocked storage elements (latches and flip-flops) and logic gates. When a block-level timing verifier verifies a synchronous system, it can obtain all necessary information on clocked storage elements such as set-up times from a circuit description. However, unlike bipolar designs, MOS transistors can be used as switches but these MOS transistors are difficult to model at the block level properly.

In the switch-level timing verification [24, 23], the system is described at MOS transistor level. The switch-level timing verifiers usually extract linear chains of transistors rather than functional logic blocks. Then, delays through the transistor chains are computed using a simple method, as will be described in Chapter 4. The switch-level timing verifiers have

some difficulties dealing with synchronous systems: first, a variety of design techniques such as precharged busses and domino logic impose different timing constraints, as described in Chapter 2. Second, the identification of clocked storage elements and the computation of their set-up times are difficult. However, switch-level timing verifiers model MOS transistors that are used as switches more accurately than block-level timing verifiers. Therefore, they are more appropriate for MOS VLSI designs.

3.5. CRITICAL-PATH ANALYSIS

Critical-path analyzers examine path delays leading to each node in a circuit, while pruning non-critical paths. Therefore, they need to visit nodes in a systematic manner. There are three approaches available for this purpose: *modified depth-first search* [23], *modified breadth-first search* [24], and *topological-order based search.* Two aspects that deserve a discussion are complexity and how each approach deals with loops in the circuit. In this section, the complexity of the three approaches and their methods of handling feedback loops are discussed.

3.5.1. COMPLEXITY OF CRITICAL-PATH ANALYSIS

The running time of any critical-path analyzer is $O(m)$, if it computes the critical path for each node only once, where m is the number of edges in a given directed graph. However, if a modified depth-first search is employed, it computes the critical path for some nodes more than once on the average. Consider a directed graph fragment shown in Figure 3.3, where an edge and an arrow represent a timing primitive and the direction a signal flows through it. There are two paths to Node *I* from Node *A*: *A-B-D-F-G-H-I* and *A-B-E-H-I*. Suppose the slowest path from *A* to *I* is sought. The work required by a modified depth-first

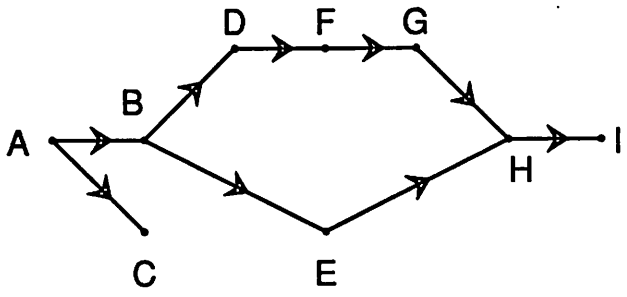


Figure 3.3 A Directed Graph Fragment

search, a modified breadth-first search and topological-order based search, will be compared next.

As its name implies, a modified depth-first search is a variation of depth-first search. Depth-first search [51, 52] starts at node v and marks it as having been visited. Next, it visits any unvisited nodes adjacent to v (*forward stepping*). If x is the most recently visited node, the search continues by visiting some unvisited node y adjacent to x. If y has been previously visited, the depth-first search finds another new node adjacent to x. If y has not been previously visited, it visits y and begins the search anew, starting at node y. After completing the search through all nodes adjacent to y, the search returns to x (*back tracing*). The process of selecting unvisited nodes adjacent to x is continued until the list of these nodes is exhausted. As an example, a depth-first search visits nodes in a graph illustrated in Figure 3.3 as follows:

$$A\text{–}B\text{–}D\text{–}F\text{–}G\text{–}H\text{–}I\text{–}(H)\text{–}(G)\text{–}(F)\text{–}(D)\text{–}(B)\text{–}E\text{–}(B)\text{–}(A)\text{–}C\text{–}(A) \tag{3.1}$$

where a node name in a parentheses represents a back tracing. When a critical-path analyzer based on a modified depth-first search examines Path $A\text{–}B\text{–}D\text{–}F\text{–}G\text{–}H$, the other path to H

through Node E has not been explored yet. Thus, the analyzer assigns Path $A-B-D-F-G-H$ as the slowest path leading to Node H. Similarly, when Node I is visited next, Path $A-B-D-F-G-H-I$ is assigned as its slowest path. After tracing back to B from I, the analyzer visits E, and then H again through a new path $A-B-E-H$. At this time, it compares the new path delay to the old critical path delay stored at Node H. If the new path delay is smaller than the old one, the analyzer traces back and visits Node C. However, if the new path delay is larger than the old critical path delay, the critical path to H is updated by the new path, $A-B-E-H$. Then, Node I, which is a child of H, is visited again to update the new critical path. In general, if the critical path of a node is first assigned or is updated by a new path, the critical-path analyzer visits its children to examine their critical paths. Two extreme cases can happen when a modified depth-first search visits already visited nodes for locating the critical path:

(1) *Best case:* a path analyzer traces back from already visited nodes without updating the critical path.

(2) *Worst case:* a path analyzer updates the critical path of an already visited node, whenever the one for any of its parent nodes is updated.

The running times for the best and worst cases are linear and exponential with a graph size respectively. The worst case corresponds to the path enumeration approach which examines all possible paths. An analyzer using a modified depth-first search on the average computes the critical path for each node more than once. On the other hand, the other two approaches (modified breadth-first and topological-order based search) compute the critical path only once for each node, when they are applied to a given acyclic digraph (if it is not acyclic, cycles must be broken in advance). The running time of these approaches to compute the longest path or shortest path in an acyclic digraph is $O(m)$, where m is the number of edges [53]; the complexity of two approaches for finding critical paths is linear, or approximately

linear if edge delays must be computed, with a circuit size.

Breadth-first search [51,52] starts at node v and marks it as having been visited. The node v is at this time said to be unexplored. A node will be said to have been explored when all nodes adjacent to it have been visited. All unvisited nodes adjacent to v are visited next. These are now visited but unexplored nodes and Node v has now been explored. The newly visited nodes which haven't been explored are placed on the end of a list of unexplored nodes. The first node on this list is the next node to be explored. Exploration continues until no unexplored nodes are left. The list of unexplored nodes operates as a queue. For a directed graph illustrated in Figure 3.3, a breadth-first search visits nodes as follows:

$$A\text{–}\{B, C\}\text{–}\{D, E\}\text{–}\{F, H\}\text{–}\{G, I\} \tag{3.2}$$

where nodes in a brace can be visited in any order. Notice that, from Equation (3.2), breadth-first search visits H before it visits G. When H is visited, no information is available on the path coming from Node G yet. Thus, when breadth-first search is used for finding the critical paths, the conditions to visit child nodes are modified; after node v is visited, its child w is visited only if all of w's parent nodes have been visited. This modified breadth-first search is called *level-based* search, because of the fact that it visits nodes from the lowest to the highest level, where the level of a node is defined as follows [54]:

[Definition 3.2] Level of a node

For a given acyclic digraph, the level of source nodes is defined as zero. For the other nodes, the level is defined as follows:

$$\textit{level of Node } j \;=\; 1 + \textit{MAX (level of parents of Node } j) \qquad \square$$

In other words, the level of Node j is the number of edges on the longest path from source nodes to Node j. Leveling can be performed by repeatedly deleting nodes with in-degree

zero and associated edges, starting from source nodes. All nodes with in-degree zero have the same level and they must be deleted at a time. The node level increases by one, each time all nodes with in-degree zero are deleted. Another method of leveling a graph, which breaks cycles at the same time, is described in the next section. The leveling of a graph illustrated in Figure 3.3 is as follows:

Level	0	1	2	3	4	5	6
	A	B C	D E	F	G	H	I

Table 3.1 Leveling of a Graph in Figure 3.3

Note that a level-based search visits *H* after *G*, while the original breadth-first search visits *H* before *G*. If nodes in a graph represent different tasks and edge directions represent precedence relations between the tasks, then the entire process can be completed without reprocessing a particular task by following the node level order. In Table 3.1, Nodes *D* and *E* have the same level. Note that the tasks corresponding to a given level can be executed in parallel.

A topological ordering of an acyclic digraph is a total ordering of its nodes such that ancestor nodes are ordered before descendant nodes [53]. While leveling is one type of topological ordering, the topological ordering in this dissertation will refer to the one which assigns different orders to different nodes. Similar to leveling, a topological ordering can be performed by repeatedly deleting a node with in-degree zero with a complexity of $O(m)$, where m is the number of edges [55]. In this case, each node with in-degree zero must be deleted individually but can be performed in any order. The node order increases by one as a node is deleted. An alternative method would be to carry out a depth-first search and order the nodes in decreasing order as they are postvisited [53]. A topological ordering of a graph

in Figure 3.3 using the latter method is shown in Equation (3.3) in increasing order, when nodes are visited as in Equation (3.1):

$$A\text{–}C\text{–}B\text{–}E\text{–}D\text{–}F\text{–}G\text{–}H\text{–}I \tag{3.3}$$

The topological order also indicates the order of work to be done without doing the same work again. When the critical-path analysis method is implemented on multi-processors, breadth-first search is a better choice than topological ordering, because it presents different tasks that can be done in parallel.

3.5.2. BREAKING FEEDBACK LOOPS IN CRITICAL-PATH ANALYSIS

Suppose that a circuit for timing verification contains signal feedback loops formed by timing primitives. Unless the circuit is redescribed using a new timing primitive which contains the feedback loop, all timing verifiers seeking the longest delay paths must break the feedback loop. Otherwise the longest delay path is not well defined, because traversing once around the loop will increase a path delay by the loop delay.

Among the three approaches available for critical-path analysis, a modified depth-first approach breaks feedback loops in more natural ways than level or topological-ordered based approach. A modified depth-first approach selects an edge to cut *dynamically* depending on the path taken around the loop during search, while the other two approaches select it *statically*. This can be illustrated using a digraph fragment containing a loop with multiple entry and exit points, which is shown in Figure 3.4. In the digraph, Path $B\text{–}C\text{–}E\text{–}G\text{–}B$ forms a loop. There are four paths passing through the loop without repeated nodes: $A\text{–}B\text{–}C\text{–}D$, $A\text{–}B\text{–}C\text{–}E\text{–}G\text{–}H$, $F\text{–}E\text{–}G\text{–}H$, and $F\text{–}E\text{–}G\text{–}B\text{–}C\text{–}D$.

Consider a critical-path analyzer using a modified depth-first search. When the analyzer enters the loop through Edge A-B during path analysis, it traverses Paths

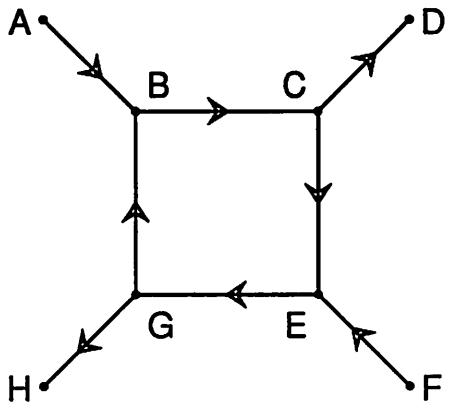


Figure 3.4 A Digraph Fragment Containing A Loop

A–B–C–D and *A–B–C–E–G–H* without cutting any edge. When the analyzer visits Node *B* again by traversing *A–B–C–E–G–B*, it traces back after cutting Edge *G–B* to break the loop. Next, when the analyzer enters the loop through Edge *F-E*, it traverses Paths *F–E–G–H* and *F–E–G–B–C–D*. When it visits Node *E* again by traversing *F–E–G–B–C–E*, it cuts Edge *C–E* and traces back. Therefore, a critical-path analyzer using a modified depth-first search cuts Edge *G–B* when it visits the loop through Edge *A–B*, or it cuts Edge *C–E* when it visits through Edge *F–E*. Because of the fact that the analyzer selects an edge to cut dynamically depending on the traversed path during path analysis, it can examine all paths passing through the loop.

Level-based search assumes that a given digraph is acyclic. Thus, a critical-path analyzer using this approach breaks loops made of timing primitives statically before path analysis, usually during leveling. An algorithm which breaks loops in a graph during leveling is shown in Algorithm 3.1 [54]. Because finding an optimal edge which blocks the

[1] Set all node levels = 0.

Mark all source nodes PROCESSED and put them on work stack.

[2] Is work stack empty?

If no, go to step [3].

Else, go to step [5].

[3] Remove a node, j, from work stack.

Update the level of child nodes, $k's$, of j, where

new level of node k = MAX { (old level), (1 + level of j) }

If all parent nodes of k are PROCESSED,

mark node k PROCESSED and put it on work stack.

[4] If all nodes have been PROCESSED, stop.

[5] Break a loop by removing an edge of the loop.

Find nodes, $p's$, whose all parent nodes are PROCESSED.

Mark $p's$ PROCESSED and put them on work stack.

Go to step [2].

Algorithm 3.1 Algorithm for Leveling and Loop-Breaking

minimum number of paths is very time consuming and impractical, a random edge is usually cut.

Like the level-based approach, topological ordering assumes that a given digraph is acyclic. Thus, a critical-path analyzer using topological ordering also must break loops in a circuit statically before path analysis. When depth-first search is utilized for topological ordering, the method can detect the last edge completing a loop. Even though such edges may not be optimal edges to cut, they are better choices than random edges in most cases; they will cut fewer forward paths. This can be seen using Figure 3.4. Suppose that a depth-first search visits the loop of B–C–E–G–B through Edge A–B during topological ordering. Then, Edge G–B will be cut to break the loop. In this case, only a path F–E–G–B–E–D is blocked. On the other hand, if the depth-first search visits the loop through Edge F–E, Edge C–E will be cut and a path A–B–C–E–G–H is blocked. Thus, in either case, only one path is blocked. Notice that the removal of other edges B–C or E–G blocks three paths. As an example, if Edge B–C is removed, Paths A–B–C–D, A–B–C–E–G–H, and F–E–G–B–C–D are blocked from path analysis.

Signals can flow through MOS transistors in both directions, from source to drain and from drain to source during operation. Thus, an MOS transistor itself forms a loop between source and drain terminals. In MOS circuits, however, most MOS transistors are intended to be *unidirectional*; their signals are expected to flow only in one direction during normal operation. Some switch-level critical-path analyzers utilize the signal flow direction of unidirectional transistors during path analysis [23,24]. Users are strongly recommended to specify signal flow directions of all unidirectional MOS transistors in a circuit. If they are not specified, a critical-path analyzer using a modified depth-first approach may examine many unnecessary paths. It will not only consume extra CPU time but also may detect unrealistic critical paths in the circuit. On other hand, if a critical-path analyzer uses level or topological-order based approach, it will determine a signal flow direction of an MOS transis-

tor arbitrarily to break a loop, if it is not specified by the user. Therefore, such loop-breakings may exclude important paths from examination by inadvertently choosing wrong directions. Note that, even for *bidirectional* MOS transistors where signals flow in both directions of source-drain channels, the users are recommended to specify the signal flow directions so that more important directions from the designers' viewpoint can be examined. Directions of signal flow through unidirectional MOS transistors can be specified by circuit designers during design, or can be determined later automatically by a computer program [42,43].

3.6. SWITCH-LEVEL CRITICAL-PATH ANALYSIS

A pass-transistor tree is shown in Figure 3.5, where signal flows are represented using arrows. In the figure, Node *p* has *m* input transistors and *n* output transistors. Suppose that a path delay from Node *(in_1)* to Node *(out_1)* is to be computed. In this situation, it is impor-

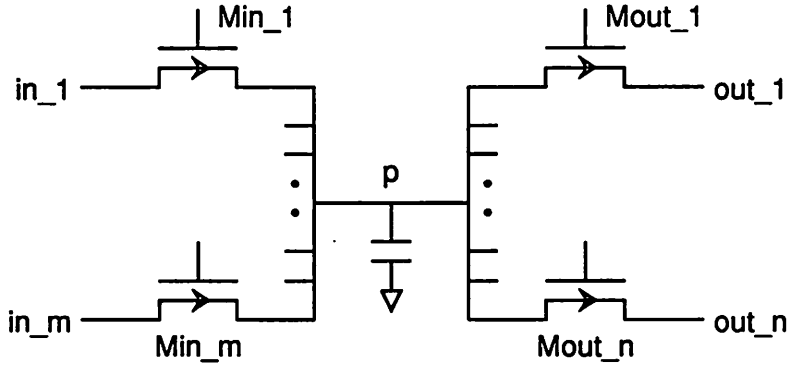


Figure 3.5 A Pass Transistor Tree

tant to know how many transistors are "on" during signal propagation through the path. The "on-transistors" affect the path delay, and hence must be considered for delay computations. Most switch-level critical-path analyzers perform value-independent analysis and they do not keep track of functional relationship among nodes. They do not have the necessary information for the determination of transistors that are on during signal propagation. Thus, they determine which transistors are on, based on assumptions as follows. When signals enter a node through more than one MOS transistor, the assumption would be that the transistors must be mutually exclusive in driving the node to avoid signal contention or too optimistic results, with the exception of "always-on" weak transistors such as NMOS depletion loads. In Figure 3.5, it is assumed that only one of m input transistors turns on at a time to drive Node p. Similarly, when a signal leaves a node through more than one MOS transistor, it is assumed that only one of the transistors, along the path for the delay computation, is on. Otherwise, the computed delay will be too pessimistic due to loading effect. In Figure 3.5, it is assumed that only one of n output transistors turns on at a time. To summarize, switch-level critical-path analyzers assume that only one MOS transistor turns on at a time between two nodes on a given signal path. They extract transistor chains rather than transistor trees to compute path delays. As an illustration, when switch-level critical-path analyzers compute a path delay from Node (*in_1*) to (*out_1*) in Figure 3.5, they extract a transistor chain of M_{in_1} and M_{out_1}. It should be noted that timing verifiers of other abstraction level use corresponding assumptions, as long as they also use the value-independent approach. For example, when a falling delay through a 2-input NOR gate is computed for block-level timing verifiers, it is usually assumed that only one input signal rises and the other remains at logic "0" (only one driver turns on and the other remains turned off), even though both input signals may rise at the same time under real operating conditions.

Another issue that deserves discussion is the selection of candidate paths by higher level as well as switch-level critical-path analyzers for the determination of critical path. Assume that, in Figure 3.5, the worst delay path from input nodes to the middle node *p* is (*in_j*)-*p*. Now, suppose that a switch-level critical-path analyzer locates the worst delay path from input nodes to an output node (*out_1*). If a critical-path analyzer uses a modified depth-first search, it determines a path it examines first, say Path (*in_1*)-*p*, as the worst delay path for Node *p*. Next, it determines an extended path (*in_1*)-*p*-(*out_1*) and determines the path as the worst delay path from input nodes to (*out_1*). Then, it computes a path delay from the next input node, *in_2*, to Node *p*. In general, suppose that Path (*in_i*)-*p* is a new path whose delay is most recently computed. If the new path delay is larger than an old worst delay stored at *p*, the worst delay path for *p* from input nodes is updated by the new path. And, the analyzer examines its extended path (*in_i*)-*p*-(*out_1*) to update the worst delay path for (*out_1*). However, if the new path delay of (*in_i*)-*p*-(*out_1*) is smaller than the old worst delay stored at Node *p*, the analyzer moves to the next input node. Thus, once the analyzer examines (*in_j*)-*p*, which is the worst delay path for Node *p*, no paths from the next input nodes to (*out_1*) will be examined. If a critical-path analyzer makes use of level or topological-order based path analysis, it first examines all paths from input nodes to *p* and finds that (*in_j*)-*p* is the worst delay path from input nodes to *p*. Then, the analyzer examines its extended path (*in_j*)-*p*-(*out_1*) only to find the worst delay path from input nodes to (*out_1*). One must notice that a critical-path analyzer, regardless of its approach to visit nodes, examines only a path extended from the worst delay path for Node *p*, when searching for the worst delay path for (*out_1*). In general, when a critical-path analyzer locates the worst delay path for a node, it examines a path only if it includes the worst delay path for one of its parent nodes. All other paths are pruned from the worst delay path search. This search space pruning greatly reduces the complexity of finding the worst delay path. However, even

though it is very rare, the worst delay path may not include the worst delay path for a parent node. For example, in Figure 3.5, the worst delay path for (*out_1*) may be *(in_k)-p-(out_1)*, while the worst delay path for its parent *p* is (*in_j*)-*p*. Similarly, it is possible that two output nodes (*out_q*) and (*out_r*) have different worst delay paths up to their parent node *p* in Figure 3.5; e.g., the worst delay path for (*out_q*) may be (*in_i*)-*p*-(*out_q*) while the one for (*out_r*) is (*in_k*)-*p*-(*out_r*). Unlike the path enumeration method that examines all paths, the critical-path analysis method prunes the search space for the worst delay paths. However, it must be stated that a chance that the real worst delay paths may be excluded from consideration is very low in practical circuits, typically less than 0.02% [24]. On the other hand, the critical-path analysis method provides substantial advantage in complexity over the path enumeration method, depending on the approach used to visit nodes. For a pass transistor chain illustrated in Figure 3.5, a path enumerator examines $O(m \times n)$ paths: all paths between each input-output node pair. A depth-first based critical-path analyzer examines from $O(m+n)$ to $O(m \times n)$ paths. However, a level or topological-order based critical-path analyzer examines only $O(m+n)$ paths: m paths from each input node to Node p and n paths from the worst delay parent of p to each output node. Partly for these reasons, *E-TV* uses the critical-path analysis method based on topological-ordering for the timing verification of digital systems.

CHAPTER 4

DELAY MODELING AND THE ELECTRICAL-LOGIC ALGORITHM

During switch-level timing verification, a delay modeler evaluates delays through transistor groups extracted by a path analyzer so that a timing-constraint checker can detect paths that possibly violate timing constraints. Thus, timing verification results are only as accurate as the delay model that a timing verifier uses. Any timing errors or critical paths reported by a timing verifier with an inaccurate delay model are not useful for the correction or improvement of a design. In this chapter, the advantages and disadvantages of the delay models used in existing switch-level timing verifiers [23, 56, 24, 57] are discussed. Then, a new form of circuit modeling and analysis technique, referred to as *Electrical-Logic (ELogic)* [58, 59, 60, 61], is presented. ELogic has been implemented using the Node Analysis method [62, 63]. In this chapter, the Node Analysis based implementation of the ELogic technique is presented in detail, including the discussion of its accuracy and stability properties. The implementation has been shown to provide a continuous trade off between efficiency and precision [61]. Finally, the use of ELogic for the delay model in timing verification is described. The model is referred to as the *ELogic delay model* in this dissertation. As one of motivations of developing an ELogic-based timing verifier, some experimental results illustrating that the ELogic delay model has much greater accuracy than the existing switch-level delay model are also included.

4.1. EXISTING SWITCH-LEVEL DELAY MODELS

The existing switch-level timing verifiers such as *Crystal* [23], *LEADOUT* [56], *TV* [24] and *Pearl* [57] extract linear chains of MOS transistors and use the *RC delay model* to compute delays through the transistor chains; nonlinear MOS transistors are replaced by linear resistors in series with a switch, and a variation of the RC time constant is used as a delay. Consider a linear RC chain of Figure 4.1. Assume that Node *in* is a driving node with a strong signal source such as a voltage source. The simplest RC delay model is the *lumped RC delay model* which lumps all resistances and capacitances together to compute a time constant. Therefore, the delay from the driving node *in* to Node *n* is given as follows:

$$delay = \sum_{i=1}^{n} Ri \sum_{i=1}^{n} Ci$$

$$= (R1 + R2 + \cdots + Rn)(C1 + C2 + \cdots + Cn)$$

This model is overly pessimistic, because of the fact that each capacitance is multiplied by total resistance of the chain, including resistors that are not on the charging or discharging path. The error due to lumping resistances and capacitances can be avoided by using the

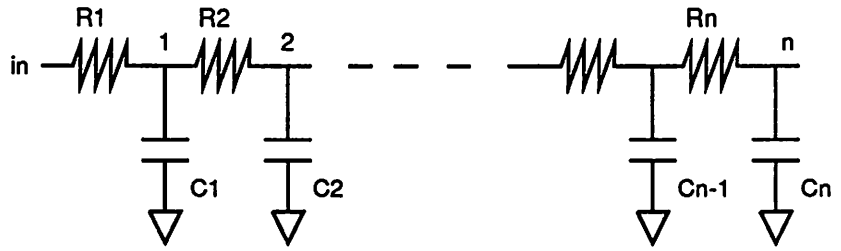


Figure 4.1 A Linear RC Chain

results of Penfield-Rubenstein model [64]. While the Penfield-Rubenstein model computes upper and lower bounds for signal delay in RC tree networks, their average is used as a "typical" delay in Crystal [23] and TV [24]. Using this method, the delay value at Node n through the RC chain is given as follows:

$$delay = \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{i} Rj \right) Ci \right)$$

$$= R1\,C1 + (R1 + R2)\,C2 + \cdots + (R1 + R2 + \cdots + Rn)\,Cn$$

The error of RC delay models also comes from their poor incorporation of input waveform shape and load conditions into the delay computation. Therefore, some RC delay models use a *ratio approach* [65] for improved accuracy, which combines factors such as the input rise time, the output load, and transistor size into a single ratio, called *rise-time ratio*. This ratio is then used to determine the "effective resistance" of a transistor, that provides better delay estimates.

The RC delay models are very efficient in delay computation and their accuracy has been improved by using the results of Penfield-Rubenstein and the ratio approach. However, they still suffer from poor accuracy. It is claimed that, even though the RC delay models have a poor absolute accuracy, the relative accuracy is good enough for ordering the worst delay paths and, therefore, they can be used for timing verification successfully [24]. However, it should be pointed out that the tasks of a timing verifier are not only ordering the worst delay paths but also detecting problematic paths that may violate timing constraints. Thus, the absolute accuracy as well as the relative accuracy is important. If the delay model with a poor absolute accuracy is used, the timing verifier can not determine correctly if there are any timing errors or not. In addition, the clock period of synchronous designs recommended by such a timing verifier is not meaningful. Furthermore, due to poor relative accuracy, the RC

delay model has a high chance of choosing the wrong worst delay path, as will be revealed in a CMOS example soon. Designers usually evaluate the worst delay paths reported by a timing verifier using more exact analysis programs such as SPICE2 [5]. However, the evaluation of the wrong worst delay paths is not helpful for the correction of timing errors. Another weakness of the RC delay models is that they assume there is only one direct path from a reference node (power supply or ground) to the signal-nodes of the circuit. When more than one reference node drives a node at the same time, the RC delay models cannot compute a delay to the node correctly. As an illustration, consider an NMOS inverter. When the pull-down delay of the inverter is computed, only the driver of the inverter is considered. Even though the depletion load in the NMOS inverter is on during the pull-down, it is not taken into account for a pull-down delay computation.

Timing verifiers examine a large number of paths in a system. Therefore, the delay model for timing verification must be efficient. On the other hand, the accuracy of timing verifiers is determined by the accuracy of the delay model they make use of. Thus, the delay model must be accurate too. Even though RC delay models are efficient, they have a poor accuracy as described earlier. What is required is a delay model which is not only fast but also accurate enough to be used for timing verification. As one of such delay models, the *ELogic delay model* is presented in the next section.

4.2. THE ELECTRICAL-LOGIC MODELING TECHNIQUE

In a conventional circuit modeling and simulation approach, a timestep is selected and the voltages (and currents) corresponding to the new time are computed. A key idea of *Electrical-Logic (ELogic)* is to model a technology in terms of a set of discrete, possibly non-uniformly spaced states of network variables (such as voltages and currents) and to solve

for the time required to make a transition between adjacent states. The idea of using additional states (beyond 0 and 1) to provide a more precise model of the behavior of a digital circuit, in both the time-domain and for static (clocked, settled) analysis, has evolved over the past decade (e.g. [7,22,66,8]). In addition, the notion of providing a less-precise circuit simulation to improve the performance of circuit simulators for digital MOS circuits was introduced in [67] and has been pursued since by many researchers (e.g. [68,69,70,71]). A generalization of logic simulation to the electrical domain, both in signal value and strength, was introduced in [72] and has been implemented and extended in a number of systems (e.g. [73,74,75,76]).

ELogic is a generalization of many of these earlier efforts, from both the logic and the electrical points of view, and is defined as follows:

[Definition 4.1] Electrical-logic modeling technique

1. A set of discrete states of network variables $\{S^i \mid i=1,...,N_s\}$ is defined. The network variables can be charges, fluxes, voltages, and currents.

2. Network variables are only allowed to make a transition from one state to an adjacent state:

$$S^i \rightarrow S^j, \quad j = i \pm 1$$

3. The *first time* t_{n+1} at which the *next state* of the variables (S^{i+1} or S^{i-1}) is achieved is computed, rather than solving for network variables x at a given time point:

$$t_{n+1} = \min \{ t \mid x(t) = S^j, j = i \pm 1, t > t_n, \textit{ where } x(t_n) = S^i \}$$

4. The number and values of the states of the variables can be varied between analyses and between portions of a circuit in the same analysis.

When the behavior of a network variable x is described in the form of $x = f(t)$, there may be

more than one t_j that satisfies $f(t_j) = S^{NEXT}$, where S^{NEXT} is the next discrete state. The ELogic technique computes the first one among the possibly many timepoints after the current timepoint, as has been defined in (3).

4.3. NODAL ANALYSIS BASED ELOGIC ALGORITHM

The ELogic modeling technique can be implemented using any circuit analysis method [62, 63]: Nodal Analysis, Mesh Analysis, Loop Analysis, Cut-set Analysis, and Hybrid Analysis. The analysis method determines network variables and circuit elements for analysis.

In this dissertation, an ELogic algorithm which implements the Nodal Analysis method is presented. Unless otherwise specified, ELogic denotes the Nodal Analysis based ELogic algorithm in this dissertation. ELogic supports following circuit elements: linear resistors and capacitors, voltage-controlled resistive elements, voltage-controlled capacitors, voltage-controlled current sources, and independent current sources. In addition, ELogic supports independent voltage sources, as is described soon. Since ELogic is based on Nodal Analysis, node voltages are chosen as network variables and ELogic solves for the amount of time required for a node to make a transition between adjacent discrete node-voltage states. An example of the ELogic output voltage waveforms at two neighboring nodes, j and k, are illustrated in Figure 4.2. In the figure, the voltages *V0* through *V5* represent the discrete set of voltage states, $\{V^i \mid i=0,...,5\}$. As an example, the node voltage at j is *V1* at time *t1*. At *t1*, ELogic finds the next voltage state for Node j, which is *V2* in this example. Then, ELogic computes the time for Node j to achieve *V2*. If a node voltage is computed to lie between two adjacent states at some steady-state value, its value must be rounded off to the nearest voltage state. Therefore the *precision* of an ELogic waveform is determined by the number

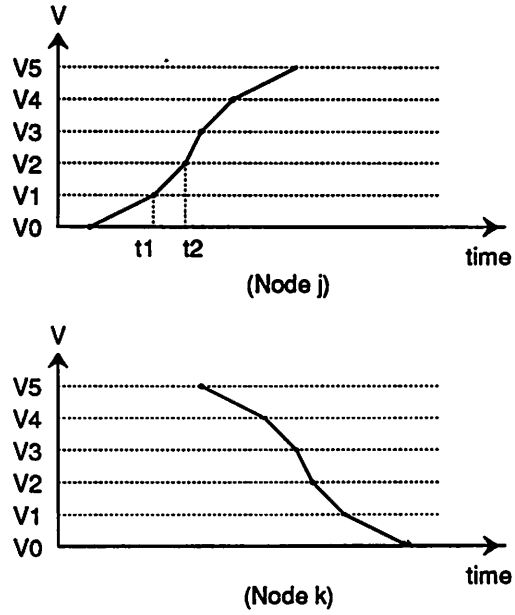


Figure 4.2 An Example of ELogic Output Waveforms

and values of the voltage states defined. As more voltage states are used, the *waveform fragments* (set of voltage state changes and corresponding times) that represent activity at a node will contain more time points and, as a result, solving for the waveform will take longer. Hence ELogic provides a continuous precision-efficiency trade off between the circuit and logic/switch-level simulation. The experimental results on its trade-off between precision and efficiency were presented in [61].

ELogic is used as a basis of the "Fast Timing simulation mode" in MOTIS3 [71] and Lsim's circuit simulator, Smile [77]. The idea of solving for a time rather than solving for node voltages is also used in the program SPECS [75], developed independently, to obtain reasonable timing results by appropriately handling the unknown state (X) during transitions.

In SPECS, transistor subnetworks are described utilizing capacitive nodes and macro-switches, such as pull-up, pull-down, and pass transistors. Then, the macro-switch is modeled using a current source to find the nearest time point at which any of the subnetwork nodes reaches its next voltage. This method is efficient but suffers from accuracy problems. On the other hand, the ELogic algorithm analyzes a circuit using more detailed transistor-level models, so its accuracy is comparable to that of circuit-level analysis in the extreme case.

4.3.1. TRANSITION TIME COMPUTATION OF THE ELOGIC ALGORITHM

ELogic assumes that each node, except voltage source nodes and ground, has grounded capacitance to guarantee a finite slope of the voltage waveform. There are a wide variety of discretization techniques in time for the purpose of solving a system of differential equations numerically. There are three issues to consider for the selection of a discretization method: accuracy, stability and efficiency. ELogic uses the explicit Euler (Forward Euler) integration method [78, 79, 63] for discretization, which is a convergent first-order integration method. While higher order methods provide more accurate solutions, they are computationally more expensive than a first-order method. ELogic shares the notion used in timing simulators which sacrifice accuracy in favor of efficiency [67, 68, 69, 70, 71]. It is known that an implicit integration method has better stability property than the explicit integration method. However, when the implicit integration method is used to solve for node voltage at j, it needs other node voltages at the same timepoint. When applied to very large systems that contain nonlinear MOS transistors, an implicit method requires a large matrix solution at each timepoint, hence it is more time consuming. By making use of the explicit integration method, ELogic can compute the timepoints of nodes independently of one another, and as a

result can exploit the multirate behavior of waveforms [80] efficiently. While the region of stability of the explicit Euler integration method is not as large as that of the implicit methods for a test equation of $\dot{y} = -\lambda y$, it is possible to keep the ELogic algorithm inside the stable region for a test circuit. This is described shortly. There is also another class of integration methods, called *semi-implicit* methods [81, 82, 83]. As the name implies, the semi-implicit method is a mixture of explicit and implicit integration methods. It is formulated to be as implicit as possible to obtain better stability properties than using explicit methods, without making it necessary to perform a standard matrix solution at each timepoint. The semi-implicit methods include one iteration of the Jacobi-relaxation method or the Seidel-relaxation method at the nonlinear equation level after applying the implicit Euler (Backward Euler) method to the given nonlinear differential equations. Thus, they are referred to as the *Jacobi semi-implicit* and *Seidel semi-implicit* method, respectively [83]. These semi-implicit integration methods have been used widely for timing analysis, because of their efficiency and better stability than the explicit integration methods for a certain class of circuits (e.g., if the matrix A is diagonally dominant with negative diagonal entries, when a system of equations is formulated in the form of $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t)$ [67, 68, 84]). However, a drawback with the use of these methods is that tightly coupled feedback loops, or bidirectional circuit elements, can cause severe inaccuracies. One element that can cause such a problem is the *floating capacitor* [82]. A floating capacitor is a capacitor whose terminals are connected neither to a fixed voltage source nor to ground node. To describe the way semi-implicit methods deal with floating capacitors, the following definitions are necessary.

[Definition 4.2] Fanin variables (inputs) and fanout variables

Suppose that the behavior of a circuit is described in following system of nonlinear differential equations:

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}) = 0$$

where x is a variable vector and u is an input vector to the circuit. A *fanin variable (input)* of x_j is defined as any variable x_k (any input u_k), $k \neq j$, which, if its value changes, may result directly (not involving other variables) in a change of the value of the variable x_j. A *fanout variable* of x_j is a variable x_l, $l \neq j$, whose value can be influenced directly by a change in the value of $\dot{x}_j$. □

When the Nodal Analysis method analyzes a circuit, the variables computed are node-to-datum voltages. Thus, the fanin and fanout variables are the voltage at the fanin and fanout nodes that are defined as follows:

[Definition 4.3] Fanin and fanout nodes

A *fanin* node FI_i of node i is defined as any node which, if its state changes, may result directly (not involving other nodes) in a change in state at node i. A *fanout* node FO_i of Node i is a node whose voltage state can be influenced directly by a voltage change at i. □

When Jacobi and Seidel semi-implicit methods are used to solve for Node j at t_{n+1}, they apply an implicit Euler integration method to model the grounded capacitance at j. However, because they perform only one iteration at the nonlinear equation level, it is equivalent to applying a fixed voltage source at FI_j; Jacobi semi-implicit method applies a fixed voltage source whose value equals the node voltage at FI_j at t_n and Seidel semi-implicit method applies a fixed voltage source whose value equals the node voltage at FI_j at t_{n+1} (if it has been already computed) or t_n (otherwise). Therefore, unlike the explicit Euler integration method, these methods turn all floating capacitors into grounded capacitors equivalently.

The equations for the computation of transition times are derived as follows. By applying Kirchoff Current Law (KCL) [62] at each node, a system of equations can be formulated in following form:

$$\mathbf{F}(\mathbf{v},\dot{\mathbf{v}},\mathbf{u}) = 0 \tag{4.1}$$

where **v** is a vector of node-to-datum voltages and **u** is an input vector (current sources) to the circuit. By moving all terms that are not associated with capacitive elements to the right handside, Equation (4.1) yields

$$\dot{\mathbf{Q}}(\mathbf{v}) = -\mathbf{f}(\mathbf{v},\mathbf{u}) \tag{4.2}$$

where **Q(v)** is a matrix of node charges and **f(v,u)** is a vector of current leaving nodes through all elements other than capacitive ones. Therefore, Equation (4.2) can be written as follows:

$$\mathbf{C}(\mathbf{v})\dot{\mathbf{v}} = -\mathbf{f}(\mathbf{v},\mathbf{u}), \text{ where } \mathbf{C}(\mathbf{v}) = \frac{\partial \mathbf{Q}(\mathbf{v})}{\partial \mathbf{v}} \tag{4.3}$$

Since each node has a grounded capacitor, the capacitance matrix **C(v)** is strictly diagonally dominant. i.e.,

$$|C_{ii}| > \sum_{j=1, j\neq i}^{n} |C_{ij}|, \text{ for } all\ i \tag{4.4}$$

From Equation (4.4), the Gershgorin Circle theorem [85] indicates that zero is not an eigenvalue of **C(v)**. Hence **C(v)** is nonsingular and $\mathbf{C}^{-1}(\mathbf{v})$ exists. By moving **C(v)** of Equation (4.3) to right handside,

$$\dot{\mathbf{v}} = -\mathbf{C}(\mathbf{v})^{-1}\,\mathbf{f}(\mathbf{v},\mathbf{u}) \tag{4.5}$$

The application of the explicit Euler method to above equation yields

$$\mathbf{v}_{n+1} - \mathbf{v}_n = -h\ \mathbf{C}^{-1}(\mathbf{v}_n)\,\mathbf{f}(\mathbf{v}_n,\mathbf{u}_n) \tag{4.6}$$

where $\mathbf{v}_n$ is a node-to-datum voltage vector at time t_n and h is a timestep. Equation (4.6)

can be decoupled for each Node j as follows (in this section, Node j represents the jth node when circuit equations are written in the form of (4.5)):

$$v_{(j),n+1} - v_{(j),n} = -h\ \mathbf{C_j^{-1}}(\mathbf{v_n})\ \mathbf{f}(\mathbf{v_n}, \mathbf{u_n}) \tag{4.7}$$

where $v_{(j),n}$ is a voltage at Node j at t_n and $\mathbf{C_j^{-1}}(\mathbf{v_n})$ is the jth row of $\mathbf{C^{-1}}(\mathbf{v_n})$. The transition time for a node to make a given voltage change is discussed separately for circuits without and with floating capacitors.

When there are no floating capacitors in the circuit, a capacitance matrix $\mathbf{C}(\mathbf{v_n})$ and its inverse, $\mathbf{C^{-1}}(\mathbf{v_n})$, are diagonal matrices. Therefore, Equation (4.7) turns out to be:

$$v_{(j),n+1} - v_{(j),n} = -h\ C_{jj}^{-1}(\mathbf{v_n})\ \mathbf{f_j}(\mathbf{v_n}, \mathbf{u_n}) \tag{4.8}$$

where

$v_{(j),n}$: voltage at Node j at t_n.

$C_{jj}(\mathbf{V_n})$: grounded capacitance at Node j at t_n

$f_j(\mathbf{v_n}, \mathbf{u_n})$: current leaving Node j at t_n through all elements other than capacitive ones

Let $v_{(j),n+1}$ and $v_{(j),n}$ in above equation be the next and present voltage states, S_{Next} and S_{Now}. ELogic computes a time for Node j to make a transition from S_{Now} to S_{Next}, as follows:

$$\textit{Transition time}, h = \frac{C_{jj}(\mathbf{V_n})\ (\mathbf{S_{Next}} - \mathbf{S_{Now}})}{-f_j(\mathbf{v_n}, \mathbf{u_n})} \tag{4.9}$$

Suppose that ELogic computes the transition time of Node j in Figure 4.3. From the figure, the denominator of Equation (4.9) is given as follows:

$$-f_j(\mathbf{v_n}, \mathbf{u_n}) = \sum_{i=1}^{N} I_{ij}(t_n)$$

where N is the number of j's fanin nodes from which there is a DC path to j, and $I_{ij}(t_n)$ is a

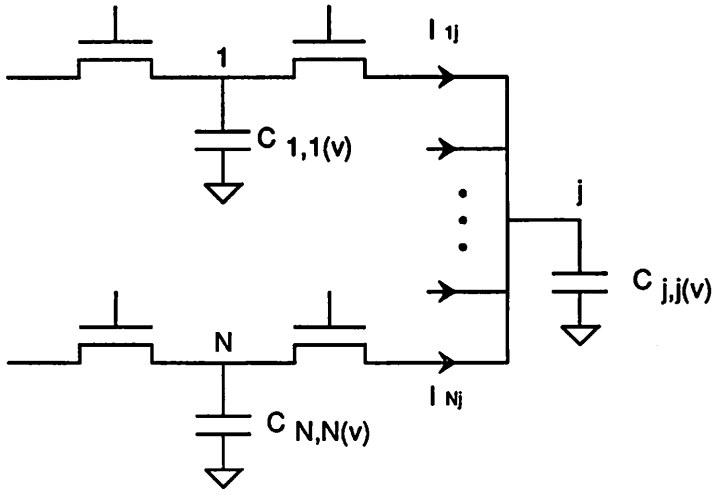


Figure 4.3 A Circuit Fragment without Floating Capacitors

current entering Node j from i at t_n. Notice that ELogic can substitute a grounded capacitor at the fanin node i of j by a constant voltage source whose value is equal to $v_i(t_n)$, because it applies the explicit Euler integration method to a circuit without floating capacitors.

Next, consider a circuit which has floating capacitors. From Equation (4.7), when there are floating capacitors in the circuit, the time h for Node j to make a transition from S_{Now} to S_{Next}, is given as follows:

$$Transition\ time, h = \frac{S_{Next} - S_{Now}}{-\mathbf{C_j^{-1}}(\mathbf{v_n})\ \mathbf{f}(\mathbf{v_n}, \mathbf{u_n})} \tag{4.10}$$

In above equation, $\mathbf{C_j^{-1}}(\mathbf{v_n})$ is the jth row of $\mathbf{C^{-1}}(\mathbf{v_n})$ which is no longer a diagonal matrix. It is well known that the complexity to obtain the inverse of a matrix is $O(n^3)$, where n is the dimension of the matrix [63]. Therefore, it is important for ELogic to keep the dimension of

matrix for inverse computation as small as possible. Note that, if the user includes floating capacitors that are intentionally added or functionally important only, $\mathbf{C}(\mathbf{V_n})$ will be sparse. In fact, many designers have approximated the gate capacitance of MOS transistor by attaching the same amount of capacitance from a gate to ground when analyzing a large MOS design. i.e., they have modeled the gate capacitance as grounded capacitance rather than floating capacitance). Therefore, ELogic computes $\mathbf{C}^{-1}(\mathbf{v_n})$ by exploiting the fact that the inverse of a block diagonal matrix is a collection of the inverse of individual submatrix on the diagonal, as follows:

$$\begin{bmatrix} a & 0 & 0 & 0 & 0 \\ 0 & \cdot & 0 & 0 & 0 \\ 0 & 0 & \cdot & 0 & 0 \\ 0 & 0 & 0 & \cdot & 0 \\ 0 & 0 & 0 & 0 & n \end{bmatrix}^{-1} = \begin{bmatrix} a^{-1} & 0 & 0 & 0 & 0 \\ 0 & \cdot & 0 & 0 & 0 \\ 0 & 0 & \cdot & 0 & 0 \\ 0 & 0 & 0 & \cdot & 0 \\ 0 & 0 & 0 & 0 & n^{-1} \end{bmatrix}$$

Because each submatrix is much smaller than $\mathbf{C}(\mathbf{V_n})$ in dimension, $\mathbf{C}^{-1}(\mathbf{v_n})$ can be computed very efficiently. Furthermore, Equation (4.10) is only interested in the jth row of $\mathbf{C}^{-1}(\mathbf{v_n})$. Thus, ELogic computes only the inverse of one submatrix which contains Node j, not the other submatrices. Note that the nonzero elements of $\mathbf{C}(\mathbf{v_n})$ represent the connectivity between nodes through floating capacitors: if $C_{i,j}$ is nonzero, $C_{j,i}$ is nonzero, and Nodes i and j are connected through floating capacitance. Therefore, ELogic constructs a submatrix that contains Node j for inverse computation by tracing nodes that are reachable from Node j through floating capacitors.

When Node j has no floating capacitors, the associated submatrix is a one-element matrix with $C_{j,j}(\mathbf{V_n})$, a grounded capacitance at j. Thus, $\mathbf{C_j}^{-1}(\mathbf{v_n})$ can be reconstructed as follows:

$$\mathbf{C_j}^{-1}(\mathbf{v_n}) = [0\,0\cdots 0\,0\,C_{j,j}^{-1}(\mathbf{V_n})\,0\,0\cdots 0\,0] \tag{4.11}$$

The substitution of Equation (4.11) into Equation (4.10) yields Equation (4.9). Thus, for those nodes without floating capacitors, ELogic uses Equation (4.9) whether or not the circuit contains floating capacitors.

When Node j has floating capacitors connected to its fanin nodes, the fanin nodes can not be replaced by constant voltage sources. The substitutions will turn floating capacitors into grounded capacitors equivalently, as described with semi-implicit integration methods. Let a set of nodes, $\{p, \cdots, j, \cdots, q\}$, are reachable each other through floating capacitors, and the corresponding diagonal submatrix of $\mathbf{C}(\mathbf{v_n})$ be $\mathbf{E}$. $\mathbf{E}^{-1}$ is computed by applying the Gaussian Elimination to an augmented matrix, $[\mathbf{E} \mid \mathbf{I}]$, as follows [63] :

$$[\mathbf{E} \mid \mathbf{I}] \xrightarrow{\text{Gaussian Elimination}} [\mathbf{I} \mid \mathbf{E}^{-1}]$$

Let $[e_{jp} \cdots e_{jj} \cdots e_{jq}]$ be a row vector of $\mathbf{E}^{-1}$, which corresponds to Node j, where the subscript of each element represents a node name. Then, $\mathbf{C_j}^{-1}$ is constructed as follows:

$$\mathbf{C_j}^{-1}(\mathbf{v_n}) = [0 \cdots 0\, e_{jp} \cdots e_{jj} \cdots e_{jq}\, 0 \cdots 0] \tag{4.12}$$

Notice that the above construction needs only one row of $\mathbf{E}^{-1}$. Thus, ELogic places Node j at the right-bottom corner of $\mathbf{E}$ so that the last row of $\mathbf{E}^{-1}$ can become the necessary row to construct $\mathbf{C_j}^{-1}$. By doing so, the corresponding row is available as the forward elimination of the Gaussian Elimination is completed, and hence the back substitution will not be needed. From Equations (4.10) and (4.12), the voltage change rate at Node j at t_n is given as follows:

$$\begin{aligned} \textit{voltage change rate at Node } j \textit{ at } t_n &= -\mathbf{C_j}^{-1}(\mathbf{v_n})\ \mathbf{f}(\mathbf{v_n}, \mathbf{u_n}) \\ &= \sum_{k=p}^{j} [e_{jk}\ (-f_k(\mathbf{v_n}, \mathbf{u_n}))] = \sum_{k=p}^{j} [e_{jk}\ I_k(t_n)] \end{aligned}$$

where $I_k(t_n)$ is the sum of the currents entering Node k at t_n through all elements other than

the capacitive ones. Therefore, the computational work to process Node j with floating capacitors increases N times, where N is the number of nodes reachable from j through floating capacitors, when compared to the work for a node without floating capacitors.

If nonlinear capacitor models are available, the user can describe a circuit more accurately. However, if all capacitors in a circuit are linear, $\mathbf{C}(\mathbf{v})$ remains the same at all timepoints; $\mathbf{C}^{-1}$ needs be computed only once before an analysis starts rather than at each timepoint. Thus, the analysis will require much less CPU time by using linear capacitors only. A timing verifier is expected to examine numerous paths in a system. While ELogic is able to handle nonlinear capacitors, only linear capacitors are allowed by the ELogic-based timing verifier (*E-TV*) for reasons of efficiency.

Since the ELogic algorithm is based on the Nodal Analysis method, it supports only current sources as inputs to a circuit ($\mathbf{u}$ in Equation (4.1)). Thus, independent voltage sources with a resistive element in series are transformed to independent current sources with the resistive element in parallel (*source transformation*). If a capacitor connects Node j and a varying voltage source, the voltage source affects the current that leaves j. On the other hand, if a capacitor connects Node j and a fixed voltage source, the capacitor is a grounded capacitor at j, not a floating capacitor. Therefore, a system of equations can be formulated from a circuit which contains grounded independent voltage sources as follows:

$$\begin{aligned} \mathbf{C}(\mathbf{V})\,\dot{\mathbf{v}} &= -\,\mathbf{f}(\mathbf{V}, \dot{\mathbf{v}}_{\mathbf{in}}) \\ &= -\mathbf{G}(\mathbf{V})\mathbf{v} + \tilde{\mathbf{G}}(\mathbf{V})\mathbf{v}_{\mathbf{in}} + \tilde{\mathbf{C}}(\mathbf{V})\dot{\mathbf{v}}_{\mathbf{in}} \end{aligned} \tag{4.13}$$

where

$\mathbf{v} \in R^n$: node-to-datum voltages excluding voltage sources.

$\mathbf{v}_{\mathbf{in}} \in R^p$: voltage sources.

$\mathbf{V} \in R^{n+p}$: node voltages consisting of v and $\mathbf{v_{in}}$.

$\mathbf{C}(\mathbf{V}) \in R^{n \times n}$: node-to-datum capacitance matrix, except voltage source nodes.

C_{ii}: sum of capacitances connected to Node i

C_{ij}: - (capacitance connected from Node j to Node i)

$\tilde{\mathbf{C}}(\mathbf{V}) \in R^{n \times p}$: voltage-source node capacitance matrix.

$\tilde{C}_{ij}$: capacitance connected from voltage source j to Node i

$\mathbf{G}(\mathbf{V}) \in R^{n \times n}$: node-to-datum conductance matrix, except voltage source nodes.

G_{ii}: sum of conductance connected to Node i

G_{ij}: - (conductance connected from Node j to Node i)

$\tilde{\mathbf{G}}(\mathbf{V}) \in R^{n \times p}$: voltage-source node conductance matrix.

$\tilde{G}_{ij}$: conductance connected from voltage source j to Node i

Then, the same procedures from Equation (4.5) apply. An example using linear resistors and capacitors is illustrated in Figure 4.4. From the figure, applying KCL at the three nodes yields the following:

$$\begin{bmatrix} C01+C1+C12 & -C12 & -C01 \\ -C12 & C02+C2+C12 & -C02 \\ -C01 & -C02 & C01+C02 \end{bmatrix} \begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{v}_{in} \end{bmatrix} =$$

$$-\begin{bmatrix} G01+G1+G12 & -G12 & -G01 \\ -G12 & G2+G12 & 0 \\ -G01 & 0 & G01 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_{in} \end{bmatrix}$$

Since v_{in} and $\dot{v}_{in}$ are known, the last equation can be eliminated. By arranging the resulting equations, following equations in the form of Equation (4.13) are obtained:

$$\begin{bmatrix} C01+C1+C12 & -C12 \\ -C12 & C02+C2+C12 \end{bmatrix} \begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} =$$

$$-\begin{bmatrix} G01+G1+G12 & -G12 \\ -G12 & G2+G12 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} G01 \\ 0 \end{bmatrix} v_{in} + \begin{bmatrix} C01 \\ C02 \end{bmatrix} \dot{v}_{in}$$

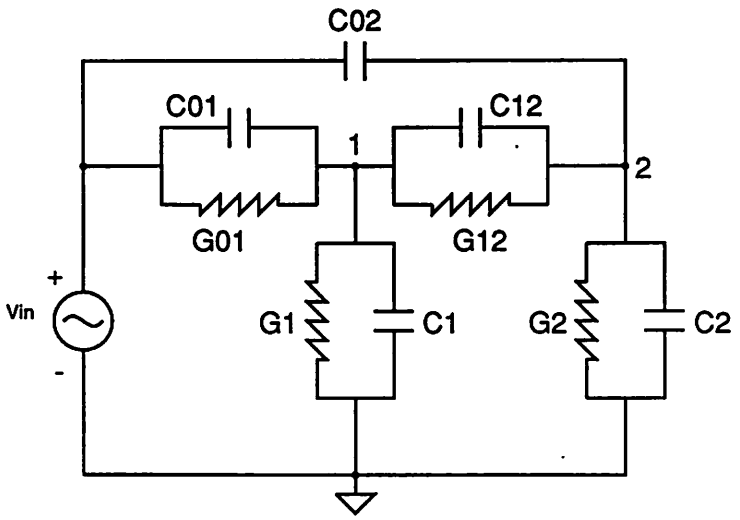


Figure 4.4 An ELogic Circuit Example With A Voltage Source

4.3.2. IMPLEMENTATION OF THE ELOGIC ALGORITHM

ELogic is implemented using the *event-driven* algorithm for efficiency, which has been used successfully for *event-driven simulation* both at logic level [86, 87] and circuit level [84, 88, 39, 58, 80]. Event-driven simulation is a general transient simulation technique which views the time-domain response of a dynamic system as a succession of *events* that have two attributes of the *time* (e_τ) and the *nature* (e_η) of the occurrence. The definition of an event (e_η) and the implementation of the *event scheduler* - the mechanism responsible for finding the next event (or events) and scheduling new events - are program and analysis-level dependent. For the ELogic algorithm, they are defined in the following way:

[Definition 4.4] Event and event-driven implementation of the ELogic algorithm

An *event* is defined as a change in the voltage state of a node. An *event-driven* implementation refers to the fact that Node j is processed at t_n only if an event has occurred at t_n at Node j or at any of its fanin nodes FI_j. □

Processing Node j at t_n involves the following steps:

(1) update the voltage at Node j at t_n.

(2) obtain the next voltage state V_{Next}.

(3) determine whether Node j will make a transition to V_{Next}.

(4) if Node j will make a transition to V_{Next}, compute the transition time (h_j), and schedule Node j again in the time queue at the future time point ($t_{n+1} = t_n + h_j$) at which the node will be processed again.

If any of FI_j changes state while Node j is scheduled in the time queue, the node is *removed* from the time queue and rescheduled after the recomputation of the transition time using the new fanin state (this often happens if the input transition is much faster than the output response at j). If, after j has been processed, ELogic determines that j will not make a transition, then j is removed from the time queue (i.e., it has become *latent*). Node j will then only be processed following an event at a fanin node. Suppose that no nodes change voltage at t_n (i.e., if all nodes have become latent), then no nodes will be processed in the future. In this case, the node voltages at t_n, $\mathbf{v}(t_n)$, are a solution of the system for $t \geq t_n$, as will be proven next.

[Lemma 4.1] Assume that there exists at least one solution for a given time-invariant system

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t)), \quad \text{where} \quad \mathbf{y}(0) = \mathbf{y}_0. \tag{4.14}$$

If an initial slope, $\mathbf{y}'(0)$, is zero, then, $\mathbf{y}(t) = \mathbf{y}_0$ is a solution of the system for $t \geq 0$.

Proof: Suppose that $\mathbf{y}(t) = \mathbf{y}_0$ is a solution of Equation (4.14) for $t \geq 0$. Then, Equations (4.15) and (4.16) are obtained.

$$\mathbf{y}'(t) = 0, \quad \text{for } t \geq 0. \tag{4.15}$$

$$\mathbf{f}(\mathbf{y}(t)) = \mathbf{f}(\mathbf{y}_0) = \mathbf{f}(\mathbf{y}(0)) = \mathbf{y}'(0) = 0, \quad \text{for } t \geq 0. \tag{4.16}$$

From Equations (4.15) and (4.16),

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t)) = 0, \quad \text{for } t \geq 0. \tag{4.17}$$

Since $\mathbf{y}(t) = \mathbf{y}_0$, for $t \geq 0$, satisfies the given system equation, it is a solution of the system.

□

If $\mathbf{f}(\mathbf{y}(t))$ of Equation (4.14) is continuous and satisfies the Lipschitz condition for $0 \leq t \leq b$, $\mathbf{y}(t) = \mathbf{y}_0$ is a unique solution of the system for $0 \leq t \leq b$ [89].

In the case of a parallel LC resonant circuit [62,90], there is only one node. Because the voltage across the LC circuit keeps changing even after reaching a peak voltage (where a slope is zero), Lemma 4.1 does not seem to hold for this circuit at first glance. Notice, however, that the circuit belongs to the class of second-order circuits whose system equation is given as in Equation (4.18)

$$y'' + 2\alpha y' + w_0^2 y = u_s(t). \tag{4.18}$$

rather than first-order circuits of Equation (4.14). The lemma does not apply to a second-order equation. However, the system equation of any second-order circuit made of linear time-invariant elements can *always* be written in the form of first-order state equation as follows:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} a_{11} a_{12} \\ a_{21} a_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} \tag{4.19}$$

In fact, the behavior of the LC resonant circuit is formulated in the form of Equation (4.19), if the Modified Nodal Analysis method [91,90] (usually abbreviated to MNA in the literature) is used, which selects an inductor current as well as a node voltage as network variables. Since the inductor current changes at a timepoint where the node voltage is at its peak, not all the derivatives of network variables are zero as required to apply Lemma 4.1. Therefore, the lemma is not applicable. The ELogic algorithm uses the Nodal Analysis method and hence an event means that a node achieves a new voltage state as defined in Definition 4.4. However, if other analysis methods are used for the implementation of the ELogic technique, the occurrence of any of the network variables achieving a new state must be defined as an event. Assuming MNA is used to solve an LC resonant circuit, if an inductor achieves a new current state, it is an event. A network variable y_j is then processed at t_n only if the variable y_j or any of its fanin variables achieves a new variable state at t_n, where a fanin variable is defined in Definition 4.2.

4.3.3. STABILITY OF THE ELOGIC ALGORITHM

The stability of numerical integration methods is problem-dependent. Therefore, the stability is investigated using following equation whose exact solution is given by $y(t) = e^{-\lambda t}$:

$$\dot{y} = -\lambda y, \quad y(0) = 1 \tag{4.20}$$

where λ and y are generally complex numbers [78,63]. The initial value problem of Equation (4.20) is chosen as the test equation, because it is the *simplest* equation commonly encountered in practice. Even though the behavior of a system is usually represented in the form of nonlinear equations as follows (which is the case when MOS circuits are analyzed):

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}),$$

its local behavior about some operating point $\mathbf{y}_0$ can often be approximated by *variational linear equations*:

$$\delta\dot{\mathbf{y}} = \mathbf{A}\,\delta\mathbf{y}$$

where

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{y})}{\partial \mathbf{y}}\Big|_{\mathbf{y}=\mathbf{y}_0}$$

The region of absolute stability for the explicit Euler integration method is in the circle $|1-\sigma| \leq 1$, illustrated in Figure 4.5, where $\sigma = h\lambda$. When the explicit Euler method is used to solve the test differential equation of Equation (4.20) with a set of values of h and λ inside the stable region, a perturbation in a single mesh value y_n will produce a change in subse-

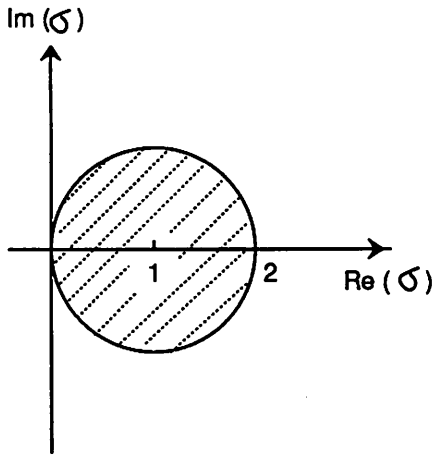


Figure 4.5 Region of Absolute Stability for the Explicit Euler method ($\sigma = h\lambda$)

quent values which does not increase from step to step. Note that a voltage change is bounded in the ELogic algorithm. Therefore, even though the explicit Euler method is not A-stable nor stiffly stable [78, 79], it is possible to keep σ inside the stable region. This is discussed next.

Consider a linear RC circuit shown in Figure 4.6(a), where a voltage at Node j charges up or discharges from V_{Now} to V_{EQ}. Figure 4.6(b) illustrates the charging-up waveform and an ELogic rising transition from V_{Now} to V_{Next} at Node j. The linear RC circuit is meaningful to investigate the stability of the ELogic algorithm, as the linear test equation of Equation (4.20) is meaningful for the investigation of the stability of numerical integration methods. Thus, the stability property of the ELogic algorithm will be derived using the linear RC circuit.

[Lemma 4.2] The explicit Euler method is stable for the linear RC circuit of Figure 4.6(a) in the circle $|1-\sigma| \leq 1$ on σ–*plane*, where $\sigma = h\lambda = \frac{h}{RC}$.

Proof: From the test circuit, a voltage across a capacitor C, $v_c(t)$, and its rate of change are given as follows:

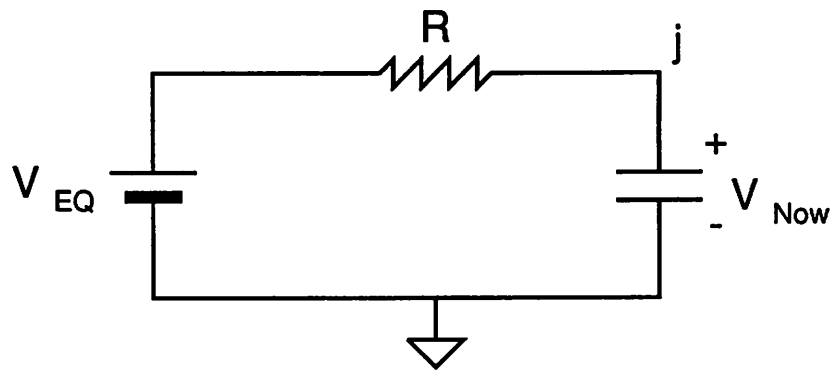
$$v_c(t) = V_{EQ} + (V_{Now} - V_{EQ})\, e^{-\lambda t} \;, \quad where\ \lambda = \frac{1}{RC} \tag{4.21}$$

$$\dot{v}_c(t) = \lambda\,(V_{EQ} - V_{Now})\, e^{-\lambda t} \tag{4.22}$$

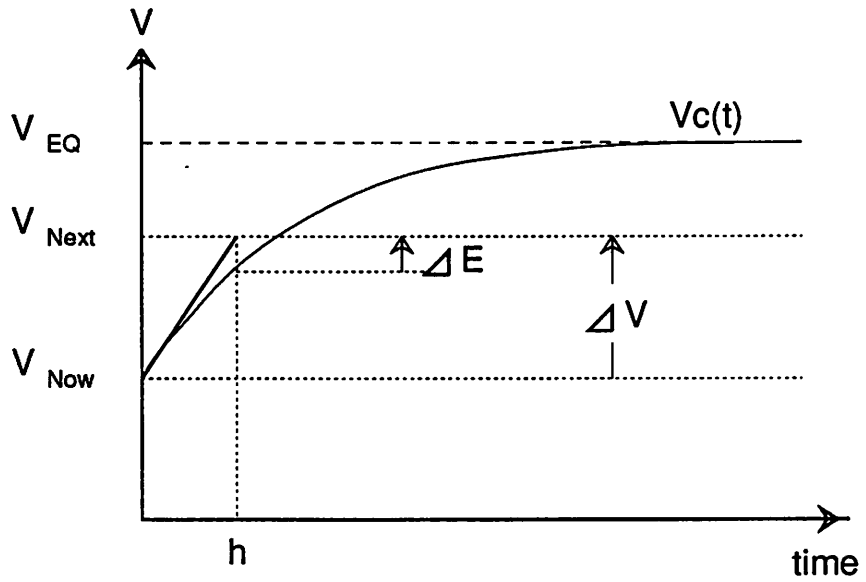
Thus, a system equation is given by

$$\dot{v}_c(t) = -\lambda\, v_c(t) + \lambda\, V_{EQ} \;, \quad where\ \ v_c(0) = V_{Now} \tag{4.23}$$

Let v_n denote a computed value of $v_c(t)$ at t_n by using a difference method of the explicit Euler. To see the change due to a perturbation of size δ in v_n at subsequent time points t_{n+k}, $k>0$, let $\bar{v}_n$ be $(v_n + \delta)$. By solving Equation (4.23) from t_n with $\bar{v}_n$, using the explicit Euler



(a) Linear RC Circuit



(b) Rising Waveform at Node *j*

Figure 4.6 Figures for the Investigation of the Stability of ELogic

integration method, the node voltage at t_{n+k} is computed as follows:

$$\tilde{v}_{n+k} = v_{n+k} + (1 - h\,\lambda)^k\,\delta = v_{n+k} + (1 - \sigma)^k\,\delta \tag{4.24}$$

Therefore, the region of the explicit Euler method for the linear RC circuit of Figure 4.6(a) is in the circle $|1 - \sigma| \leq 1$ on σ–*plane*, as shown in Figure 4.5. □

For the purpose of investigating the effect of the size of a voltage change on the stability of ELogic, it is necessary to study how the location of σ varies on Figure 4.5 by changing the size of a voltage change.

[Lemma 4.3] When the ELogic algorithm is used to solve a linear RC circuit of Figure 4.6(a), the value of σ depends on a particular voltage change Node j makes as follows:

$$\sigma = h\,\lambda = \frac{\Delta V}{V_{EQ} - V_{Now}} \tag{4.25}$$

where ΔV is $(V_{Next} - V_{Now})$, a voltage change Node j makes, and V_{Now} (V_{Next}) is the current (next) voltage state at Node j.

Proof: From Equation (4.22),

$$\dot{v}_c(0) = \lambda\,(V_{EQ} - V_{Now}) \tag{4.26}$$

Because the ELogic algorithm uses the explicit Euler method, the transition time h for Node j to make a voltage change of ΔV is (refer to Figure 4.6(b)):

$$h = \frac{\Delta V}{\dot{v}(0)} = \frac{\Delta V}{\lambda\,(V_{EQ} - V_{Now})} \tag{4.27}$$

Thus, σ is given as:

$$\sigma = h\,\lambda = \frac{\Delta V}{V_{EQ} - V_{Now}} \qquad \square$$

Lemma 4.3 indicates that σ can be bounded inside the stable region by bounding the voltage

changes in the ELogic algorithm.

[Theorem 4.1] The ELogic algorithm is stable for the linear RC circuit of Figure 4.6(a), if and only if it allows Node j to make a transition between discrete voltage states when Conditions 4.1 are satisfied.

Conditions 4.1

(1) The next voltage state (V_{Next}) and V_{EQ} must be in the same direction, when compared to the present voltage state (V_{Now}).

(2) The voltage change ΔV the ELogic algorithm allows must not be larger than twice the difference of V_{EQ} and the present voltage state.

$$| \Delta V | = | V_{Next} - V_{Now} | \leq 2 | V_{EQ} - V_{Now} | \tag{4.28}$$

Proof: Let σ_1 denote σ of an ELogic transition with a particular voltage change. Lemma 4.2 states that the explicit Euler method is stable for the linear RC circuit of Figure 4.6(a), when σ_1 is in the circle $|1-\sigma| \leq 1$ on σ–*plane*. Therefore, it is sufficient to prove that:

Claim4.1.1

If σ_1 is inside the circle $|1-\sigma| \leq 1$, the corresponding ELogic transition satisfies Conditions 4.1.

Claim4.1.2

If an ELogic transition satisfies Conditions 4.1, σ_1 is inside the circle $|1-\sigma| \leq 1$.

Claim4.1.1 will be proven first. Since $\lambda \left(= \frac{1}{RC}\right)$ is real for the test circuit, $\sigma_1 (= h\lambda)$ is also real. Thus, if σ_1 is in the circle $|1-\sigma| \leq 1$, $0 \leq \sigma_1 \leq 2$. From Lemma 4.3,

$$0 \leq \frac{\Delta V}{V_{EQ} - V_{Now}} \leq 2 \tag{4.29}$$

For a rising transition ($V_{EQ} > V_{Now}$), Equation (4.29) yields the following:

$$0 \leq \Delta V \leq 2\,(V_{EQ} - V_{Now}) \tag{4.30}$$

Similarly, followings are derived for a falling transition ($V_{EQ} < V_{Now}$) from Equation (4.29):

$$0 \leq -\Delta V \leq 2\,(V_{Now} - V_{EQ}) \tag{4.31}$$

Since ΔV is $(V_{Next} - V_{Now})$, Equations (4.30) and (4.31) are summarized as Conditions 4.1. Claim4.1.2 can be proven easily by following the proof of Claim4.1.1 in the opposite order. Thus, its proof is omitted. □

Conditions 4.1 are used for ELogic to determine if a node will make a transition to the next state.

4.3.4. ACCURACY OF THE ELOGIC ALGORITHM

Another important issue for discussion is the accuracy of the ELogic algorithm. The ELogic algorithm is convergent when it solves a circuit whose node voltages are monotonically rising or falling: any desired degree of accuracy can be achieved by picking a small enough voltage step (ΔV).

[Theorem 4.2] When the ELogic algorithm solves a circuit whose system equation $\mathbf{v}' = \mathbf{f}(\mathbf{v})$ satisfies a Lipschitz condition and node voltages are strictly monotonically rising or falling for $0 \leq t \leq b$, $v_{j,m}$ approaches $v_j(t)$ for all nodes j for all $0 \leq t \leq b$ as n approaches ∞ with $\Delta V = \frac{V_{\max}}{n}$, where $v_{j,m}$ is a voltage at Node j after m transitions computed by the ELogic algorithm, $\mathbf{v_j(t)}$ is a true voltage at Node j at time $t = \sum_{k=1}^{m} h_{j,k}$, where $h_{j,k}$ is the kth transition time at Node j, ΔV is a voltage step, and $V_{\max}$ is the maximum node voltage in a circuit during operation.

Proof: ΔV approach zero as n approaches ∞. Since all node voltages are strictly

monotonically rising or falling for $0 \le t \le b$, transition times $h_{j,k}$ are defined and approach zero as n approaches ∞ for all nodes j and for all transitions $1 \le k \le m$. The ELogic waveform of the m transitions at node j will be the same as the one produced by a difference method using the explicit Euler with timesteps of $h_{j,k}$, $1 \le k \le m$. Since the explicit Euler method is a convergent method [78], Theorem 4.2 holds. □

In Figure 4.6(b), ΔE illustrates the transition error of the ELogic algorithm after Node j made a transition between two voltage states.

The other local error of the ELogic algorithm is a "round-off" error that occurs due to the precision of ELogic states. While the transition error of ELogic occurs when a node makes a transition to the next state, the round-off error occurs if a node does not make a transition to the next state because Conditions 4.1 are not met. This type of round-off error is obviously bounded by a voltage step ΔV. Note that, as in the case of local versus global truncation error, where bounding the local truncation error at a timestep makes no guarantee about the global error of difference methods after many timesteps especially when nonlinear circuits are solved, the choice of a voltage step-size in ELogic bounds the error at each transition but provides no guarantee about the global error after many transitions. However, our experience with ELogic is that on average the local errors do not accumulate, if transitions are allowed only when the local behavior of a nonlinear circuit at the operating point satisfies Conditions 4.1. Of course, a pathological nonlinear circuit could be constructed in order to amplify the error due to a choice of state-values. In this case, we would argue that this was a poor choice of states for the particular circuit or design-style. Such circuits can be found even for accurate circuit simulators using any finite floating-point precision.

4.4. THE ELOGIC ALGORITHM FOR MOS CIRCUITS

In ELogic, the Shichman-Hodges model [92] equations are used to represent MOS transistors. If desired, more sophisticated models, or a table-lookup model [93, 94, 95, 96] can be used to improve accuracy.

When processing Node j without floating capacitors at t_n, ELogic determines if it will make a transition to next state, as follows. First, ELogic linearizes MOS transistors using the small-signal model [63] and obtains the Thevenin equivalent circuit [62] of fanin nodes of Node j, FI_j, at a given timepoint t_n. The Thevenin equivalent circuit of the FI_j at j can be obtained by source transformation after computing the Norton equivalent circuit of each of FI_j. The Thevenin equivalent circuit represents the local behavior of the circuit to Node j at t_n. The Thevenin equivalent circuit has the same configuration as the stability test circuit shown in Figure 4.6(a). Thevenin equivalent voltage source, V_{EQ}, is the open-circuit voltage at Node j at t_n, which can be obtained after disconnecting a grounded capacitor at the node, $C_{j,j}$. Then, ELogic computes the next voltage state by comparing V_{EQ} and the present state, V_{Now}. Node j will make a transition to the next voltage state only if the conditions of Equation (4.30) or (4.31) are satisfied. As an illustration, one might consider two voltage states, $V_k < V_{k+1}$. Let the threshold voltage V_{th} be the middle value of the two voltage states. Assuming that the present state is V_k, if V_{EQ} lies above V_{th}, Equation (4.30) is satisfied and hence a rising transition is made.

While the previously described scheme works very well in practice, occasionally it is possible for nodes to oscillate between two adjacent states without changes in the state of the fanin nodes. This numerical, adjacent-state oscillation may occur when the exact steady-state node voltage is not close enough to a discrete voltage state. This artificial oscillation is illustrated in Figure 4.7. Let V_{exact} denote the exact steady-state voltage at Node j. In the figure,

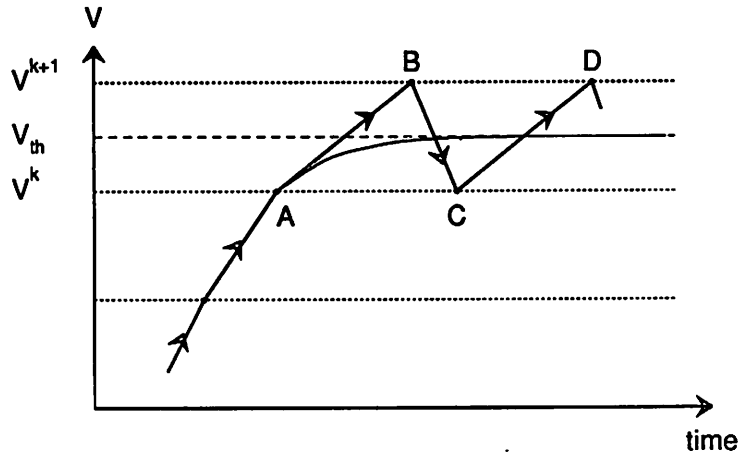


Figure 4.7 Illustration of Numerical, Adjacent-State Oscillation

V_{exact} lies between V_k and V_{k+1}. Assume that the present voltage state of Node j is V^k (A in Figure 4.7) and V_{EQ} lies between V_{th} and V^{k+1}. Then, Node j makes a transition from V^k to V^{k+1}. When the voltage at Node j reaches V^{k+1} (B in Figure 4.7), V_{EQ} is computed again. If the new V_{EQ} remains between V_{th} and V^{k+1}, the voltage state of Node j will remain at V^{k+1} (this is what happens in most cases). However, because one of the input voltage states used to evaluate MOS transistors changed from V^k to V^{k+1}, occasionally, the new V_{EQ} may move to between V^k and V_{th}. In this case, Node j will make a transition back to V^k and artificial oscillation occurs. In order to prevent the adjacent-state oscillation, the ELogic algorithms employs a *cycle detector*: the cycle detection is performed at each node as it makes a transition to an adjacent state. If the transition reverses a direction more than once, and the event that caused the transition was not external to the node (i.e not a state change at a fanin node), then the cycle detector prevents the node from making the second reversal transition (transi-

tion *C-D* in Figure 4.7). One must note that this cannot suppress real oscillations in the circuit, since they *must* involve more than two states or a change in state at at least one fanin node. If a local oscillation occurs within a state (i.e., below the resolution of the analysis technique) and, if such an oscillation is truly part of the circuit characteristic, then more states should be used to model its behavior effectively.

Now, consider Node j having floating capacitors. As mentioned earlier, when ELogic processes the node, it computes the Norton equivalents of fanin nodes (FI_j) to obtain the Thevenin circuit at j. Note that the Norton equivalent of a capacitor in series with a voltage source involves a current source which is a function of the rate of change of the value of the voltage source. Therefore, if there is a floating capacitor between j and FI_j, the algorithm has to compute the rate of change of the voltage at FI_j to obtain the Norton equivalent circuit of FI_j at t_n. However, computing voltage change rates at FI_j is expensive. Furthermore, if Node j has no resistive elements, the Norton equivalent circuit at the node consists of only a current source and a capacitance. Hence V_{EQ} cannot be computed and the Thevenin equivalent at the node does not exist. Therefore, if Node j has floating capacitors and its voltage change rate is not zero, ELogic determines the node makes a transition to the next voltage state. From Equation (4.10), a voltage change rate at Node j is given as follows:

$$\textit{rate of voltage change at Node } j \; = -\mathbf{C_j^{-1}(v_n)}\ \mathbf{f}\,(\mathbf{v_n},\mathbf{u_n})$$

As it does for nodes without floating capacitors, the cycle detector prevents Node j from making the second reversal transition to suppress artificial oscillation, when j reverses the transition direction more than once, and the event that caused the transitions was not a state change at FI_j.

4.5. THE ELOGIC DELAY MODEL FOR TIMING VERIFICATION

The ELogic algorithm can be used for as a delay model for timing verification as follows. First, the path analysis section extracts a chain of transistors from a circuit, in a systematic way: breadth-first, depth-first, or topological-order-based manner. The input to the chain of transistors is a waveform fragment. The effect of the input change propagates as waveform fragments through the circuit. Then, the delay is obtained by comparing the waveform fragments at nodes in the transistor chain. Since the size of the transistor chain is usually small, the ELogic algorithm is practical for delay evaluation, even when the chains have floating capacitors.

The accuracy of the ELogic delay model has been compared to that of existing switch-level delay models, using the switch-level critical-path analyzer, Crystal [23]. Crystal has three RC delay models: lumped RC model, lumped slope RC model, and distributed slope RC model. Crystal was modified to use the ELogic delay model, while the path analysis section was left the same [97, 61]. The modified version of Crystal will be referred to as *E–Crystal* in this dissertation. The data structure of E-Crystal was modified to store the waveform segment. This waveform segment is used as an input to other transistor chains, extracted later. As mentioned earlier, RC delay models assume that there is only one direct path from a reference node (power supply or ground) to the signal-nodes of the circuit. Thus, a transistor chain extracted by Crystal is a series of pass transistors from the reference node to a gate or output node. This chain, as is, may not be the correct subcircuit to simulate using the ELogic delay model. For example, when the falling transition of NMOS inverter is examined, only the NMOS driver is extracted and the NMOS depletion load is missing. Therefore, a function which extracts missing elements, if any, from the circuit description was added. Simulation by the ELogic delay model is performed until an output node reaches

a steady state. After the simulation is finished, E-Crystal updates the worst delay waveform segment on the output node. While E-Crystal uses the ELogic delay model, the detailed implementation is slightly different from the ELogic delay model used in *E-TV*, described earlier in this chapter. The detailed implementation of the ELogic delay model used in E-Crystal can be found in [60, 97].

The accuracy of the ELogic delay model employed in E-Crystal was compared to that of Crystal's distributed slope RC model (DS model), using a CMOS microprocessor SOAR (Smalltalk On A RISC) chip [98] and its ALU (Arithmetic Logic Unit). The DS model uses the ratio approach [65] and the results of Penfield-Rubenstein model [64]. It is the most accurate among Crystal's three RC delay models.

Depending on the voltage step chosen for the ELogic delay model, the DS model has been shown to be about 30 to 300 times faster. Note that the only difference of E-Crystal and Crystal is their delay model. In this case, the CPU times directly reflect the efficiency of the delay model employed. However, CPU times of timing verification also depend on the path analysis approach. For this reason, *E-TV* uses the topological-order based search for path analysis, which is more efficient than a modified depth-first search used in Crystal. As described in Chapter 3, the topological-order based search locates the worst delay path to each node only once, while a modified depth-first search locates the worst delay path to each node more than once on the average and the running time is exponential with the circuit size in the worst case. Experimental results of *E-TV* can be found in Chapter 6.

The ALU of SOAR has 1694 transistors and 1189 nodes. Both Crystal (DS model) and E-Crystal using 0.5V steps ran to extract 15 worst delay paths of the circuit. Figure 4.8 illustrates the results. In the figure, the X axis represents the delays by SPICE2 and the Y axis represents the delays estimated by Crystal and E-Crystal. If the delay estimates by Crystal or

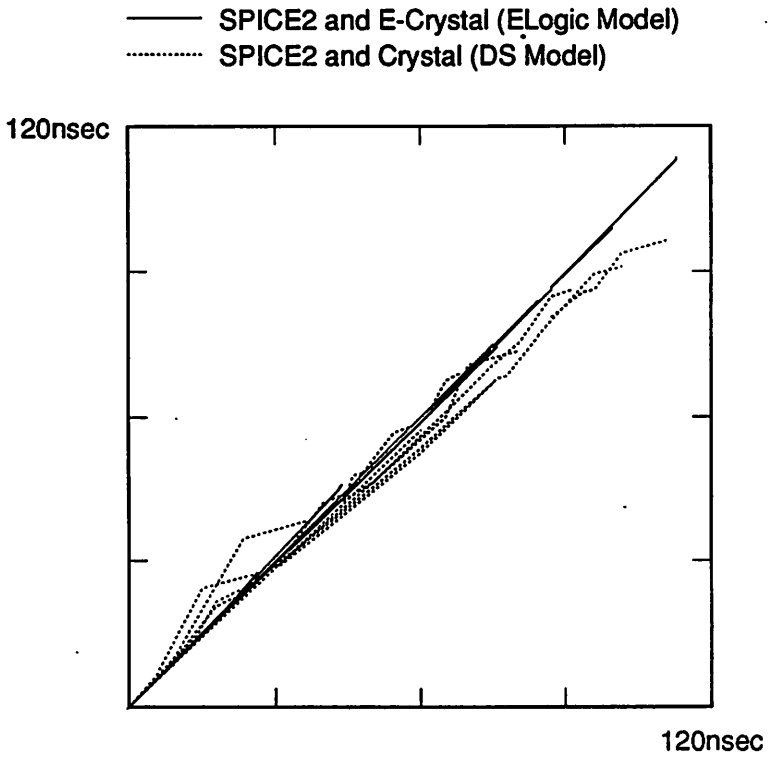


Figure 4.8 Delay Estimates Comparison on SOAR ALU
X-axis: SPICE2 Y-axis: Delay Model

E-Crystal are exactly the same as those by SPICE2, a 45° straight line from left bottom to right top corner appears. The dotted lines compare the delay estimates by Crystal and SPICE2, for 15 worst delay paths extracted by Crystal. The solid lines compare the delay estimates by E-Crystal using 0.5V step and SPICE2, for 15 worst delay paths extracted by E-Crystal. Each line of the figure corresponds to one worst delay path from input nodes to output nodes of the ALU. Each segment of the line corresponds to a chain of transistors in the path. The breakpoint represents the delay estimates up to that transistor chain. From Figure 4.8, it is observed that the ELogic delay models provide more accurate delay estimates than the DS model. One should notice that the DS model alternately underestimates and overestimates the delay through a chain of transistors. Thus, for this well-behaved example, the total error in delay estimation of path is decreased by compensation. However, the DS delay model may not find the right worst delay path, especially when it is used in critical-path analysis where the search is pruned.

The SOAR chip has 34,526 transistors and 13,311 nodes. This time, the delay estimates of the two delay models were compared using the same paths. First, 12 worst delay paths of the SOAR chip were extracted by Crystal. Then, delays through these paths were estimated by E-Crystal with 0.2V step and SPICE2. The results are given in Table 4.1 as

Verifier	Model	Overall Error(%)	Avg. Path Error(%)
Crystal	DS Model	-19.3	46.4
E-Crystal	ELogic (0.1V step)	-1.2	1.6
	ELogic (0.2V step)	0.45	4.2

Table 4.1 Delay Estimates Comparison on SOAR Chip

well as in Figure 4.9. In the table, "Overall Error" is an error of total delay estimates obtained for all worst delay paths by delay models compared to that of SPICE2. "Avg. Path Error" is the average of the absolute error in estimating total delays in the worst delay paths up to each chain of transistors. In Figure 4.9, the delay estimates by the DS model spreads much more widely than in Figure 4.9, since the underestimation and the overestimation did not alternate as for ALU. This fact accounts for the large error of the DS model shown in Table 4.1. In Figure 4.9, SPICE2 determines that Path *A* is the worst delay path among the extracted 12 paths. However, the DS model in Crystal fails in ordering delay paths and reports Path *B* as the worst delay path. The results indicate that the DS model, while it is the most accurate among three RC delay models available, does not have good absolute accuracy nor good relative accuracy in this example. Even though the worst delay paths reported by timing verifier are frequently re-evaluated using more accurate technique, the evaluation of wrong worst delay paths such as Path *B* is not helpful. Since the ELogic delay model is more accurate, it certainly reduces the chance of reporting wrong worst-case delay paths, as illustrated in the figure.

The experimental results demonstrate the use of the ELogic delay model improves the accuracy of delay estimates greatly over the existing RC delay models. While the accuracy improvement can also be achieved by using the standard simulation approach, it is too slow to be practical. The ELogic delay model is reasonably fast and accurate. Compared to SPICE [5, 41], ELogic typically runs a factor of 30 to 300 times faster and produces a delay error of 1% to 20% when it uses 0.1V to 1V step-size, depending on the nature of the circuit. More comparisons of accuracy and simulation speed between ELogic and SPICE can be found in Chapter 6 of this dissertation and in [61]. The ELogic delay model also provides a trade-off between speed and accuracy in the same analysis as well as over a number of ana-

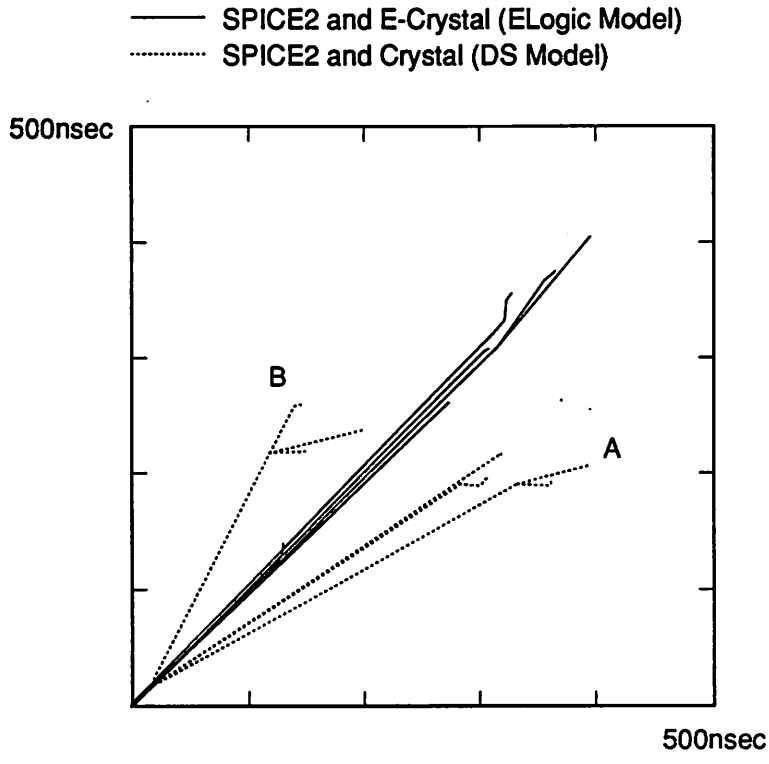


Figure 4.9 Delay Estimates Comparison on SOAR
X-axis: SPICE2 Y-axis: Delay Model

lyses, by varying the number of voltage states [61]. In addition, a waveform at a node with the worst delay is used as an input to other transistor chains. The easiest way to store a waveform is using a list of voltage-timepoints for a transition (waveform segment). When the input waveform is not given as a smooth function, the standard simulation approaches have difficulty converging. The ELogic delay model handles such a waveform segment without difficulty, when it is used as an input to transistor chains extracted later. These are some of motivations to develop a timing verifier that uses the ELogic delay model.

CHAPTER 5

ELECTRICAL-LOGIC BASED TIMING VERIFIER

(E-TV)

In this chapter, the concept and the approaches employed for the development of an accurate timing verifier for MOS digital VLSI systems, referred to as *Electrical-logic based Timing Verifier (E-TV)*, are presented.

The main purpose of *E-TV* is to detect the possibility of timing errors that violate the maximum delay constraints in multiphase synchronous and combinational MOS digital systems. If a signal does not get to a space point by the intended timepoint, it is regarded as a timing error. In addition, *E-TV* serves as a *critical path finder* which locates the critical paths in a system. For a synchronous system, *E-TV* reports the paths in each logic segment in the critical order using their *evaluation-time margin.* For a given clocked path, the evaluation-time margin is defined as a time interval from the transition timepoint at the succeeding space reference to the succeeding time reference of the path. Thus, it represents the severity of the timing problem of the clocked path. This information can be used to improve the speed of the system. It was noted that a timing verifier or critical path finder can be only as accurate as its delay model. The experimental results illustrated that *E-TV* is not only fast but also accurate enough to be practical. The experimental results are presented in Chapter 6. In addition, *E-TV* provides a speed-accuracy trade-off, as with the ELogic algorithm.

Like most other timing verifiers, *E-TV* propagates logic signals through a system value-independently. Hence it computes the worst-case path delays and uses them to detect

timing errors or to locate the critical paths. *E-TV* uses a critical-path analysis approach: the search space is pruned and only the critical paths of a system are examined. *E-TV* visits each node once in the topological order so as to find the worst delay path leading to it. Thus, as mentioned in Chapter 3, the running time of *E-TV* is approximately linear with a circuit size. After extracting candidate transistor chains for the worst delay path to each node, *E-TV* computes path delays using the ELogic delay model. As does the leveling, the topological ordering assumes that there are no loops in a graph. Therefore, *E-TV* breaks signal-flow loops in a circuit while it orders nodes topologically. For the verification of a synchronous system, *E-TV* propagates clock signals through the system. Then, it uses the actual clock signals that arrive at clocked storage elements to check timing violations. Therefore, clock skew is taken into account automatically.

5.1. SIGNAL FLOW THROUGH MOS TRANSISTORS

The direction of signal flow is the direction that a logic "0" or "1" signal flows through an MOS transistor. Note that the logic "0" flows in the opposite direction of current flow while the logic "1" signal flows in the same the direction. As an example, when the output node of an inverter discharges, the current flows from the output node to ground through a pull-down transistor, but the logic "0" signal flows from ground to the output node. In principal, the logic signal can flow in either direction through an MOS transistor channel, from drain to source terminals or from source to drain terminals. Therefore, each MOS transistor channel constructs a simple signal-flow loop which must be broken for the topological ordering of nodes. In a typical design, however, most transistors are *unidirectional* such that signals flow only in one direction during operation, and only a small number of MOS transistors are *bidirectional*. When breaking a signal-flow loop by an unidirectional transistor, it is very

important to preserve an edge which represents its intended signal flow, and remove the other edge to break the loop. The removal of a wrong edge blocks forward paths through the unidirectional transistor, and causes unrealistic paths to be examined. Consider Figure 5.1, where all MOS transistors are unidirectional. Signals are intended to flow through MOS transistors as represented by arrows. Assume that the loop by the source-drain channel of *M1* was broken so that a signal flows from Node *2* to Node *1*, rather than as shown. Then, a real path *A* is blocked and, instead, a bogus path *B* is examined. This illustrates that, for switch-level timing verification, it is crucial to find all unidirectional MOS transistors and to assign the directions of signal flow through them correctly.

5.1.1. AUTOMATIC DERIVATION OF SIGNAL FLOW

There are two possible approaches for assigning the direction of signal flow:

(1) The circuit designer specifies the signal flow directions during design.

(2) A computer algorithm derives the signal flow directions after the design is completed.

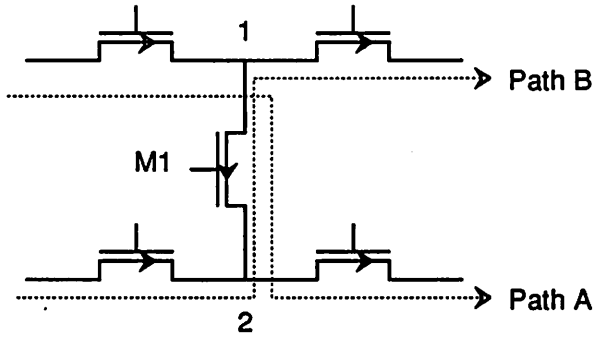


Figure 5.1 Breaking A Loop of An Unidirectional Transistor

The first approach imposes an extra burden on the designer. However, they can certainly *set* the signal flow through unidirectional MOS transistors correctly, since they understand circuit's function. In many cases, cells are replicated or created by synthesis tools, reducing the amount of work required by the designer. Additionally, while a bidirectional MOS transistor transfers logic signals in both directions, the designer may assign the signal flow so that more important paths can be examined. Thus, this approach is preferable to the second one. Nevertheless, the second approach eases the user's task of direction finding, when it is necessary to verify a design without signal flow identification. For this purpose, *E-TV* provides a program, called *FLOW*, which the user can run to set the signal flow in a design automatically in advance. In this section, the approach used in *FLOW* is described.

Consider the signal flow through an inverter. The logic "0" signal flows from ground to an output node through a pull-down transistor, and the logic "1" signal flows from a power supply to the output node through a pull-up transistor, because the ground and power supplies are working as logic information sources. If the drain or the source terminal of an MOS transistor is connected to ground or a power supply, a logic signal can flow only from the ground or the power supply to the other terminal. This can be used as a rule to set the signal flows through MOS transistors whose source or drain terminal is connected to ground or to any power supply. In fact, *FLOW* uses a set of rules to find unidirectional MOS transistors and to assign the signal flow directions through them. Recently, such approaches, which derive the directions of signal flow through MOS transistors using a set of rules, have been used in some switch-level timing verifiers such as TV [99] and Pearl [57]. For example, TV applies a set of safe rules, then unsafe rules, repeatedly to determine transistor signal flow. However, its rules for NMOS circuits, that depend on transistor ratios to find pass transistors, are said to be virtually unusable for CMOS circuits [57]. Thus, in Pearl, it is attempted to

prove that flow is impossible from either the source to drain or drain to source by searching for a transistor gate, feedback paths, and power supplies through transistor channels to some maximum search depth. The limitation of this approach is that the program can get lost in circuits that require inordinate amounts of computation, unless the search depth is limited to reasonable level. On the other hand, *FLOW* defines two kinds of information sources, as will be described soon. Then, rules are applied to the transistors having their source or drain terminal connected to the candidate nodes that are updated during the signal-flow derivation, starting from the information source nodes. Therefore, the signal flow can be derived efficiently. It should be noted that some unidirectional transistors may be left bidirectional, but no bidirectional elements can be set for the signal flow.

The basic rules developed for *FLOW* to derive the signal flow are as follows.

Rule 1. *Information source, sink, station, and signal flow* :

Nodes with a voltage source or ground are *strong* information source nodes by their definition. Input nodes to the design are *weak* information source nodes by the designer's intention. Similarly, the output node of an and-or-inverter serves as *weak* information source for other transistors. Logic signals flow from strong information source nodes to weak information source nodes, or from information source nodes to non-information-source nodes (such as information station/sink nodes that are described next) through the source-drain channels of MOS transistors. Output nodes from a design and the gate terminals of MOS transistors serve as information sink such that an entered logic signal does not necessarily have to leave. All other nodes serve as information station, where a logic signal which entered through one path must leave through another.

Rule 2. *One unset transistor at a node* :

For a given node *X* which is not an information sink, if all but one of the MOS transistors having their source or drain terminal connected to *X* are set for signal flow, and a logic signal enters *X* through them, the logic signal must leave *X* through the unset transistor. Similarly, for a given node *X* which is not an information source, if all but one of the MOS transistors having their source or drain terminal connected to *X* are set, and a logic signal leaves *X* through them, a logic signal must enter *X* through the unset transistor.

Rule 3. *Pseudo transistor*:

Two transistors, whose source or drain terminals are connected to the same node *X*, have the same direction of signal flow relative to Node *X*, if the two transistors turn on and off simultaneously or mutually exclusively by other than fixed voltage sources. Thus, these transistors can be viewed as one pseudo transistor, when applying **Rule 2.**

Rule 4. *Information circulation through loops* :

The logic signal does not circulate through loops made of the source-drain channels of MOS transistors.

FLOW first finds all and-or-inverters in a design and sets the signal flow through their pull-up and pull-down transistors toward output nodes. *FLOW* then sets the signal flow through transistors having their source or drain terminal connected to ground, any power supply, or the output node of an and-or-inverter, using **Rule 1.** Note that pass transistors connected between the output nodes of two and-or-inverters are not set, because they have the same strength. Then, *FLOW* sets the signal flow through unset transistors by applying **Rules 2** and **3** to nodes, repeatedly. During the signal-flow derivation, *FLOW* maintains a list of candidate nodes to apply rules. If *FLOW* sets a transistor by examining its source/drain node, the condition at the other source/drain node changes. Hence the node becomes a candidate

for a examination. When the candidate list becomes empty, the signal-flow derivation is completed and unset transistors are considered as bidirectional.

5.1.2. PSEUDO TRANSISTORS FOR SIGNAL-FLOW DERIVATION

It has been mentioned in **Rule 3** that a number of transistors can be viewed as one pseudo transistor for the purpose of the signal-flow derivation in some cases. In this section, the other two situations where *FLOW* views a number of transistors as one pseudo transistor are presented.

When a number of transistors are connected in parallel between two nodes, they form a loop or loops made of the source-drain channels of the transistors. As described in **Rule 4**, information does not circulate through the loop. Thus, the transistors have the same direction of signal flow relative to two nodes and hence can be viewed as one pseudo transistor for the signal-flow derivation, when applying **Rule 2**.

A *biconnected* graph is a connected subgraph with no cut-vertices, which has only two end points connected outside of it. Roughly speaking, in order for a logic signal to pass through a biconnected graph, one end point must serve as an entrance and the other one must serve as an exit for the passage. Therefore, when applying **Rule 2**, a group of transistors whose source-drain channels form a biconnected graph behaves as one pseudo transistor at two end nodes for the direction derivation, provided that none of the source/drain nodes of the transistors serves as an information source or sink.

[Lemma 5.1] A group of transistors whose source-drain channels form a biconnected graph behaves as one pseudo transistor connecting the two end nodes for the signal-flow derivation, provided that none of the source/drain nodes of the transistors serves as an information source or sink.

Proof: Let G_B denote a biconnected graph formed by MOS transistors, where edges represent their source-drain channels. Figure 5.2 illustrates nine possible cases, depending on the signal flows through edges of G_B connected to two end nodes X and Y. Note that all transistors of G_B connected to X or Y can be represented by one edge without a loss of generality, if their directions of signal flow are the same with respective to X or Y. Due to the fact that the information does not circulate through a loop (**Rule 4**), followings are required for Figures 5.2(a) to (g) to be valid:

1. (a) and (e) must have information sink in G_B.
2. (b) must have information source or sink in G_B.
3. (c) and (f) must have information source in G_B.
4. (d) must have information sink in G_B.
5. (g) must have information source in G_B.

Therefore, if the biconnected graph G_B contains no information source and sink nodes, only Figures 5.2(h) and (i) are possible. In both (h) and (i), all transistors in G_B, that are connected to X or Y, have the same direction of the signal flow with respect to X or Y. Thus, a group of transistors forming a biconnected graph behaves as one pseudo transistor for the signal-flow derivation of transistors connected to two end nodes. □

FLOW uses the above lemma when applying **Rule 2** at end nodes. By doing so, it improves the chance of assigning the signal flow, compared to existing approaches. For those transistors in G_B that are not connected to end nodes, the signal flow is set to prevent information loops. Some of the transistors may be bidirectional. An example of transistors forming a biconnected graph is the pull-up or the pull-down transistor group of an and-or-inverter.

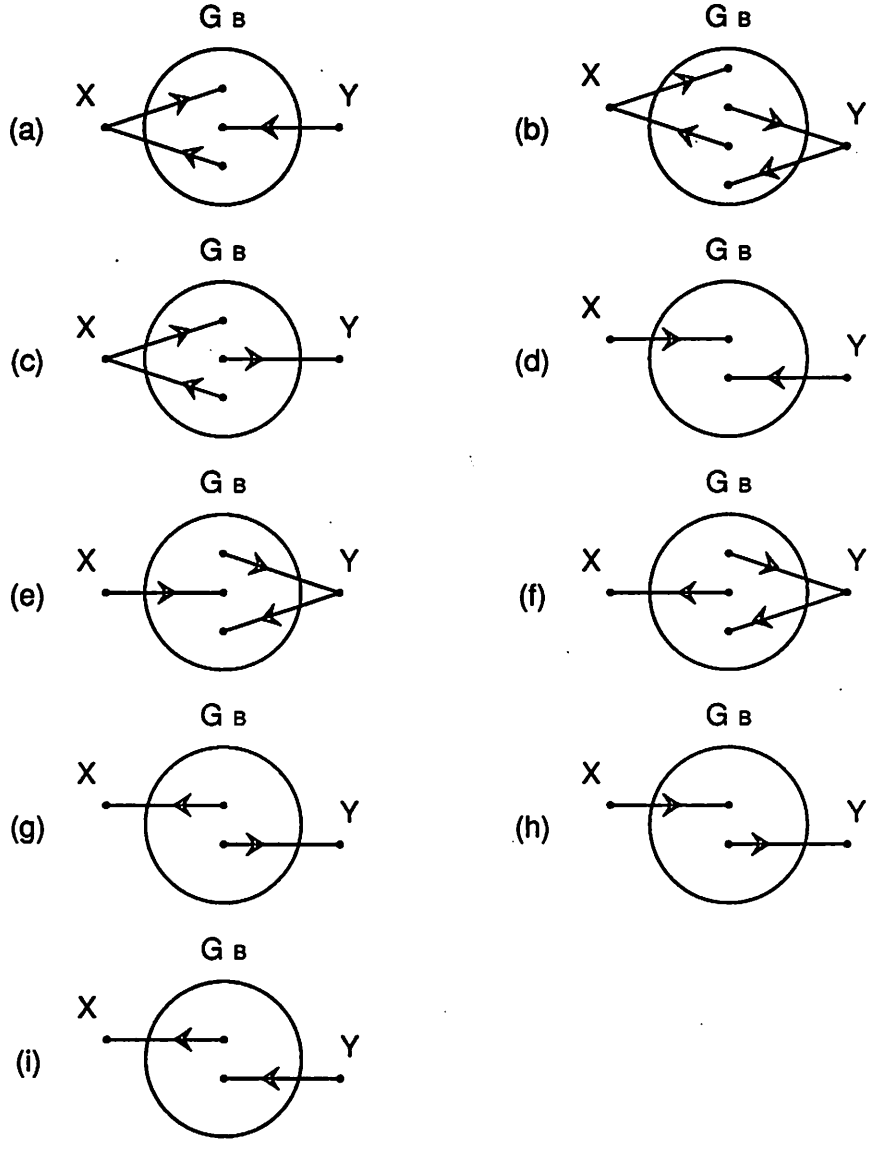


Figure 5.2 Signal Flow through A Biconnected Graph

5.2. WORST-CASE OPERATIONS

E-TV defines two types of events, depending on the method for signal propagation: *value-dependent* and *value-independent events*. A value-dependent event is the one which is made possible by value-dependent signal propagation. It is guaranteed to occur at given or computed timepoints, since all necessary conditions are supported. On the other hand, the value-independent event is made possible by value-independent signal propagation. The event is *expected* to occur at the given/computed timepoints, only if all necessary conditions are supported. Because it is not known if all necessary conditions are supported, its occurrence is uncertain. Value-dependent and value-independent signal propagations have been described in Chapter 3. The events include the rising or falling transition of node voltages, switching of pass elements, or open/close operations of paths. When a transition at a node is value-dependent, the node is said to be value-dependent. Otherwise, the node is said to be value-independent.

In *E-TV*, all signal nodes except clock nodes and *logic-control nodes* (to be described in Section 5.3) are value-independent. Therefore, the analysis of a particular transistor chain may involve some value-dependent nodes and some value-independent nodes. This section describes the effect of event types in determining the timepoint at which a path opens and closes in the worst case and the predecessor which is responsible for the worst-case operation. The results are used to find the worst (longest) delay predecessor. For example, the output of an NMOS and-or-inverter is pulled up after all pull-down paths become open. Thus, the worst rising delay predecessor will be the input node which opens its pull-down path last.

5.2.1. SERIES CONNECTION OF TWO PASS ELEMENTS

Consider Figure 5.3, where two pass elements (*PE1* and *PE2*) are connected in series. Assume that Node *A* is value-dependent. The open operation of a path is investigated. Let $T_{OFF}(PE1)$ and $T_{OFF}(PE2)$ denote the times at which *PE1* and *PE2* turn off in the worst case, respectively. Because the worst-case time point at which Path *A-C-E* closes or opens is investigated instead of the worst-case path delay, the delay through *PE1* or *PE2* needs not be considered. First, suppose that *PE1* and *PE2* turn off value-dependently. Then, because both pass elements are guaranteed to turn off at the given times, the following results are obtained:

(1) Worst-case operation 1 (series connection, open operation)

Conditions

(1) Series connection of two pass elements (Figure 5.3).

(2) *PE1* and *PE2* turn off at $T_{OFF}(PE1)$ and $T_{OFF}(PE2)$, value-dependently (Node *B* and *D* are value-dependent).

Results

(1) The predecessor for the worst-case operation is Node *B*, if *PE1* turns off first. Otherwise, the predecessor for the worst-case operation is Node *D*.

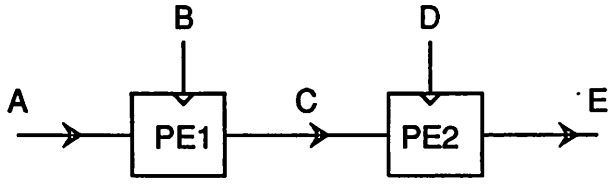


Figure 5.3 Two Pass Elements Connected in Series

(2) The path opens at *Min* ($T_{OFF}(PE1)$, $T_{OFF}(PE2)$) in the worst case.

(3) Node *E* becomes value-dependent.

Now, suppose that *PE1* and *PE2* turn off value-independently. There is no guarantee that the pass elements will turn off at the specified times. Thus, the following results are obtained:

(2) Worst-case operation 2 (series connection, open operation)

Conditions

(1) Series connection of two pass elements (Figure 5.3).

(2) *PE1* and *PE2* turn off at $T_{OFF}(PE1)$ and $T_{OFF}(PE2)$, value-independently (Node *B* and *D* are value-independent).

Results

(1) The predecessor for the worst-case operation is Node *B*, if *PE1* turns off later. Otherwise, the predecessor for the worst-case operation is Node *D*.

(2) The path opens at *Max* ($T_{OFF}(PE1)$, $T_{OFF}(PE2)$) in the worst case.

(3) Node *E* becomes value-independent.

The next case is where one pass element turns off value-dependently, while the other one turns off value-independently. Suppose that *PE1* turns off value-dependently and *PE2* turns off value-independently. If *PE1* turns off first, Path *A-E* will open at that time. However, when $T_{OFF}(PE1) > T_{OFF}(PE2)$, some conditions that are necessary for *PE2* to turn off at $T_{OFF}(PE2)$ may not be supported. As a result, it is not clear if *PE2* turns off at $T_{OFF}(PE2)$, earlier than *PE1*. In the worst case, the path may open when *PE1* turns off. Therefore, regardless of which one turns off first, the worst-case open-operation predecessor of the path is Node *B*.

(3) Worst-case operation 3 (series connection, open operation)

Conditions

(1) Series connection of two pass elements (Figure 5.3).

(2) *PE1* turns off at $T_{OFF}(PE1)$ value-dependently and *PE2* turns off at $T_{OFF}(PE2)$ value-independently (Node *B* is value-dependent and Node *D* is value-independent).

Results

(1) The predecessor for the worst-case operation is Node *B* which is value-dependent

(2) The path opens at $T_{OFF}(PE1)$ in the worst case.

(3) Node *E* becomes value-dependent.

From **Worst-case operations 1 to 3**, the following observation is obtained.

[Observation 5.1] Worst-case open-operation of series connection

Consider two pass elements, connected in series as in Figure 5.3. Suppose that Node *A* is value-dependent. Then, the path opens value-dependently if, and only if, at least one pass element turns off value-dependently. The predecessor of the path for the worst-case open-operation is the gate of a pass element which turns off value-dependently, or the gate of the pass element which turns off first if both turn off value-dependently. The worst-case path-open time is the first time at which any pass element turns off value-dependently. □

Now, consider the worst-case close-operation of the path. Let $T_{ON}(PE1)$ and $T_{ON}(PE2)$ denote the times at which *PE1* and *PE2* turn on in the worst case, respectively. The path closes only when both pass elements turn on. Therefore, the worst-case close time of Path *A-E* depends on the pass element which closes later, regardless of node type.

(4) Worst-case operation 4 (series connection, close operation)

Conditions

(1) Series connection of two pass elements (Figure 5.3).

(2) *PE1* and *PE2* turn on at $T_{ON}(PE\,1)$ and $T_{ON}(PE\,2)$.

Results

(1) The predecessor for the worst-case operation is the gate of a pass element which turns on later.

(2) The path closes at $Max\,(T_{OFF}(PE\,1), T_{OFF}(PE\,2))$ in the worst case.

(3) The node type of E follows the type of the worst-case operation predecessor.

5.2.2. PARALLEL CONNECTION OF TWO PASS ELEMENTS

The effect of event types on parallel connection of pass elements is found to be the dual of the effect on series connection. The worst-case operations of two pass elements connected in parallel, which is shown in Figure 5.4, are summarized as follows.

(1) Worst-case operation 5 (parallel connection, open operation)

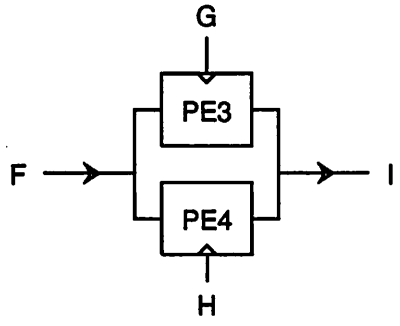


Figure 5.4 Two Pass Elements Connected in Parallel

Conditions

(1) Parallel connection of two pass elements (Figure 5.4).

(2) *PE3* and *PE4* turn off at $T_{OFF}(PE3)$ and $T_{OFF}(PE4)$.

Results

(1) The predecessor for the worst-case operation is the gate of a pass element which turns off later.

(2) The path opens at $Max(T_{OFF}(PE3), T_{OFF}(PE4))$ in the worst case.

(3) Node type of *I* follows the type of the worst-case operation predecessor.

(2) Worst-case operation 6 (parallel connection, close operation)

Conditions

(1) Parallel connection of two pass elements (Figure 5.4).

(2) *PE3* and *PE4* turn on at $T_{ON}(PE3)$ and $T_{ON}(PE4)$, value-dependently (Node *G* and *H* are value-dependent).

Results

(1) The predecessor for the worst-case operation is Node *G*, if *PE3* turns on first. Otherwise, the predecessor for the worst-case operation is Node *H*.

(2) The path closes at $Min(T_{ON}(PE3), T_{ON}(PE4))$ in the worst case.

(3) Node *I* becomes value-dependent.

(3) Worst-case operation 7 (parallel connection, close operation)

Conditions

(1) Parallel connection of two pass elements (Figure 5.4).

(2) *PE3* and *PE4* turn on at $T_{ON}(PE3)$ and $T_{ON}(PE4)$, value-independently (Node *G* and *H* are value-independent).

Results

(1) The predecessor for the worst-case operation is Node *G*, if *PE3* turns on later. Otherwise, the predecessor for the worst-case operation is Node *H*.

(2) The path closes at $Max\,(T_{ON}(PE3), T_{ON}(PE4))$ in the worst case.

(3) Node *I* becomes value-independent.

(4) Worst-case operation 8 (parallel connection, close operation)

Conditions

(1) Parallel connection of two pass elements (Figure 5.4).

(2) *PE3* turns on at $T_{ON}(PE3)$ value-dependently and *PE4* turns on at $T_{ON}(PE4)$ value-independently (Node *G* is value-dependent and Node *H* is value-independent).

Results

(1) The predecessor for the worst-case operation is Node *G* which is value-dependent.

(2) The path closes at $T_{ON}(PE3)$ in the worst case.

(3) Node *I* becomes value-dependent.

From **Worst-case operations 6 to 8**, the following observation is obtained.

[Observation 5.2] Worst-case close-operation of parallel connection

Consider two pass elements, connected in parallel as in Figure 5.4. Suppose that Node *F* is value-dependent. Then, the path closes value-dependently if, and only if, at least one pass element turns on value-dependently. The predecessor of the path for the worst-case close-operation is the gate of a pass element which turns on value-dependently, or the gate of the pass element which turns on first if both turn on value-dependently. The worst-case path-close time is the first time at which any pass element turns on value-dependently. □

5.2.3. COMPOSITE CONNECTION OF PASS ELEMENTS

Once the worst-case open/close-operation has been performed for two pass elements that are connected in series or in parallel, they can be represented by one super pass element for further worst-case operations. The worst-case operation predecessor of the path represents the gating node of the super pass element. Then, two pass elements that include a super pass element (or super pass elements) can again be modeled as one super pass element after the worst-case open/close-operation. Therefore, the worst-case operation predecessor and the worst-case open/close-times of a complex path can be obtained by modeling two pass elements as one super pass element repeatedly. Note that the repeated application of the super pass element justifies that **Observations 5.1** and **5.2** hold even when more than two pass elements are connected in parallel or in series.

5.3. SYSTEM MODELING AND TIMING VERIFICATION

Unlike other switch-level timing verifiers, *E-TV* deals with voltage waveform. In order to determine the transition time of a waveform, *E-TV* defines the logic threshold voltage of an inverter as follows:

[Definition 5.1] Logic threshold voltage of an inverter, V_{LT}

The logic threshold voltage of an inverter, V_{LT}, is defined as the input or the output voltage of an inverter when the input node and the output node have the same voltage. □

An inverter with its input and output tied together generates V_{LT}. The derivation of V_{LT} can be found in [25, 100]. Because V_{LT} depends on the transistor size of an inverter and the technology used, *E-TV* defines that a transition occurs when a waveform crosses a typical value of V_{LT}, specified by the user.

5.3.1. MODELING OF SYNCHRONOUS SYSTEMS

Figure 5.5 illustrates how *E-TV* models a synchronous system, using a 3-phase clocking system as an example. It should be noted that only the system inside the box with thick borderline is modeled by *E-TV*, and pulse and clock generators are excluded. Clock signals are said to be *primary*, if they are fed directly from a clock generator without passing through resistors or and-or-inverters of clock control circuit. Otherwise, they are called as *secondary* clock signals. While primary clock signals are the outputs of a clock generator, *E-TV* models them as ideal pulse power supplies. The internal resistance of the clock generator and the long wires of a layout can be modeled using resistors. In the figure, primary/secondary clock signals, *CLK1*, *CLK2*, and *CLK3*, and clocked storage elements *(CSE's)* are used for the global synchronization of the system. Any signals that are not used for global synchronization, while they are derived from a clock generator, are not treated as clock signals in *E-TV*. *E-TV* allows the designer to use both transparent and edge-triggered clocked storage elements in the same system. Notice that the figure illustrates *CLK1* logic evaluation section only, which consists of *CLK* 1–*CLK* 2 and *CLK* 1–*CLK* 3 logic segments. Once the primary clock signals are defined by the user, *E-TV* extracts all primary and secondary clock nodes, that are in turn used to identify clocked storage elements, logic segments, and logic evaluation sections. *E-TV* treats clock signals in a special way and they are value-dependent in propagation. Even though it is possible to regard clock signals as ordinary value-independent signals during a verification [23], treating clock signals specially in *E-TV* provides a significant advantage. This advantage is described later in this section. The timing constraints imposed by clock nodes and clocked storage elements have been described in detail in Chapter 2.

There is one other type of signal that is modeled by pulse power supplies: *logic-control (CTRL)* signals. The logic-control signal controls the signal flows through pass transistors in

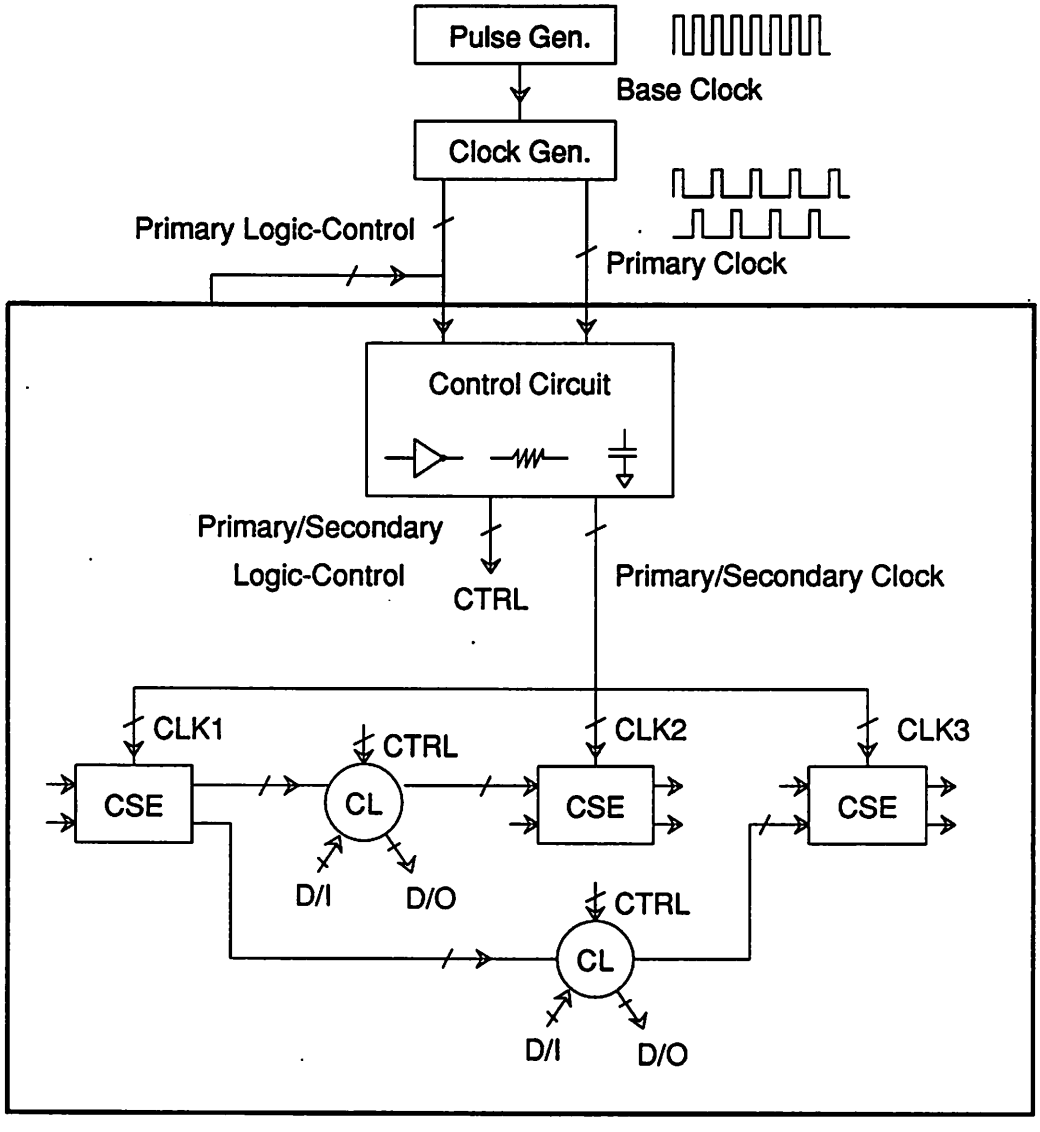


Figure 5.5 Modeling of The Synchronous System by *E-TV*
(3-Phase Clocking System)

combinational logic *(CL)* between pairs of clocked storage elements *(CSE's)*. Like clock signals, logic-control signals are value-dependent in propagation. The primary and secondary logic-control signals are defined in the same way. The logic-control signal imposes timing constraints; the output terminal of a pass transistor that is controlled by the logic-control signal must settle before the pass transistor turns off.

The logic-control signal is utilized for two purposes. First, the designer may use it to model a periodic control signal derived from a clock generator, that he/she does not want to view as a global synchronization clock signal. Second, it may be used to model a control signal that is generated in a design itself. In a typical design, there are many cases when a part of the design is controlled by a signal generated at another part of the design. Because such a control signal is value-independent, it does not impose any timing constraints, unless it is treated specially. The designer can use the logic-control signals to model such control signals. Even though the real control signals may not be periodic in general, they are specified in the form of pulse power supplies in order to represent their timing relationship to clock signals.

Data-in (D/I) nodes in Figure 5.5 are nodes that are driven from off-chip. They correspond to *Inputs* of Figure 2.1. *Data-out (D/O)* nodes drive off-chip and they correspond to *Outputs* of Figure 2.1. When verifying an incomplete design, *D/I* and *D/O* nodes are useful to model nodes that can be perceived as input/output nodes temporarily, but will be in a combinational logic block upon design completion. For each *D/I* node, the design objective of signal arrival time can be specified for timing verification with the logic evaluation section it belongs to. Like other ordinary signal nodes, D/I and D/O nodes are value-independent.

5.3.2. TIMING VERIFICATION OF SYNCHRONOUS SYSTEMS

E-TV performs a timing verification for each logic evaluation section, one by one. The timing verification is composed of two tasks: propagating signal transitions through a logic evaluation section and checking the signal timing against constraints. For the verification of a logic evaluation section, value-independent rising/falling transitions start to propagate from the preceding reference storage elements when they turn on. The value-independent transitions also start to propagate from *D/I* nodes at the times specified by the user. The propagation finishes when all signals arrive at the succeeding reference storage elements or *D/O* nodes. Then, the signal timings are compared to constraints. As described in Chapter 2, the propagation through clocked paths must finish by the succeeding time reference. Hence *E-TV* locates the worst delay path (longest delay path) leading to each succeeding clocked storage element, using the worst-case delay. Then, it computes the evaluation-time margin of the path, using the actual arriving clock signal at the succeeding clocked storage element. If the evaluation-time margin of any worst delay path is less than zero, *E-TV* reports that the path violates the maximum timing constraint imposed by clock signals. *E-TV* also reports the worst delay paths in each logic segment in the critical order using their evaluation-time margin. This provides the designer with the correct information necessary to improve a system speed. Note that the longest delay path is not necessarily the most critical path because of clock skew. If a combinational logic block contains dynamic circuits such as precharged/predischarged modules and N/P-type dynamic logic gates, *E-TV* checks if their associated timing constraints, described in Chapter 2, are satisfied during a signal propagation.

Consider a 3-phase clocking scheme, illustrated in Figure 5.6. For the sake of simplicity, *CTRL*, *D/I*, and *D/O* nodes are omitted. Assume that all clock nodes are primary and

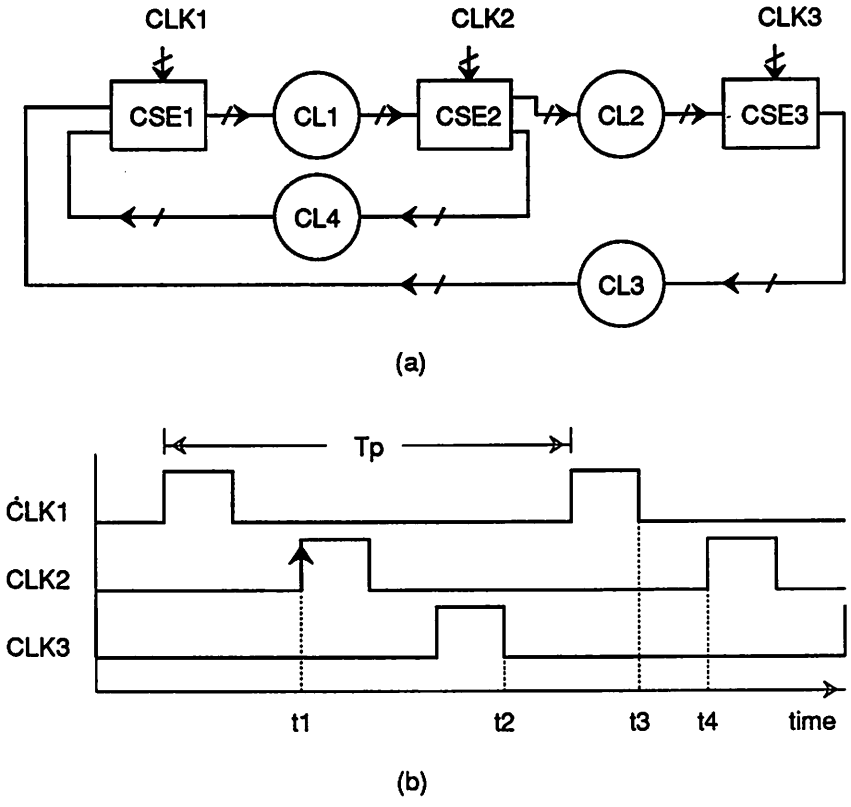


Figure 5.6 A 3-Phase Clocking Scheme
(a) System Model (b) Clock Signals

CSE1, *CSE2*, and *CSE3* are active-high transparent storage elements. There are three logic evaluation sections, in the figure: *CLK1*, *CLK2*, and *CLK3* sections. *CLK1* section consists of *CSE1*, *CL1*, and *CSE2*. *CLK2* section consists of *CSE2*, *CL2*, *CSE3*, *CL4*, and *CSE1*. *CLK3* section consists of *CSE3*, *CL3*, and *CSE1*. Suppose that *E-TV* verifies *CLK2* logic evaluation section first, then, *CLK3*, and *CLK1* sections. In order to verify *CLK2* section, *E-TV* must separate it from the remainder of the system. Thus, *E-TV* locates *CSE2* by finding all storage elements clocked by *CLK2*. Then, traversing from *CSE2* along the signal flow, *E-TV*

identifies combinational logic, *CL2* and *CL4*, and the succeeding reference storage elements, *CSE3* and *CSE1*. Next, *E-TV* visits nodes in *CLK2* section in the topological order. For each node, it extracts transistor chains, analyzes them using the ELogic delay model, and determines the worst delay path leading to it. In this way, *E-TV* propagates signals through the *CLK2* section from *CSE2*. Suppose that *t1* is chosen as the preceding time reference. Then, when the signal arrives at *CSE3* or *CSE1*, the arrival time is compared to *t2* or *t3* respectively to determine if there is any possible timing violation. After the verification of the *CLK2* section, *CLK3* and *CLK1* sections are examined in the same way. The procedures for the verification of synchronous systems are shown in Algorithm 5.1.

---

[1] Read in circuit and construct a data structure
[2] Find unspecified data-in and data-out nodes
[3] Find secondary clock/logic-control nodes and compute their waveforms
[4] Put all nodes in the system in the topological order
[5] Propagate fixed nodes that are set at logic "1" or "0" by the user
[6] Prune out a part of circuit that is not between primary inputs and outputs specified by the user
[7] Find all dynamic circuits using precharging and predischarging
[8] Find all clocked storage elements
[9] For each clock phase,
  [9.a] Identify a logic evaluation section by extracting clocked paths
  [9.b] Visit nodes in the section in the topological order
  [9.c] For each node,
    [9.c.1] Extract transistor chains and compute delays using the ELogic delay model
    [9.c.2] Locate the longest delay path leading to the node
  [9.d] Detect timing errors, using actual arriving clock/logic-control signals
  [9.e] List clocked paths in the critical order using the worst-case evaluation-time margin

Algorithm 5.1 Procedures for The Synchronous System Verification

---

The occurrence of clock edges is like a rolling wheel. There is no first occurring clock edge. Therefore, *E-TV* orders clock edges internally for convenience (the order of clock edges has been defined in Definition 2.7 in Chapter 2). For example, when *E-TV* verifies *CLK2* section of Figure 5.6, clock edges are arranged in the order of *CLK2(R) - CLK2(F) - CLK3(R) - CLK3(F) - CLK1(R) - CLK1(F).* This ordering of clock edges makes it easy for *E-TV* to detect timing violations, since all delay-limiting clock edges come after the preceding time reference, *CLK2(R).* An exception to this is a clock signal which controls the pre(dis)charging time of dynamic nodes. Consider a node *A* which precharges while *CLK1* is high. If Node *A* follows an NMOS pass transistor clocked by *CLK2,* it belongs to *CLK2* logic evaluation section. When *CLK2* section is verified, *CLK1(R)* and *CLK1(F)* are ordered after *CLK2(R).* However, the precharging of Node *A* is performed before *CLK2(R).* Therefore, the clock edges, *CLK1(R)* and *CLK1(F),* that limit the precharging of Node *A* occur before *CLK2(R).*

Some timing verifiers [23] treat clock signals in a different way; the user specifies whether the preceding clock signal is "rising" or "falling" (changing), while setting the succeeding clock signals at logic "1" or logic "0" (fixed) to separate the logic evaluation section under verification from the rest of a circuit. As an example, when *CLK1* logic evaluation section in Figure 5.6(a) is examined, the user sets *CLK1* at "rising" and *CLK2* at logic "0". As *CSE1* turns on with *CLK1(R),* an input signal begins to propagate through it. When the signal arrives at *CSE2,* the propagation stops, because the logic "0" signal turns *CSE2* off. Even though this approach is efficient, it has the following weaknesses. First, the approach may not separate the logic evaluation section for examination from the rest of the circuit in some cases. In Figure 5.6, *CLK1* section has only *CLK* 1–$\overline{CLK\,2}$ logic segment (all *CSE*'s are active-high). If *CSE2* includes both active-high and active-low transparent clocked storage

elements, the *CLK1* section would also have *CLK*1–*CLK*2 logic segment. In this case, the conditions for *CLK2* to turn off active-high and active-low transparent clocked storage elements conflict with each other. As a result, the *CLK1* section can not be separated successfully from the rest, until the user separates the clock nodes for two types of transparent clocked storage elements. This is a big burden on the user.

Another weakness would be that the approach has no clock information for the check of the timing constraints. Thus, it can not detect the possibility of timing errors. Instead, the approach reports the worst delay path leading to each succeeding clocked storage elements. It is the user's responsibility to find the critical paths by considering clock skew, and determine whether they satisfy the given set of timing constraints.

In addition, the user may not obtain the worst delay paths for each logic segment, because the approach does not distinguish between logic segments. Consider the case where the *CLK2* logic evaluation section of Figure 5.6 is being verified. The section consists of *CLK*2–$\overline{CLK3}$ and *CLK*2–$\overline{CLK1}$ logic segments. While the two logic segments have the same preceding time reference, they have different succeeding time references. If the preceding time reference for the two segments is *t1*, the succeeding time references for *CLK*2–$\overline{CLK3}$ and *CLK*2–$\overline{CLK1}$ segments are *t2* and *t3*, respectively. Since the *CLK*2–$\overline{CLK1}$ segment has more time for the signal propagation through *CL4* than the *CLK*2–$\overline{CLK3}$ segment does for *CL2*, the designer may assign larger path delays to *CL4*. In the extreme case, any path in *CL4* may have a larger delay than the worst delay of *CL2*. In this case, the approach can report the worst delay paths in *CL2* only after reporting all paths in *CL4*. Note that the number of worst delay paths or critical paths to be reported by a timing verifier is finite. Even though the number can be variable, the user may not know a sufficient number. However, a path *A* in *CL2* may be more critical than a path *B* in *CL4* due to the

shorter evaluation period available, even though it has a smaller delay. Therefore, even though it is more complex, the better approach is to treat clock signals in the way that *E-TV* does.

It has been mentioned that, for a *D/I* node, the logic evaluation section to which it belongs needs to be specified. This is described, using a circuit segment shown in Figure 5.7, where all *CSE's* are active-high transparent. Assume that a circuit inside the box with dashed borderline is to be verified. Then, Nodes *a* and *b* are described as *D/I* nodes. Let *CL1* and

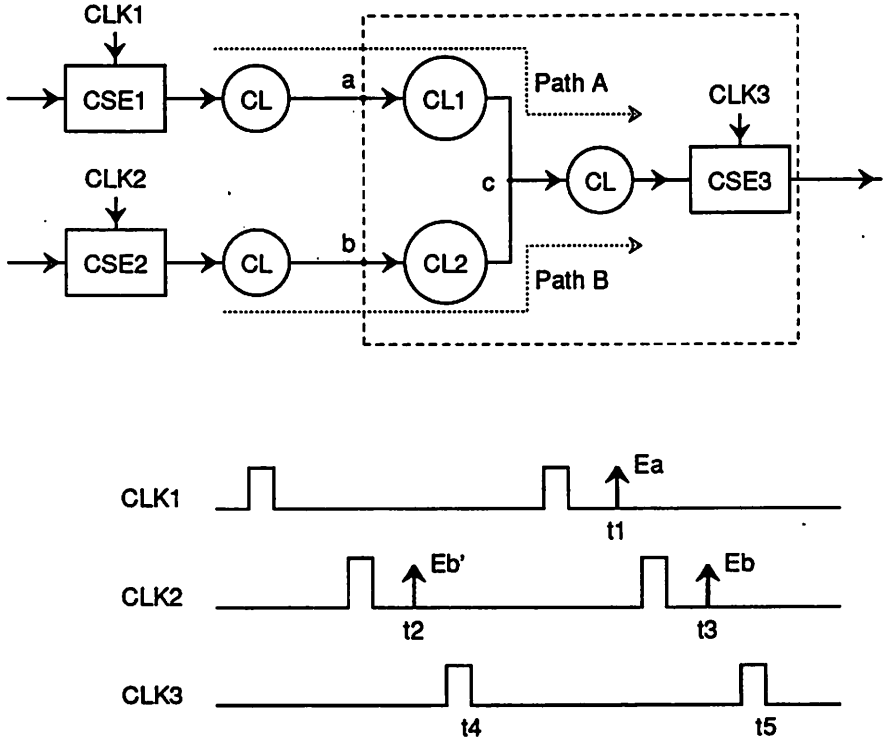


Figure 5.7 Clocked Paths with D/I Nodes

CL2 have the same propagation delay, *D*. Consider the case that the signal arrivals at Nodes *a* and *b* are represented as *Ea* and *Eb*, as shown on a timing chart, without specifying the logic sections they belong to. There are two paths to reach Node *c* from outside the box, Paths *A* and *B*. The signal arrival time at Node *c* through Path *A* is *(t1 + D)* and the one through Path *B* is *(t3 + D)*. Therefore, Path *B* is selected as the worst delay path leading to Node *c*. However, the user may specify that the signal arrives at Node *b* at *Eb'*, one clock period before *Eb*, because it is expected to arrive there once every clock period. In this case, the signal arrival time at Node *c* through Path *B* is *(t2 + D)* and Path *A* will be chosen as the worst delay path leading to Node *c*. This happens because the signal arrival times at *D/I* nodes have not been adjusted properly with respect to the succeeding time reference. As a result, the succeeding time references for two paths are not aligned on the timing chart; i.e., *t5* constrains Path *A*, while *t4* constrains Path *B*.

One of the solutions for this seems to be to request the user to specify the signal arrival times at *D/I* nodes in the way that the succeeding time references for all paths, that begin at the *D/I* nodes and end at the storage elements clocked by the same clock phase, can be the same. However, it is not only very time consuming but also impossible when paths starting from the same *D/I* node end at storage elements clocked by different clock phases. Similar problems may exist among clocked paths that begin from *D/I* nodes and that begin from the clocked storage elements inside the box. Therefore, *E-TV* asks the user to specify the logic evaluation sections that *D/I* nodes belong to so that it can handle all paths consistently whether they begin from *D/I* nodes or from clocked storage elements inside the box. When *E-TV* verifies *CLKi* logic evaluation section, it moves the signal arrival times at *D/I* nodes that belong to the section to within the first clock cycle after *CLKi* (*R*). When the associated logic evaluation sections are not specified, *E-TV* excludes paths from such *DI* nodes to the

first-encountered clocked storage elements from verification.

When *E-TV* verifies *CLKi* section, some MOS transistors inside the section may be controlled by nodes outside the section. *E-TV* regards such control signals as readily available before the verification of the section starts. *E-TV* assumes that they may be logic "0" or "1" fixed signals and hence both cases are examined.

5.3.3. MODELING AND TIMING VERIFICATION OF COMBINATIONAL LOGIC CIRCUITS

A synchronous system consists of clocked storage elements with combinational logic between them. Therefore, the timing verification of combinational logic circuits is not much different from that of the synchronous system. In fact, *E-TV* considers a combinational logic circuit as a circuit between clocked storage elements in a synchronous system. The model is seen in Figure 5.8. In the figure, *D/I*, *D/O*, and logic-control (*CTRL*) nodes are defined as those in the model of a synchronous system. The signal arrival times can be specified for each *D/I* nodes. Since no clock nodes exist in a combinational logic circuit, only logic-control nodes impose timing constraints. Unless otherwise specified, the same algorithms and approaches are used for the verification of the combinational logic circuit and the synchronous system.

In case of a combinational logic circuit, the value-independent rising/falling signals propagate from *D/I* nodes at the times specified by the user. If *CTRL* nodes are encountered during the signal propagation, their associated timing constraints are checked as for the synchronous system. When all signals arrive at *D/O* nodes, the verification is completed. Since there are no clock signals in a combinational circuit, *E-TV* locates the worst (longest) delay path leading to each *D/O* node from *D/I* nodes. Then, they are reported from the one with the

longest delay. The procedures for the verification of the combinational logic circuit are presented in Algorithm 5.2. Note that the algorithm is similar to Algorithm 5.1 for the verification of the synchronous system except that the procedures associated with clock signals are omitted. Therefore, only the algorithm for synchronous system verifications is discussed in this chapter.

5.4. DELAY COMPUTATIONS

During the propagation of rising and falling signals, *E-TV* computes nominal rising and falling transition times using a typical logic threshold voltage of an inverter, V_{LT}. These nominal transition times are used to detect the worst delay path (longest delay path) and timing errors in a system. *E-TV* stores the worst delay rising and falling waveform fragments for each node. These waveforms are used as input waveforms to the next transistor chains. Therefore, *E-TV* computes transition times and delays waveform-dependently.

The waveform-dependent computation has the following advantages. First, when the propagation delay through inverting stages is computed, the phase inversion is properly reflected. As an example, consider a chain of two inverters. Let T_U and T_D denote the rising and falling delays of the inverter. Then, the worst-case delay through the chain of two inverters is (T_U+T_D). However, suppose that $T_U > T_D$. If a delay is computed waveform-independently, the worst-case delay through each inverter is T_U and hence the worst-case delay through two inverters is $(2\times T_U)$. However, when the ELogic delay model is used, a rising input pulls the output of the first inverter to ground, which in turn pulls the output of the second inverter high. Or, a falling input pulls the output of the first inverter high, which in turn pulls the output of the second inverter low. Therefore, the correct worst-case delay (T_U+T_D) is obtained in any case.

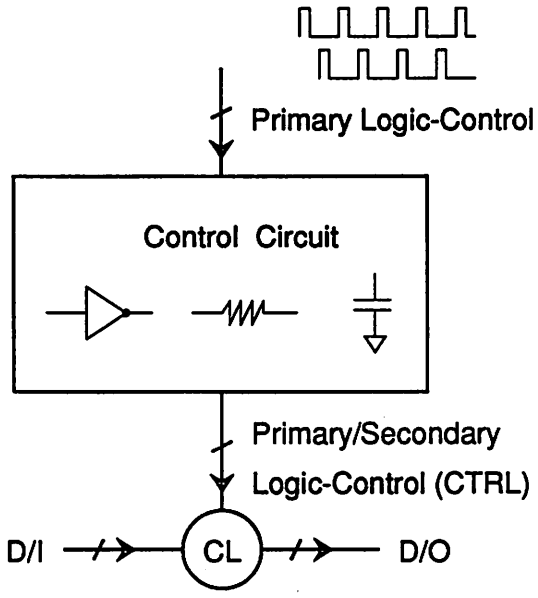


Figure 5.8 Modeling of The Combinational Logic Circuit by *E-TV*

[1] Read in circuit and construct a data structure
[2] Find unspecified data-in and data-out nodes
[3] Find secondary logic-control nodes and compute their waveforms
[4] Put all nodes in the system in the topological order
[5] Propagate fixed nodes that are set at logic "1" or "0" by the user
[6] Prune out a part of circuit that is not between primary inputs and outputs specified by the user
[7] Visit nodes in the topological order
[8] For each node,
  [8.a] Extract transistor chains and compute delays using the ELogic delay model
  [8.b] Locate the longest delay path leading to the node
[9] Detect timing errors, using actual arriving logic-control signals
[10] List the paths of [8.b] from the one with the longest delay, that end at *D/O* nodes.

Algorithm 5.2 Procedures for The Combinational Logic Verification

Another advantage of waveform-dependence is that realistic gate voltages of MOS transistors are used for delay computations. When the logic "1" signal of 5V passes through an NMOS pass transistor, it becomes a weak logic "1" signal; i.e., the output voltage will be about one threshold voltage below the gate voltage of the NMOS pass transistor. If NMOS transistors are controlled by the weak logic "1" rather than the strong logic "1", the propagation delays through them will be longer. Since *E-TV* uses a realistic waveform obtained from the preceding transistor chains, it provides realistic propagation delays. An additional advantage of waveform-dependence is that it naturally incorporates an input waveform shape and load conditions into the delay computation. Thus, delays can be computed accurately.

5.5. NODE ORDERING AND LOOP CUTTING

E-TV orders nodes in a given system topologically by carrying out a depth first search. Loops in the system are broken during the node ordering. The procedures are presented in Algorithm 5.3, using *pidgin* C. The key idea of the algorithm is as follows:

(1) During forward steps, if a node all of whose descendent nodes at any depth have been visited is met, it constitutes a parallel path. Such nodes are marked as *FINISHED* in Algorithm 5.3.

(2) During forward steps, if a node such that some of its descendent nodes have been visited, while some are not, is met, then there exists a loop. Such nodes are marked as *ACTIVE* in Algorithm 5.3.

(3) Order nodes in decreasing order as they are postvisited.

In the algorithm, *root* nodes are nodes whose in-degree is zero, such as ground, power supplies, and *D/I* nodes.

```
TopoOrder()                /* Topological ordering of nodes */
{
  Order = total number of nodes;

  foreach (npi ∈ {set of root nodes}) {
    Visit(npi);
  }
}


Visit(np)
{
  Mark np ACTIVE;

  foreach (fopi ∈ {set of child nodes of np }) {
    if ( fopi is FINISHED ) {        /* Parallel path */
    }
    else if ( fopi is ACTIVE ) {     /* Loop detected */
      /* NOTE 1 */
    }
    else {                           /* Not visited yet */
      Visit(fopi);
    }
  }

  Mark np FINISHED;
  Order = Order - 1;
  Order of np = Order;
}
```

Algorithm 5.3 Topological Node Ordering and Loop Breaking

In case of the synchronous system, the topological ordering is performed for the whole system before a verification begins and for each logic evaluation section during a verification. The topological order of nodes in the whole system is used for a simple switch-level simulation to propagate *fixed* nodes that the user set at logic "1" or "0" (this is discussed in Section 5.6). During a verification, the topological order of nodes in each logic evaluation section is

used to visit nodes systematically to locate the worst delay path.

As mentioned in Chapter 3, the static breaking of signal-flow loops blocks forward paths. Certainly, the number of blocked forward paths depends on the edge removed to break a loop. Therefore, it is important to choose an edge to remove intelligently. However, finding an optimal edge which blocks the minimum number of forward paths is NP complete. *E-TV* uses simple strategies to reduce the chance of blocking forward paths as follows:

(1) When a signal-flow loop is made of MOS transistor channels only and does not involve the gate of an MOS transistor, the loop is broken immediately.

(2) When a signal-flow loop involves the gate of an MOS transistor, *E-TV* traces back up to the gate of the most recently visited MOS transistor and cuts there.

This algorithm is performed at the place of *NOTE 1* in Algorithm 5.3. The above two cases are illustrated in Figures 5.9 and 5.10, where an arrow represents the signal flow through an MOS transistor. In Figure 5.9, a loop is made of MOS transistor channels only. Therefore, after traversing M1-M2-M3-M4-M5, *E-TV* breaks the loop at *A* as soon as it detects the loop. In Figure 5.10, a loop involves the gate of M6. After traversing M1-M2-M3-M4-M6-M7, if *E-TV* detects the loop at Node *A*, it traces back to the gate of M6 and breaks the loop at Node *C*. If the loop is broken at Node *A* rather than at Node *C*, all forward paths coming from Nodes *D* and *E* will be blocked. Even though the strategy is simple, it greatly reduces the chance of blocking forward paths.

5.6. CONSTANT NODE VALUES

Some timing verifiers such as Crystal [23] allow the user to set certain nodes at logic "0" or "1". Such nodes are called as *fixed nodes*. In addition, fixed nodes include nodes with a fixed voltage supply, ground, or nodes forced to be set by the other fixed nodes. When a

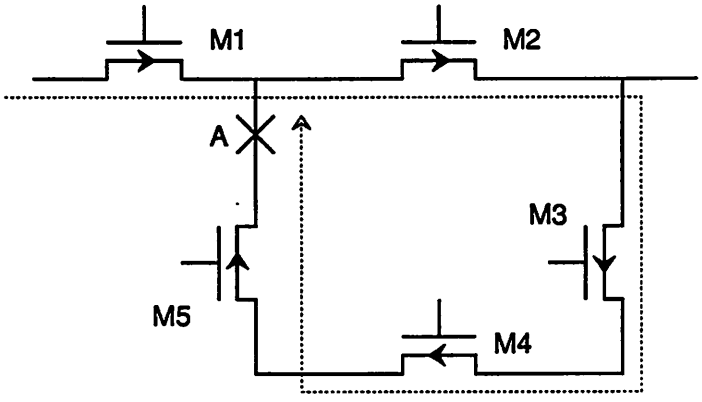


Figure 5.9 A Loop Made of MOS Transistor Channels Only

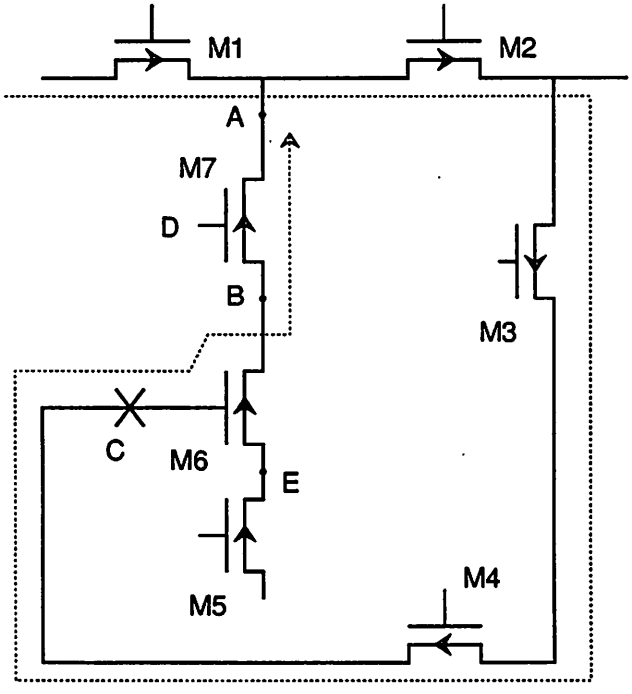


Figure 5.10 A Loop Involving The Gate of An MOS Transistor

node is set at a fixed value, *E-TV* considers that the node does not change its value during the timing verification. If a node is fixed in value, then any MOS transistor whose gate is attached to that node is forced to be either turned on or turned off, depending on the type of the transistor and the fixed value. If a transistor is forced off because its gate is fixed, then no paths involving that transistor are considered. As a result, paths passing through the transistor are excluded from timing verification. Therefore, while fixed nodes allow the user to eliminate unwanted paths from verification, they must be used with consideration. As mentioned previously, fixed nodes set other nodes at a fixed logic value. For example, if an input of a NOR gate is fixed at logic "1" value, then its output is fixed at logic "0" value. Thus, *E-TV* performs a simple switch-level simulation to check if already fixed nodes cause other nodes to also be fixed. The user may specify some transistors such as depletion NMOS loads are weak in their conductance. These transistors are weaker in strength than ordinary transistors in determining the logic state of nodes. When a node has two incoming paths with different logic signals, a path through a transistor with stronger conductance determines the logic state at the node. In the case that the conductance to logic "0" and logic "1" are the same, which should not happen in a properly described circuit, the node is not fixed. It should be noted that logic signals at fixed nodes propagate through transistors that are turned on by already fixed nodes.

E-TV propagates fixed nodes according to the topological ordering of nodes, because it indicates an order of tasks to be done without reprocessing a particular task. This simple switch-level simulation is very efficient, but powerful enough to propagate fixed nodes for the verification of a variety of NMOS and CMOS designs. The weakness of the current implementation is that it checks the conductance of local transistors only, rather than checking the effective conductance of the entire path back to the fixed voltage source node.

5.7. CLOCKED STORAGE ELEMENTS

Transparent clocked storage elements are classified into positive-active and negative-active types, while edge-triggered clocked storage elements are classified into positive edge-triggered and negative edge-triggered types. Their associated timing constraints on clocked paths with given clock signals have been described in Chapter 2. The clocked storage elements are used as the boundaries of clocked paths and for the derivation of timing constraints. Therefore, if not all clocked storage elements are found, the verification results will be incorrect. However, as mentioned in Chapter 3, the identification of clocked storage elements and their set-up time computations are some of the difficulties that switch-level timing verifiers have. Some existing switch-level timing verifiers implicitly assume that the clocked storage element in a MOS design is the pass transistor register type [23, 24]. For others, the discussion on the kinds of clocked storage elements allowed in a design and the way the set-up times are taken into account are not available in literature [56, 57]. This section describes how *E-TV* models both transparent and edge-triggered clocked storage elements. In the next section, it is described how *E-TV* identifies the clocked paths and checks their satisfaction of timing constraints with the set-up times and clock skew taken into account.

There are two possible approaches to identify clocked storage elements in a design:

(1) The user specifies the clocked storage elements used in the input file.

(2) The program identifies the clocked storage elements used from a circuit description.

The first approach has been used for block or higher level timing verifications. For example, block-level timing verifiers [22, 37] provide the models of clocked storage elements, that are similar to the models of other functional logic blocks such as logic gates. The user or the synthesis tool, when used in synthesis loops, is requested to specify the set-up time and the hold time of the elements. The models of clocked storage elements can be referenced to

describe a design.

While this approach is easy on a program, it is not practical to use at the switch level, because clocked storage elements usually consist of many transistors. It is not easy to describe which transistor belongs to which type of clocked storage element. In addition, suppose that the user obtained a transistor-level circuit description from a layout, using a circuit extractor [101, 102, 103, 104]. Then, it would be very time-consuming for the user to find all clocked storage elements and identify their type. Furthermore, the user may have to describe each clocked storage element individually due to different internal parasitic capacitances, rather than describing them in a fashion similar to referencing subcircuits for a circuit simulation. Thus, the second approach was chosen for *E-TV*.

In order to identify the clocked storage elements used, a program may explore:

(1) a transistor-level circuit description

or

(2) a higher level system description.

When the user designs a system in a top-down fashion within a hierarchical design environment, system descriptions that are higher than transistor level are available. These higher level descriptions may be inputs to synthesis tools or circuit designers. In any case, the higher level descriptions are more abstract functional descriptions. Thus, the higher level description is easier to explore than a transistor-level description to find clocked storage elements. However, this method is applicable only to the top-down hierarchical design methodology, where circuit elements are linked between different levels in some way, so that the information on the clocked storage elements can be transferred to transistor level properly. When a system is designed in a bottom-up fashion, or when only a transistor-level description has been obtained without a higher level description, the use of this method is out of the

question. Since this method is not robust enough for the general use, *E-TV* explores a transistor-level description for the identification of clocked storage elements.

There are two possible approaches for a program to find clocked storage elements and decide their type from a transistor-level description.

(1) Comparing the circuit topology of the elements.

(2) Approximating the element type.

The first approach works as follows. A program has a circuit topology library of all kinds of clocked storage elements available, along with other information such as their type. After finding a candidate transistor group for a clocked storage element, the program compares its topology to those stored in the library. If the circuit topology matches, the candidate is determined to be the clocked storage element with the matched topology. This *topology-matching* method is robust. A new design of clocked storage element can be easily added at any time. However, notice that some types of clocked storage elements, such as a CMOS edge-triggered D flip-flop, are made of as many as 28 MOS transistors. Therefore, this method consumes too much CPU time for topology comparison and is not practical.

The second approach works as follows. Any clocked storage element has at least one transistor clocked by a clock signal, which works as the core of the element. Therefore, it is possible to treat each clocked transistor as a pass transistor register. Consider the *CLK1* logic evaluation section shown in Figure 5.11. *M1* and *M2* are clocked transistors of the preceding and the succeeding clocked storage elements. For a verification, a signal starts to propagate from Node *A* at *CLK1(R)* as *M1* turns on. If *M2* is met during a traversal, it is checked whether the signal at *B* settles before *CLK2(F)* or not. This approach is used by the existing switch-level timing verifiers [23, 24]. Even though the method is very fast, it has disadvantage. While the method assumes that all clocked transistors are pass transistor registers, they

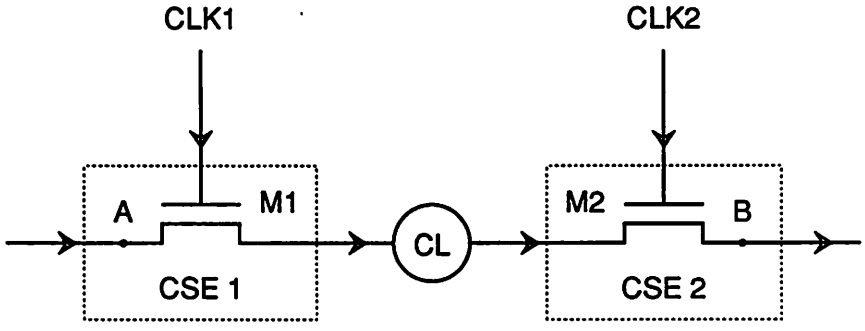


Figure 5.11 *CLK1* Logic Evaluation Section

may be the NMOS/PMOS drivers of inverting stages. Because a traversal is performed along the signal flow, there may be paths that do not pass through clocked transistors. Such an example is shown in Figure 5.12, where a signal propagation must finish while *CLK1* is high. The traversal along the signal flow is represented using a dotted line. Thus, some clocked paths may be excluded from examination.

Another disadvantage concerns the end points of clocked paths. Even though the end points of the logic section for verification must be the output nodes of the succeeding clocked storage elements, this method worries about the output terminals of clocked transistors. Timing verifiers employing this method will check whether the output terminals of clocked transistors settle by a specific timepoint, rather than checking if the outputs of the succeeding clocked storage elements settle. Thus, the verification results and the worst delay of the section are optimistic. Finally, the method does not distinguish the clocked pass transistors in dynamic logic gates such as domino logic or NORA logic from those in clocked storage elements. As a result, it has difficulty checking the satisfaction of the timing constraints associ-

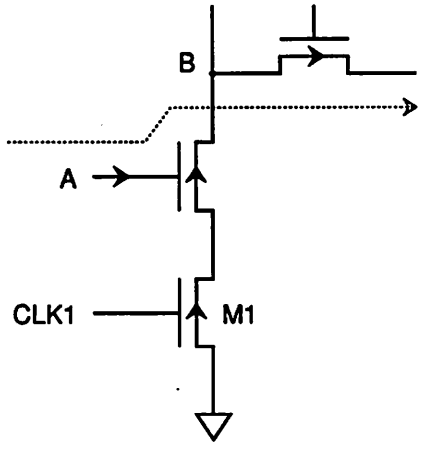


Figure 5.12 A Clocked Path Which Does Not Pass Through A Clocked Transistor

ated with the dynamic logic gates.

To overcome such a disadvantage, *E-TV* approximates clocked storage elements in a different way. In *E-TV*, two basic elements that compose clocked storage elements are defined: *transparent clocked elements* and *memory elements*. A transparent clocked element latches information during a certain clock phase and a memory element stores the latched information. There are two kinds of transparent clocked elements: *pass transistor registers* and *clocked inverting stages*. A pass transistor register merely provides a path for a signal to pass with the same polarity during a certain clock phase. A clocked inverting stage is an inverting stage whose drivers are controlled by clock signals. An example of a clocked inverting stage is a NAND gate with a clock signal applied to one of its input nodes. During a certain clock phase, the input signal of a clocked inverting stage is transferred to its output node with a reversed polarity. Clocked inverting stages eliminate difficulties that arise from

the fact that any clocked transistor is treated as a pass transistor register. Memory elements, the other element that composes a clocked storage element, may be a dynamic memory element as simple as a node capacitance. Or, they may be static memory elements such as cross-coupled inverters, whose feedback path may or may not be cut by a pass transistor (pass transistors), and RS latches. *E-TV* uses transparent clocked elements and memory elements as the beginning and ending boundaries of clocked paths to define a logic evaluation section. The satisfaction of timing constraints associated with these boundaries are examined.

E-TV assumes that a transparent clocked storage element consists of a transparent clocked element (or transparent clocked elements) followed by a memory element (or memory elements), where the number of transparent clocked elements and memory elements used depends on the kind of the clocked storage element. As an example, a transparent D flip-flop, shown in Figure 5.13, consists of two transparent clocked inverting stages followed by one memory element.

E-TV assumes that an edge-triggered clocked storage element comprises two levels of transparent clocked storage elements that are clocked by *CLK* and $\overline{CLK}$. Figure 5.14 illustrates a negative edge-triggered clocked storage element whose structure is represented as two levels of transparent storage elements. In the figure, *T_CSE* is a positive-active transparent clocked storage element such as a NMOS pass transistor register followed by an inverter. While *CLK* is high, the first *T_CSE* is active and the signal at Node *A* propagates to Node *B*. Let δ denote the time necessary for Node *B* to store a signal. Then, a signal at Node *A* during δ before *CLK(F)* will be stored at Node *B* at *CLK(F)*. Assume that an inverter shown to generate $\overline{CLK}$ is delay free. When the second *T_CSE* becomes active at *CLK(F)*, the signal at Node *B* propagates to Node *C*. Thus, two *T_CSE*'s work as a negative edge-triggered clocked storage element which samples an input signal at Node *A* available during δ

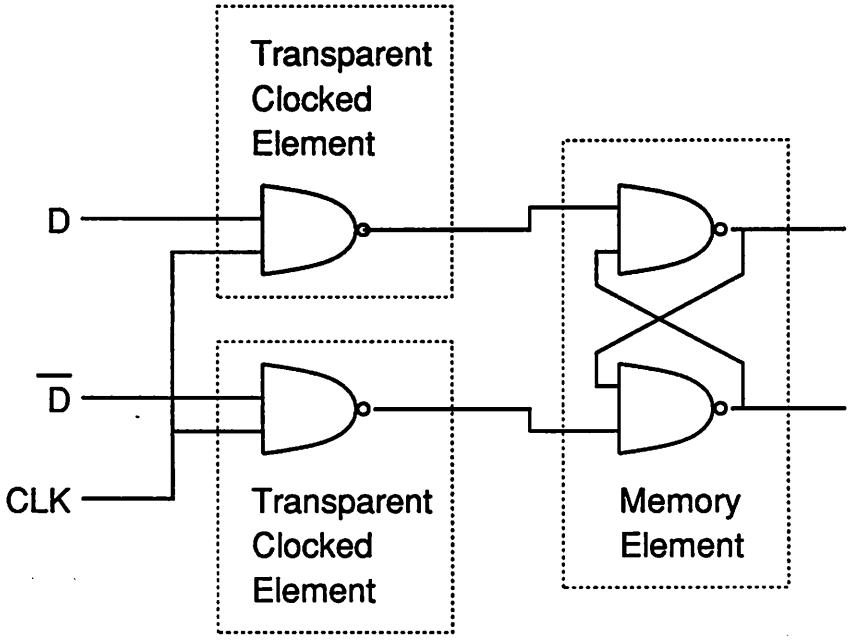


Figure 5.13 Transparent D Flip-Flop

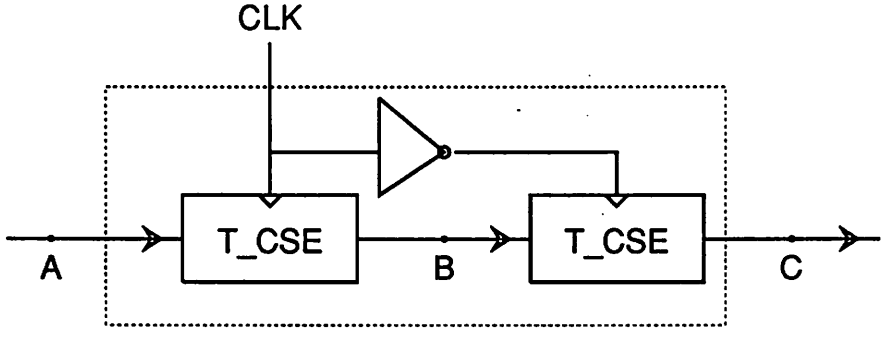


Figure 5.14 Structure of The Edge-Triggered Clocked Storage Element

before *CLK(F)* and changes an output signal at Node *C* at *CLK(F)*. To detect the possibility of timing errors associated with an edge-triggered element, *E-TV* applies timing constraints twice by looking at it as two cascaded transparent clocked storage elements. When examining a clocked path ending at Node *C* in Figure 5.14, *E-TV* examines two clocked paths, the one ending at *B* and the other from *A* to *C*. Because the second one is very short, it almost always satisfies the timing constraints.

5.8. TIMING VERIFICATION OF LOGIC EVALUATION SECTIONS

As defined in Chapter 2, the set-up and the hold times of clocked storage elements are sampling intervals that are located before and after an activating clock edge. They are useful concepts for block-level timing verification, where clocked storage elements are represented as black boxes. A timing verifier simply checks whether the input signals of clocked storage elements are kept unchanged during set-up and hold times, whose values are specified by the user [22, 37].

At the switch level, on the other hand, a timing verifier must compute the set-up and the hold times. The set-up and the hold times are attributed to the difference of the internal delays of input and clock signals. Thus, they can be computed by locating the paths for those two signals in a clocked storage element. However, a better approach is to compare the clock arrival and the input signal settling times at appropriate points inside a clocked storage element directly. This can be described with the help of Figure 5.15. In the figure, CSE1 and CSE2 are transparent type consisting of an NMOS pass transistor register, clocked through two inverters, and a capacitance. If the block-level approach is employed, a program computes the set-up times of CSE1 and CSE2 and a delay through *CL*, D_{CL}. Then, Equation (2.2) is used to detect a possible timing error. However, a timing verifier can detect the pos-

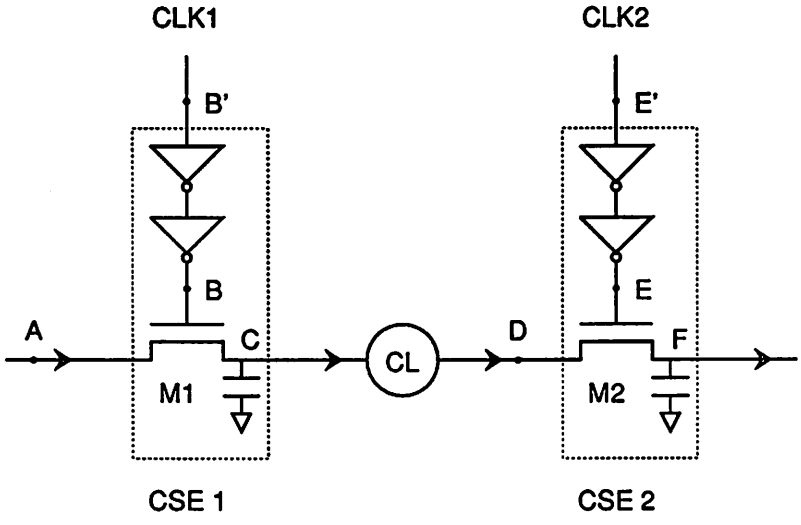


Figure 5.15 *CLK1* Logic Evaluation Section

sible timing errors without involving the set-up and the hold times by directly comparing the transitions at the clock node of pass transistor register and its output node as follows. Transistor *M1* turns on, when a clock signal at an internal node *B* rather than *B'* goes high. At the same time, a signal starts to propagate from Node *A* through *C*, *CL* and *M2*, to Node *F*. During this propagation, *M2* is forced to remain on. Then, the transition at Node *F* is compared to the falling transition at an internal clock node *E* rather than *E'*, at which *M2* actually turns off. Notice that this *direct-comparison* approach is more convenient and less error-prone than the block-level approach for switch-level timing verification. The advantage of the direct-comparison approach includes the fact that the influence of clock skew is automatically taken into account for verification. Since the direct-comparison method uses the actual arriving clock signals at clocked transparent elements, it is not concerned about clock skew.

Due to the above reasons, *E-TV* uses the direct-comparison approach for the detection of possible timing errors.

E-TV defines three core nodes in a clocked storage element for the direct-comparison approach: *Clock_Node*, *Start_Node*, and *End_Node*. *Clock_Node* is a clock node of a transparent clocked element. It provides two timepoints: a preceding time reference (t_{PR}) at which a signal starts to propagate and a succeeding time reference (t_{SR}) by which the signal propagation must finish. *Start_Node* is a node which defines the starting point of clocked paths, while *End_Node* is a node which corresponds to the end point of clocked paths. Thus, *Start_Node* and *End_Node* are the preceding and succeeding space references (s_{PR} and s_{SR}), respectively. Typically, *Start_Node* is the output node of a transparent clocked element and *End_Node* is the output node of a memory element. In *E-TV*, a clocked path is defined as a path that begins from the *Start_Node* of a preceding clocked storage element and ends at the *End_Node* of a succeeding clocked storage element. For the clocked path illustrated in Figure 5.15, *Clock_Node*'s are Nodes *B* and *E*. *Start_Node* is Node *C* and *End_Node* is Node *F*. The direct comparison of transition times are made between *Clock_Node* and *End_Node*.

The idea of *E-TV* for the verification of the synchronous system can be summarized as follows:

(1) Edge-triggered clocked storage elements are assumed to be two levels of transparent clocked storage elements, as shown in Figure 5.14.

(2) Transparent clocked storage elements are viewed as a transparent clocked element (or transparent clocked elements) followed by a memory element (or memory elements), as shown in Figure 5.13.

(3) Clocked paths are defined as paths between the *Start_Node*'s of preceding clocked storage elements and the *End_Node*'s of succeeding clocked storage elements.

(4) Signals start to propagate from *Start_Node*'s, when the transparent clocked elements of preceding clocked storage elements turn on (become active).

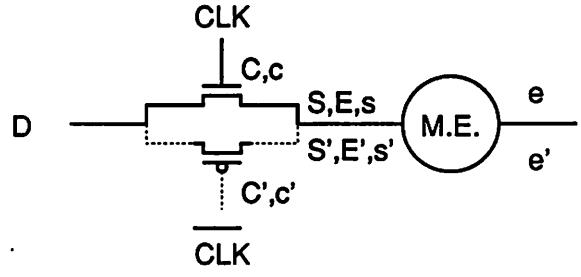
(5) During a signal propagation, the transparent clocked elements of succeeding clocked storage element must remain active (the *clocking delay* of the succeeding clocked storage element must be zero).

(6) The possible timing errors are detected by the direct-comparison of transition times at the *Clock_Node* and *End_Node* of succeeding clocked storage elements. If Equation (5.1) is satisfied, *E-TV* determines that the clocked path satisfies timing constraints.

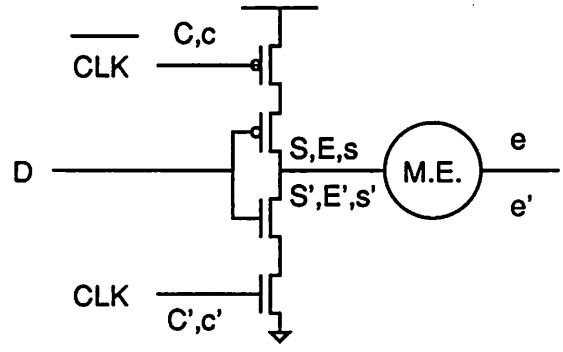
$$t_{End\_Node} \leq t_{Clock\_Node} \tag{5.1}$$

where t_{End_Node} is a transition time at *End_Node* of succeeding clocked storage element and t_{Clock_Node} is a timepoint at which the transparent clocked element of the succeeding clocked storage element becomes inactive.

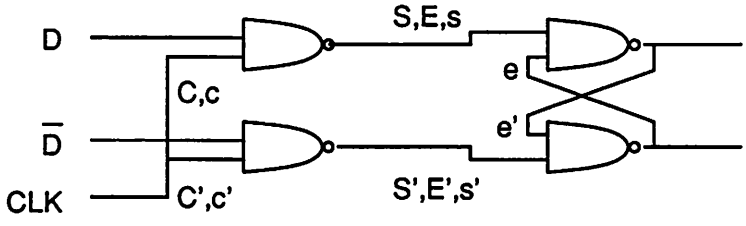
In the case that a transparent clocked element is followed by a static memory element, *End_Node* is usually the output node of the static memory element after passing through a feedback loop. However, the speed of signal propagation through the memory element is very fast. Thus, *E-TV* approximates the output terminal of transparent clocked element as *End_Node* (i.e., *Start_Node* and *End_Node* are the same). *E-TV* finds these three kinds of nodes by propagating clock signals and identifying whether clocked transistors are pass transistor registers or they are part of clocked inverting stages. Figures 5.16 and 5.17 illustrate the true *Clock_Node*, *Start_Node*, and *End_Node* of clocked storage elements that are commonly used in MOS VLSI designs and their approximations by *E-TV*. In both figures, an *M.E.* in a circle represents a dynamic or static memory element. The true *Clock_Node*, *Start_Node*, and *End_Node* are denoted as *c*, *s*, and *e*, while *E-TV*'s approximations are



(a)

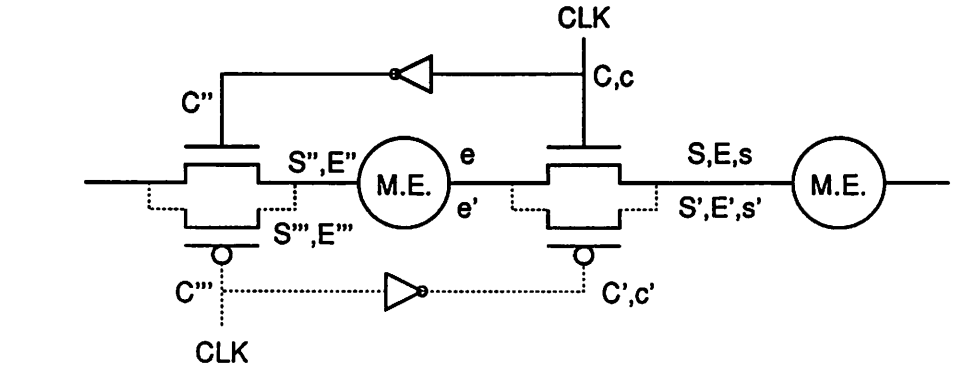


(b)

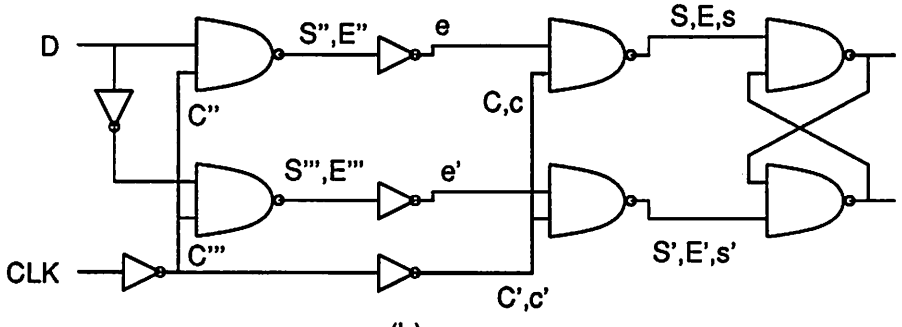


(c)

Figure 5.16 Transparent Clocked Storage Elements
(a) Pass Transistor Register (b) C^2MOS Buffer
(c) Transparent D Flip-Flop



(a)



(b)

Figure 5.17 Edge-Triggered Clocked Storage Elements
(a) Pass Transistor Type (b) Edge-Triggered D Flip-Flop

denoted as *C*, *S*, and *E*. When there are a number of sets of *Clock_Nodes*, *Start_Nodes*, and *End_Nodes*, they are represented by adding a single quotation mark (e.g., *C'*, *C''*, *C'''*, $\cdots$). In Figure 5.17, *c*, *s*, and *e* are obtained when the edge-triggered clocked storage element is treated as one element, while *C*, *S*, and *E* are obtained by partitioning it into two transparent clocked storage elements.

A node can serve as the *End_Node* of more than one clocked path as illustrated in Figure 5.18(a) and (b), where a circle represents a combinational logic. A node can serve as the *Start_Node* of more than one clocked path as in Figure 5.18(c). Or, combinational logic can be used more than once per clock cycle. The logic path illustrated in Figure 5.18(d) is used twice per clock cycle, for the periods of $CLK\,1$–$\overline{CLK\,2}$ and $CLK\,3$–$\overline{CLK\,4}$.

E-TV finds each logic evaluation section by traversing a system along signal flows from the *Start_Node*'s of clocked storage elements whose transparent clocked elements are active to the *End_Node*'s of clocked storage elements whose transparent elements are inactive during the given clock phase. When clocked storage elements, whose transparent elements are active during the clock phase, are encountered during traversing, *E-TV* continues the traversal.

5.9. WORST DELAY PATH AND PREDECESSOR

It is interesting to investigate whether the worst (longest) delay path and the responsible parent or predecessor can be found for a given transistor block by one simulation. There are two cases to consider:

(1) There are multiple paths which lead to the output node of a transistor block. The worst delay path of the output node and the worst delay parent is sought among the multiple paths and the source/drain terminals of the last pass transistors on the paths, respectively.

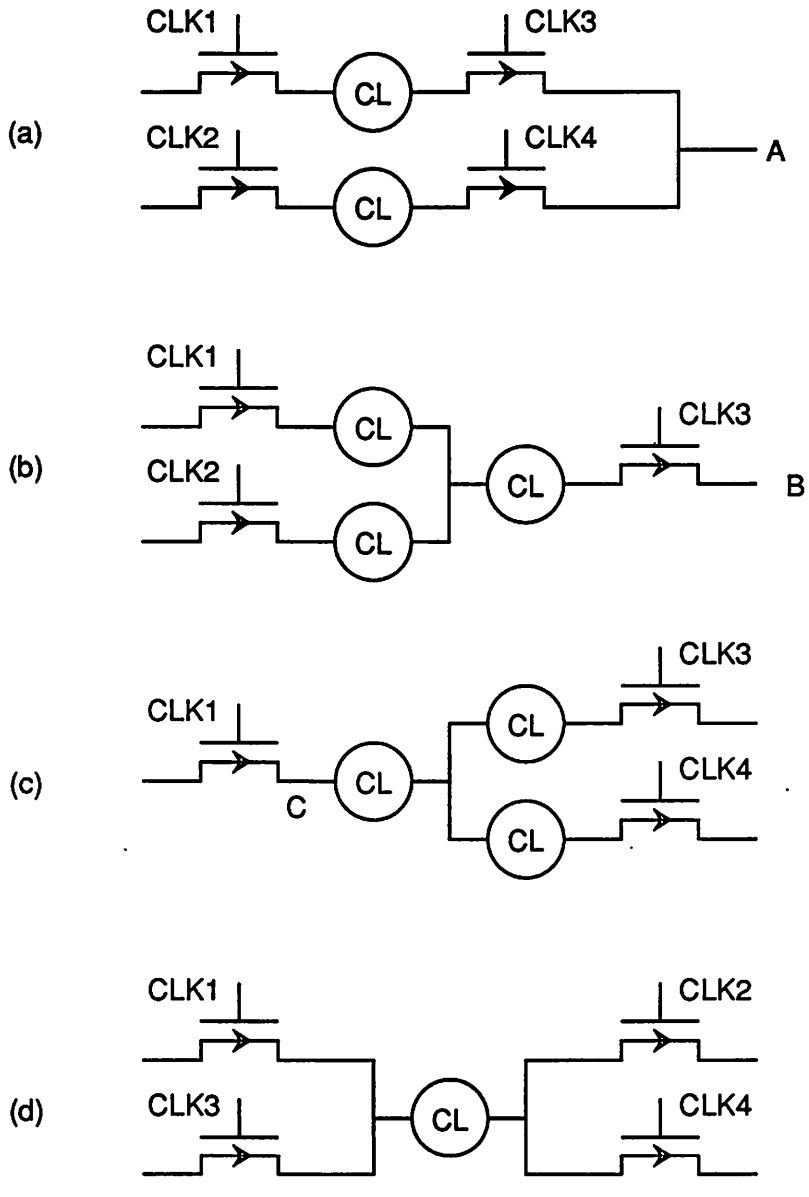


Figure 5.18 Examples of Clocked Paths

(2) There is a single path which leads to the output node of a transistor block. The worst delay parent is sought between the gate and the source/drain terminals of the last pass transistor on the path.

As an illustration of the first case, consider a circuit fragment shown in Figure 5.19(a), where four MOS transistors turn on value-independently. There are two paths leading to Node *Output* from Node *Input*. The worst rising delay parent of Node *Output* is sought

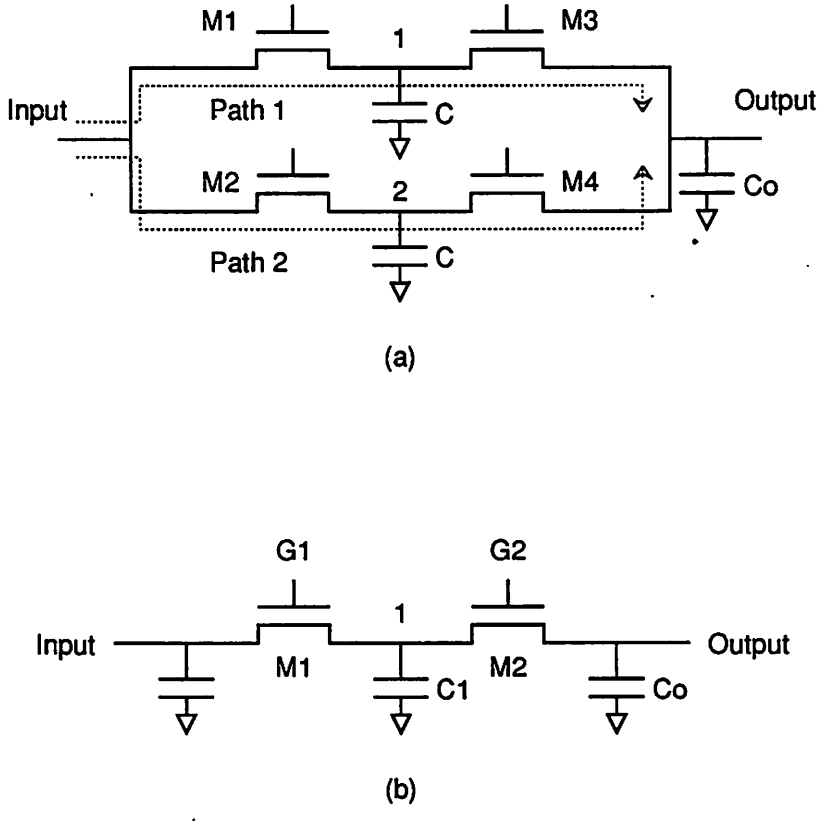


Figure 5.19 Examples of Locating The Worst Delay Path and Parent

between its two fanin nodes, *1* and *2*. A simulation can be performed by applying a rising signal to Node *Input* and a power supply *Vdd* to the gates of four MOS transistors. Let $T_D(1)$ and $T_D(2)$ denote delays from Node *Input* to Nodes *1* and *2*, obtained by the simulation, respectively. It can be queried: can Node *1* be determined as the worst delay parent of Node *Output*, if $T_D(1) > T_D(2)$? Assume that *M1* and *M2* have the same W/L ratio, *M3* has a very large W/L ratio, and *M4* has a very small W/L ratio. Since *M3* couples Nodes *1* and *Output* very strongly, the effective load capacitance at Node *1* is (*C+Co*). On the other hand, the effective load capacitance at Node *2* remains the same as *C*, because *M4* has a very small W/L ratio. Therefore, a path from Node *Input* to Node *1* will have a larger delay than a path from Node *Input* to Node *2*; i.e., $T_D(1) > T_D(2)$. However, a path from Node *1* to Node *Output* will have a much smaller delay than a path from Node *2* to Node *Output*. As a result, even though $T_D(1)$ is larger than $T_D(2)$, the lower path may be the worst delay path leading to Node *Output*, depending on the W/L ratios of *M3* and *M4*, and Node *2* may be the true worst delay parent of Node *Output*. This observation indicates that the correct worst delay parent of Node *Output* can be determined only by simulating both paths leading to Node *Output* individually and comparing their path delays. During the simulation of the upper path, *M2*, *C2* and *M4* must be excluded. Similarly, when the lower path is simulated, *M1*, *C1* and *M3* must be excluded.

For the second case, consider a circuit fragment shown in Figure 5.19(b), where *M1* turns on value-independently. Assume that the rising signals at Nodes *Input* and *G2* are given. It is going to be determined which node is the worst rising delay parent of Node *Output*, Node *1* or *G2*. A simulation can be performed by applying rising signals at Nodes *Input* and *G2*, and a power supply *Vdd* to Node *G1*. Let the rising transition times at Nodes *G2* and *1* be denoted as $t_{G2}(R)$ and $t_1(R)$. If $t_{G2}(R) > t_1(R)$, *G2* is the worst rising delay parent of

Node *Output*. However, if $t_{G2}(R) < t_1(R)$, Node *1* cannot be safely determined as the worst delay parent of Node *Output*, because $t_1(R)$ is affected by *Co* as well as *C1*. Thus, the worst delay parent of Node *Output* can be determined only by simulating a path consisting of *M1* and *C1* and by comparing the transition time at Node *1* to $t_{G2}(R)$.

The above two observations reveal that, unfortunately, the worst delay path and the worst delay parent (predecessor) of the output node of a given transistor block can not be found by one simulation, in general.

5.10. DETECTION OF THE WORST DELAY PATH AND PREDECESSOR

As investigated in the previous section, when a node has more than one candidate path for the worst delay path, delays through the candidate paths must be compared after simulating each path separately. Similarly, when a node has more than one candidate node for the worst delay parent, transition times at the candidate nodes must be compared after simulating each path leading to the candidate node. In this section, it is described how *E-TV* locates the worst delay path and the worst delay predecessor of a node in a circuit.

5.10.1. TRANSISTOR CHAINS

In order to determine the worst delay path of each node, *E-TV* extracts linear chains of transistors leading to the node. Then, it simulates each transistor chain using the ELogic delay model. When *E-TV* extracts a transistor chain, it always traces back up to a voltage source or ground. Even though this approach consumes more CPU time than tracing back by a fixed number of nodes or to another non voltage-source node, computed delay values are more accurate. However, long transistor chains are rarely used in practice. In fact, they are more likely the results of the erroneous assignment of signal flow. Thus, *E-TV* excludes a

transistor chain from consideration, if it has more transistors in series than specified by the user.

Each processed node stores two kinds of information for the extraction and the simulation of the next transistor chains: *information parent* and worst delay parent for rising and falling transitions. An information parent is the source/drain terminal of the last MOS transistor on the worst delay path towards some strong information source (power supplies or ground). Information parents are used to trace back to extract transistor chains for delay evaluation. The worst delay parent may be the same as an information parent or it may be the gate of the last transistor on the worst delay path. The worst delay parent is used to find the *outside worst delay predecessor* of a transistor chain, to which *E-TV* applies the worst delay waveform stored at the node to evaluate the path delay through the transistor chain. The outside worst delay predecessor of a transistor chain is the first worst delay parent which is not the source/drain terminal but the gate of the MOS transistor in the chain during a backward trace.

How *E-TV* extracts a transistor chain and how it determines the worst delay path of a node can be illustrated using Figure 5.20. In the figure, the information parent and the worst delay parent of processed nodes are represented using dotted lines with letters of *I* and *W*, respectively. Suppose that the worst falling delay path of Node *h* is sought. Transistor chains are obtained by tracing back along information parents, beginning from each MOS transistor through which a signal enters *h*. Thus, two transistor chains will be extracted. By tracing back from *M5*, the first transistor chain, *M5*, *M4*, *M2*, and *M1*, is obtained. Similarly, another transistor chain of *M10*, *M8*, and *M6* is obtained by tracing back from *M10*. As an example of finding the outside worst delay predecessor, consider the first transistor chain containing *M5*. Because the node between *d* and *g* with the worst delay has not been decided yet,

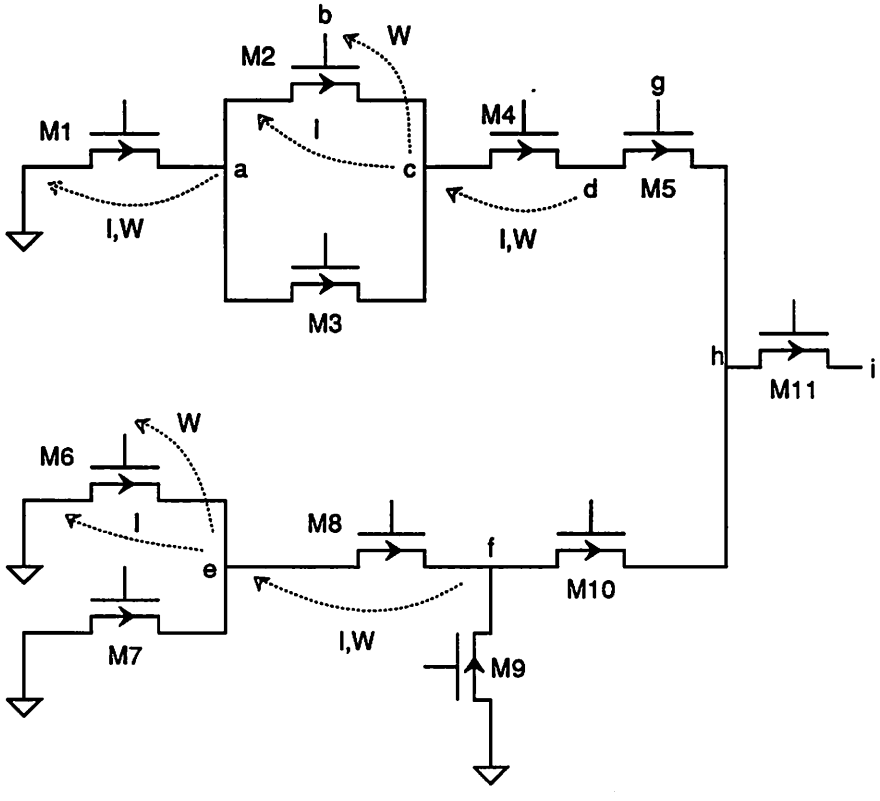


Figure 5.20 An Example of Locating The Worst Delay Path
(Grounded capacitors are omitted for convenience)

the signal arrival times at Nodes *d* and *g* are compared first. If a signal arrives at Node *g* later, the outside worst delay predecessor of the transistor chain is Node *g*. Or, if a signal arrives at Node *d* later, *E-TV* traces back along the transistor chain to find the first worst delay parent which is not on the path. Therefore, Node *b* becomes the outside worst delay predecessor. After the outside worst delay predecessor is determined, *E-TV* assigns an initial condition at each node on the path in such a way that only the nodes that come after the outside worst delay predecessor make a falling transition. It can be illustrated by supposing that

Node *b* is the outside worst delay predecessor of the transistor chain. Then, Node *a* is initialized to 0V and Nodes *c*, *d*, and *h* are initialized to 5V. During a delay evaluation, *E-TV* applies a power supply *Vdd* to the gates of all transistors except Node *b* where *E-TV* applies the worst delay waveform stored at the node. After computing delays through both transistor chains, *E-TV* determines the worst delay path leading to Node *h* by comparing their path delays. Suppose that the first transistor chain is the worst delay path of Node *h*. Then, Node *d* becomes the information parent of Node *h*. If Node *g* is the outside worst delay predecessor of the transistor chain, Node *g* becomes the worst delay parent of *h*. Otherwise, Node *d* becomes the worst delay parent.

The worst delay parents are also used to trace back to find the worst delay path from preceding clocked storage elements or data-in nodes. As an example, the worst delay path of Node *h* will be $\cdots$–*b*–*c*–*d*–*h*, if *d* is the worst delay parent of Node *h*.

5.10.2. CMOS TRANSMISSION GATES

It is known that NMOS pass transistors transfer the logic "0" signal quite well but they transfer the logic "1" signal poorly. On the other hand, PMOS pass transistors transfer the logic "1" signal well but they transfer the logic "0" signal poorly. Thus, in a CMOS design, the pair of NMOS and PMOS pass transistors is used as a transmission gate, where both logic "0" and "1" signals need to be transmitted. If the pair of NMOS and PMOS pass transistors are not treated as one transmission gate, the verification results will be very pessimistic. The reason for this is that when a path is examined for the logic "0", PMOS transistors are chosen to compose the worst delay path. On the other hand, NMOS transistors are chosen to compose the worst delay path for the logic "1" signal. *E-TV* treats the pair of NMOS and PMOS pass transistors as one transmission gate. By doing so, the accuracy of verification results is

significantly improved. For this, when *E-TV* traces back to extract transistor chains, it finds NMOS and PMOS pass transistors that are in parallel between two nodes and controlled by complementary signals.

An NMOS or a PMOS pass transistor has only two candidate nodes for the worst delay parent, gate and source/drain terminals. However, as shown in Figure 5.21, a CMOS transmission gate has three candidate nodes, *A*, *B*, and C, for the worst delay parent of *D*. Let *tran(A)* denote the transition time at Node *A* without the loading effect of Node *D*. Let *tran(B)* and *tran(C)* denote the times at which *M1* and *M2* turn on, respectively. Then, there are six cases to consider:

(1) Case 1 : $tran(B) < tran(C) < tran(A)$
(2) Case 2 : $tran(C) < tran(B) < tran(A)$
(3) Case 3 : $tran(A) < tran(B) < tran(C)$
(4) Case 4 : $tran(A) < tran(C) < tran(B)$

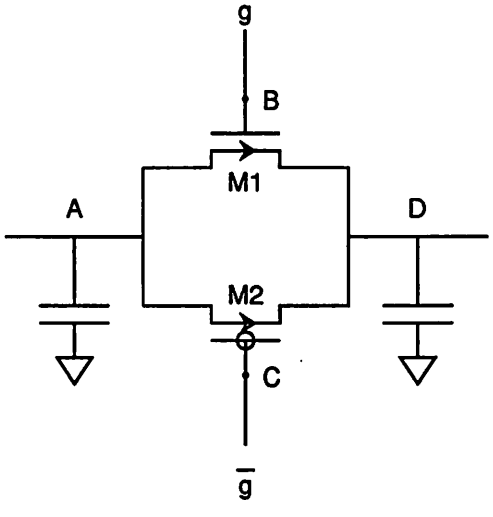


Figure 5.21 CMOS Transmission Gate

(5) Case 5 : *tran(B) < tran(A) < tran(C)*
(6) Case 6 : *tran(C) < tran(A) < tran(B)*

In the first two cases, an input signal arrives at Node *A* after both *M1* and *M2* turn on. Thus, Node *A* is the worst delay parent of Node *D*. In Cases 3 and 4, an input signal arrives at Node *A* before *M1* or *M2* turns on. Thus, one of Nodes *B* and *C*, whichever turns on later, is the worst delay parent of Node *D*. In the last two cases, when an input signal arrives at Node *A*, one pass transistor is on and the other is still off. The appropriate worst delay parent in this case is the node which keeps its pass transistor still off: Node *C* in Case 5 and Node *B* in Case 6. Therefore, *E-TV* determines a candidate node with the latest signal arrival time as the worst delay parent.

When one of two gate terminals of a CMOS transmission gate is determined to be the outside worst delay predecessor of a transistor chain, *E-TV* applies the worst delay waveform stored at each gate terminal for delay evaluation. Even though the waveforms might have been computed value-independently, they are usually related closely through one or a few inverters. The circuit condition which causes the worst delay at one gate terminal is most likely to cause the worst delay at the other gate terminal. Therefore, it is appropriate to use the worst delay waveform stored at each gate terminal. When a CMOS transmission gate is used as a succeeding clocked storage element, *E-TV* determines that a timing constraint is violated if the signal at *End_Node* does not settle before both pass transistors turn off.

5.10.3. AND-OR-INVERTERS

The block diagram of an and-or-inverter is illustrated in Figure 5.22. Both logic "0" and "1" information sources are reachable from Node *out* through separate paths. In order to determine the worst rising delay predecessor of *out* from Nodes *inputs 1* and *inputs 2*, the following two cases need be considered:

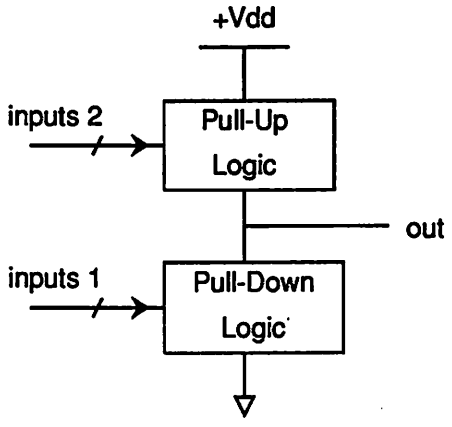


Figure 5.22 Block Diagram of An And-Or-Inverter

(1). Case 1: Pull-down path opens first. Then, pull-up path closes.

(2) Case 2: Pull-up path closes first. Then, pull-down path opens.

In Case 1, pull-down paths are already open, when a pull up begins. Thus, the worst rising delay predecessor of Node *out* is one of *inputs 2*, which closes a pull-up path last. In Case 2, even though pull-up paths are on, Node *out* cannot be pulled up fully until all pull-down paths become open. Thus, the worst rising delay predecessor is one of *inputs 1*, which opens a pull-down path last. However, it should be noted that, in this case, there is a time interval for which both pull-up and pull-down paths are on. During this interval, a direct path is established for current to flow from a power supply to ground. Thus, it is reasonable to assume that this time interval is short enough in practical designs so that an input which closes a pull-up path last can be regarded as the worst rising delay predecessor. Therefore, in any case, *E-TV* determines an input which closes a pull-up path last as the worst rising delay path of Node *out*. Similarly, an input which closes a pull-down path last is determined as the

worst falling delay path of Node *out*.

When an NMOS depletion load or an "always-on" weak PMOS transistor is used in the pull-up logic of an and-or-inverter, there are no external inputs to the pull-up logic. In this case, when all pull-down paths become open, the node voltage at *out* begins to rise up. Thus, *E-TV* determines an input node which opens a pull-down path last as the worst rising delay predecessor of Node *out*. The way to determine the predecessor of *out* for the worst-case open-operation of pull-down logic has been described in Section 5.2. The rising transition timepoint of Node *out* is computed by adding a pull-up delay to the timepoint at which the worst delay predecessor opens the pull-down path.

5.11. TIMING VERIFICATION OF DYNAMIC CIRCUITS

Precharging is an important technique for the design of combinational logic between clocked storage element pairs in MOS synchronous systems. Dynamic circuits that employ the precharging technique are precharged/predischarged modules, domino logic [31], and NORA logic [32]. Because these circuits provide the significant advantage of faster circuit operation, area reduction, and power consumption reduction in many applications, they are quite common in high performance MOS VLSI designs [98, 105]. However, the dynamic circuits have tight timing constraints to satisfy, since they make the use of the dynamic charge storage at nodes. The satisfaction of these timing constraints is difficult to examine properly at other than switch level. In spite of their importance, unfortunately, the verification of the dynamic circuits have been neglected by the existing switch-level timing verifiers.

In order to verify dynamic circuits, *E-TV* identifies them before a timing verification begins. Then, when they are encountered during the signal propagation to locate the critical paths, *E-TV* checks whether or not they satisfy timing constraints. The timing constraints

associated with dynamic circuits have been described in Chapter 2 in detail. Dynamic circuits with precharging/discharging have precharged or predischarged nodes. Thus, *E-TV* locates them by finding all precharged and predischarged nodes that can be identified by checking the following. A node *A* which precharges while ϕ is low and evaluates while ϕ is high satisfies the following two conditions:

(1) Precharging paths (pull-up paths):

While ϕ is low, at least one pull-up path between Node *A* and a power supply must be closed to establish a precharging path. While ϕ is high, all pull-up paths must be open.

(2) Evaluation paths (pull-down paths):

While ϕ is low, all pull-down paths between Node *A* and ground must be open. While ϕ is high, clocked transistors on the pull-down paths, that turn off while ϕ is low, must turn on to establish evaluation paths.

Predischarged nodes can be found by exchanging the conditions for pull-up and pull-down paths. After finding all precharged and predischarged nodes, *E-TV* determines the type of their associated dynamic circuits: precharged/predischarged modules or N/P-type dynamic logic gates used in domino and NORA logic. This information is used to examine the satisfaction of timing constraints that are described in Section 2.8.

Consider a rising transition at a precharged node. As long as the timing constraints on input signals are satisfied, the precharged logic "1" signal at the node remains undisturbed. That is, a logic "1" signal is available as soon as a precharging is done. Thus, only a precharging path is meaningful for the rising transition at a precharged node. When *E-TV* propagates the worst delay rising transition at a precharging node to the following stages, it propagates a rising transition made through a precharging path only. Similarly, in case of a predischarged node, only a falling transition through a predischarging path is propagated to

the next stages.

A precharged module and N/P-type dynamic logic gates have been illustrated in Figures 2.8 and 2.10. Even though a precharged module and an N-type dynamic logic gate are similar in configuration, they can be differentiated by observing the way inputs affect their precharged node. In case of a precharged module, inputs are applied to the source/drain terminals of pass transistors connected to a precharged node. Note that the pass transistors may be controlled by other than clock signals. In case of an N-type dynamic logic gate, the inputs from previous stages are applied only to the gate terminals of MOS transistors in the N-type logic that composes evaluation paths between its precharged node and ground (refer to Figure 2.10).

When an N/P-type dynamic logic gate has more than one evaluation path (e.g., NOR gate), all evaluation paths may share one clocked transistor. Or, if it is more convenient for layout, each evaluation path may have its own clocked transistor. Even though the gates of these clocked transistors are usually tied to the same clock-node, they may be controlled by different clock-nodes. In this case, when Equation (2.15) is used to examine the falling transition at the precharged node, $\phi_j(F)$ must be the clock signal which controls each evaluation path. However, since these clock nodes must be in very close proximity, the clock signal arrival times at the nodes will be almost the same. Thus, *E-TV* selects one of the clock nodes randomly and uses that clock node when applying Equation (2.15). Some dynamic circuits do not have clocked transistors on their evaluation paths. In this case, *E-TV* checks whether the falling (rising) transition at the precharged (predischarged) node finishes before the next precharging (predischarging) period begins.

5.12. VERIFICATION OF DESIGN REFERENCES

When transparent clocked storage elements are used, two contiguous single-stage clocked paths can trade the amount of time available for logic evaluation with each other to satisfy timing constraints. This is illustrated, using Figure 5.23. Assume that delays through *CSE*'s are negligible. If *CSE*'s are positive edge-triggered type, *CL1* and *CL2* have following separate unrelated timing constraints from Equation (2.1):

$$D_{CL1} \leq T1 \tag{5.2}$$

$$D_{CL2} \leq T2 \tag{5.3}$$

If a delay through *CL1* is too large to satisfy Equation (5.2), a clock separation *T1* must be increased. Even though D_{CL2} is small, it is of no help for *CL1* to satisfy the given constraints. Now, suppose that *CSE*'s are active-high transparent. The timing constraints are, then, given as follows from Equation (2.4):

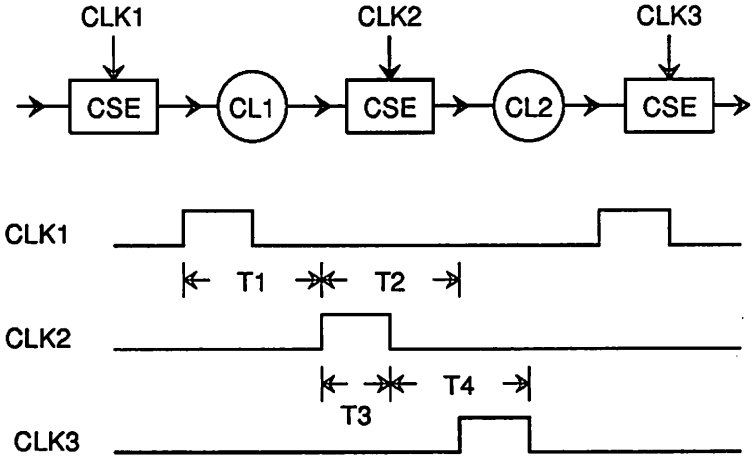


Figure 5.23 A Two-Stage Clocked Path

$$D_{CL1} \leq T1 + T3 \tag{5.4}$$

$$D_{CL2} \leq T3 + T4 \tag{5.5}$$

$$D_{CL1} + D_{CL2} \leq T1 + T3 + T4 \tag{5.6}$$

Equations (5.4) and (5.5) are timing constraints for each single-stage clocked path, while Equation (5.6) is for a two-stage clocked path. Notice that *T3* can be used for the evaluation of *CL1* or *CL2*. If D_{CL2} is small enough, *T3* can be used for the evaluation of *CL1*. This would provide the designer with flexibility in managing the system timing to satisfy constraints. Note that, however, even when all single-stage clocked paths satisfy timing constraints, their combined multi-stage clocked paths may not satisfy timing constraints, as described in Chapter 2. If the designer wish to maintain clocked path delays between pairs of transparent clocked storage elements in a bumper-to-bumper fashion to use clock separations maximally, not only is it very risky due to process variations but also the design complexity is a very big burden. Thus, the designer uses the timing constraints for edge-triggered type as "design references" for the easy management of the system timing, even though a system uses transparent clocked storage elements. In other words, assuming *CSE*'s are active-high transparent, the clocked paths of Figure 5.23 are designed to satisfy Equations (5.2) and (5.3) instead of Equations (5.4), (5.5) and (5.6). By using this design reference method, if all single-stage clocked paths satisfy the timing constraints, multistage clocked paths satisfy timing constraints automatically. Thus, if the longest path delay in each single-stage logic segment satisfies Equation (2.1), the whole system satisfies timing constraints. This design-reference method is used for *E-TV* to detect the possible timing errors of multistage clocked paths; when transparent clocked storage elements are used, *E-TV* checks if each logic segment satisfies the design reference rather than examining all multistage clocked paths individually.

CHAPTER 6

PERFORMANCE EVALUATION OF E-TV

Two microprocessor designs have been studied extensively to evaluate the performance of *E-TV*: CMOS *SOAR* and *SPUR*. CMOS *Smalltalk On A Risc (SOAR)* [98] is a 32 bit *Reduced Instruction Set Computer (RISC)*, designed at the University of California, Berkeley, using a 3 micron CMOS technology to execute the Smalltalk-80 programming language efficiently. *Symbolic Processing Using RISC's (SPUR)* [105] is a bus-oriented shared memory 32 bit multiprocessor under development at the University of California, Berkeley. Each workstation accommodates up to 12 processors, providing an aggregate performance of over 50 MIPS. These circuits were chosen because they use different design techniques (both dynamic and static logic), use state-of-the-art CMOS technologies, are large circuits, have been fabricated and do work, and, perhaps most importantly, were available in an understandable format and the designers were available for consultation.

In addition to path analysis, in this chapter, two other issues - the performance of the ELogic method and the accuracy of the MOS model used in the *E-TV* program - are discussed.

6.1. THE E-TV MOS MODEL

The present implementation of the *E-TV* program uses the Shichman-Hodges equations [92] to model MOS transistors, although any MOS companion model could be used in its place. The equations are used as the MOS1 dc model of the SPICE program [5]. While the SPICE MOS2 dc model [106, 107] models additional first- and second-order effects such as

surface field dependent mobility more accurately, it is computationally less efficient. As mentioned in Chapter 4, ELogic is a generalization of multi-state logic-level analysis (e.g. [7,22,66,8]) and circuit-level analysis which provides a less-precise analysis result to improve the performance for digital MOS circuits (e.g. [67,68,69,70,71]). Thus, analysis speed is one of the *E-TV*'s important features and the computation efficiency of the MOS model affects the performance. In addition, if well characterized and experimentally determined dc parameters are used, analysis accuracy using the MOS1 dc model is adequate for most MOS digital circuit design [29]. Therefore, the MOS1 dc model has been implemented in the present *E-TV* program. The experimental results which illustrate that the MOS1-DC model has adequate accuracy for the timing verification of MOS digital circuits are presented later. Note that if a more accurate MOS dc model is necessary, it is straightforward to replace the MOS1-DC model used in *E-TV* with a MOS2 implementation.

One of the popular MOS gate capacitance models is the piecewise linear voltage-dependent capacitance model proposed by Meyer [110] and recently improved by Sakallah et al. [111]. This model is used in the SPICE program. The present implementation of *E-TV* does not have a built-in gate capacitance model for MOS transistors and the user is required to approximate the gate capacitance by constant grounded capacitors (linear charge-voltage relationship). An experiment has been carried out to compare the performance of the constant grounded capacitor model (*LGC* model) to that of the Meyer gate capacitance model for delay computations. The experimental results showed that the LGC model is accurate enough to be used for timing verification. The LGC model and the experimental results are also presented in detail in this chapter.

6.2. PATH ANALYSIS AND THE ELOGIC METHOD

In order to evaluate the performance of the path analysis section of *E-TV* and the analysis method employed in the ELogic delay model, *E-TV* was used to extract the critical paths of the test circuits. The delays through the critical paths were then estimated by SPICE using the same MOS dc model (MOS1 dc model) and the delays estimated by the two programs were compared.

The ELogic delay model provides trade offs between accuracy and analysis speed by changing the voltage step-size. To illustrate this, *E-TV* was used to analyze test circuits using four different voltage step-sizes: 0.1V, 0.25V, 0.5V, and 1V. All CPU times presented are for a VAX 8800 running Ultrix[1]. *E-TV* reports as many critical paths in each logic segment as requested by the user, using the worst-case evaluation-time margin. Among them, SPICE [5] was used to analyze the most critical path of each logic segment for comparison. Since SPICE was used to analyze the extracted critical path only, its CPU times are not presented.

For the test circuits, only MOS2 model parameters were available. Thus, MOS1 dc model parameters for the SOAR and SPUR circuits were obtained as follows. There are five device parameters in the MOS1 model [106, 107] : *VTO* (zero-bias threshold voltage), *KP* (transconductance parameter), *GAMMA* (bulk threshold parameter), *PHI* (surface potential) and *LAMBDA* (channel-length modulation). Among them, all parameters except *KP* were used without adjustment. KP was determined by matching the dc transfer curve of an inverter made of NMOS and PMOS transistors with the most typical channel length and width for the circuit (usually the smallest device available in the technology). The current value flowing through the inverter was matched as the input voltage changed for the MOS1 and MOS2 models. The value of the model parameters used for the test circuits and the

[1] VAX and Ultrix are trademarks of Digital Equipment Corporation.

details of the method that determined *KP* of MOS1 are presented in Appendices 1 and 2, respectively. The SPICE inputs of the critical paths of the test circuits are included in Appendix 3. Copies of all software and the netlist of the test circuits are included as Appendix 4.

Sometimes, critical paths may be identical (contain the same circuit blocks) except for very minor differences near the ends. Such paths are called as *siblings*. When there are sibling critical paths in different logic segments, their path delays are almost the same, however, the succeeding time references are quite different. Thus, in this case, only the most critical path has been analyzed for comparison.

The first test circuit is the ALU of the SOAR microprocessor. The circuit uses domino logic [31] and precharged busses. A 1-bit circuit is illustrated in Figure 5.24. The logic evaluation performed by the circuit begins as $\phi 3$ goes high and must finish before $\phi 3$ goes low. The circuit contains 1,692 MOS transistors and 1,067 nodes. *E-TV* visited 408 and 871 nodes in order to locate the worst rising and falling delay paths, respectively. The numbers of transistor chains *E-TV* extracted to evaluate the rising and falling delays are 566 and 1,195, respectively. During a verification, the *select-signals* (*XOR*, *SUM*, *PASS*, *AND*, and *OR* nodes) were set at 5V so that all paths passing through the pass transistors controlled by those signals could be examined. Note that *select-signals* can be described as logic-control (CTRL) signals. In this case, *E-TV* also checks whether or not the signal at the output terminal of a pass transistor controlled by a *select-signal* settles before the pass transistor turns off. If a signal arrives earlier at an input terminal, the propagation is held until the pass transistor turns on. The worst case is when an LSB generated carry propagates to the MSB. The verification results are shown in Tables 5.1 and 5.2. Table 5.1 compares the analysis speed and the memory usage of *E-TV* for different voltage steps. The critical-path delays estimated

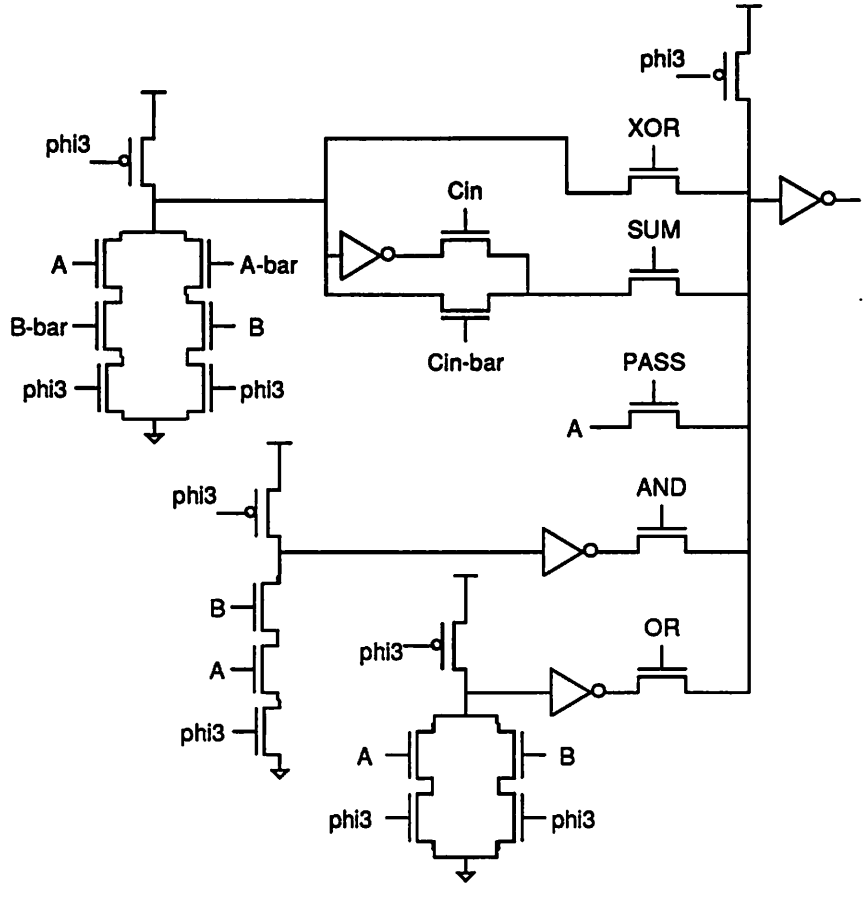


Figure 5.24 One Bit of SOAR ALU

by *E-TV* and SPICE are compared in Table 5.2. The tables illustrate the speed-accuracy trade-off of *E-TV*. That is, the delay error of *E-TV* decreases and the CPU time increases, as the voltage step-size gets smaller. This accuracy-speed trade-off can be obtained not only over analyses but also in the same analysis for the different parts of a circuit. From Table 5.2, it is clear that the ELogic method used in *E-TV* is accurate enough for practical use (delay error less than 7.4% compared to SPICE MOS1), if a voltage step of 0.5V or smaller is

used for this circuit design style.

The second test circuit is the SOAR circuit, excluding the ALU. The circuit contains 34,410 MOS transistors and 12,769 nodes. The numbers of nodes *E-TV* visited to locate the worst rising and falling delay paths are 47,226 and 49,717, respectively. *E-TV* computed the rising and the falling delays of 80,446 and 84,907 transistor chains, respectively. The verification results are presented in Tables 5.3 and 5.4. For this example, the delay error of *E-TV* is less than 13%, when a voltage step-size is not larger than 0.5V.

The first test circuit from the SPUR microprocessor is a NuBus interface controller [108] which is used in the cache controller [109]. The circuit is designed using a *Programmable Logic Array (PLA)* [25,28], whose input registers to the AND plane are active when $\phi 1$ is high, and whose output registers of the OR plane are active when $\phi 2$ is high. The circuit contains 740 MOS transistors and 231 nodes. *E-TV* located the worst rising and falling delay paths of 227 nodes, and computed the rising and the falling delays of 227 and 555 transistor chains, respectively. The analysis results are presented in Tables 5.5 and 5.6. For this example, the delay error of *E-TV* is less than 11% when compared to SPICE MOS1 and when the voltage step is 0.5V or smaller.

The next test circuit is an on-chip instruction cache (OIC) controller in SPUR, which has 1,205 MOS transistors and 404 nodes. The circuit contains four PLAs, which are connected as shown in Figure 5.25. The registers for the PLAs are pass transistor registers or precharged modules. Thus, for example, paths that end at $\phi 1$ registers may be constrained by $\phi 1(R)$ or $\phi 1(F)$. *E-TV* located the worst rising delay paths of 505 nodes and the worst falling delay paths of 532 nodes. And, the rising delays of 558 transistor chains and the falling delays of 1,615 transistor chains were evaluated by *E-TV*. The verification results are presented in Tables 5.7 and 5.8. Notice that *E-TV* identifies all logic segments and examines

	Voltage step			
	1V	0.5V	0.25V	0.1V
CPU time (sec)	37.1	60.6	96.3	219.9
Memory (M byte)	1.2	1.4	1.8	3.1

Table 5.1 CPU time and Memory Usage Comparisons of E-TV on The SOAR ALU

	E-TV				SPICE
	Voltage step				(MOS1)
	1V	0.5V	0.25V	0.1V	
$\phi3(R)$–$\phi3(F)$ seg.	77.6ns	69.3ns	67.5ns	66.9ns	64.5ns

Table 5.2 Critical Path Delay Estimates Comparison on The SOAR ALU Using E-TV and SPICE

	Voltage step			
	1V	0.5V	0.25V	0.1V
CPU time (sec)	8754	9823	11541	19842
Memory (M byte)	21	26	36	64

Table 5.3 CPU time and Memory Usage Comparisons of E-TV on The SOAR

	E-TV				SPICE
	Voltage step				(MOS1)
	1V	0.5V	0.25V	0.1V	
$\phi3(R)$–$\phi3(F)$ seg.	155.5ns	123.0ns	143.0ns	129.3ns	126.7ns
$\phi3(R)$–$\phi1(R)$ seg.	Sibling of $\phi3(R)$–$\phi3(F)$ seg.				
$\phi3(R)$–$\phi1(F)$ seg.	Sibling of $\phi3(R)$–$\phi3(F)$ seg.				
$\phi2(R)$–$\phi3(F)$ seg.	44.9ns	42.5ns	43.1ns	41.78ns	38.4ns
$\phi1(R)$–$\phi1(F)$ seg.	178.8ns	156.9ns	166.8ns	166.0ns	167.0ns

Table 5.4 Critical Path Delay Estimates Comparison on The SOAR Using E-TV and SPICE

	Voltage step			
	1V	0.5V	0.25V	0.1V
CPU time (sec)	6.2	9.6	18.3	39.1
Memory (M byte)	0.4	0.5	0.6	0.9

Table 5.5 CPU time and Memory Usage Comparisons of E-TV on The NuBus Interface Controller

	E-TV				SPICE
	Voltage step				(MOS1)
	1V	0.5V	0.25V	0.1V	
$\phi 1(R)$–$\phi 2(F)$ seg.	6.88ns	6.58ns	6.49ns	6.42ns	5.92ns

Table 5.6 Critical Path Delay Estimate Comparison on The NuBus Interface Controller Using E-TV and SPICE

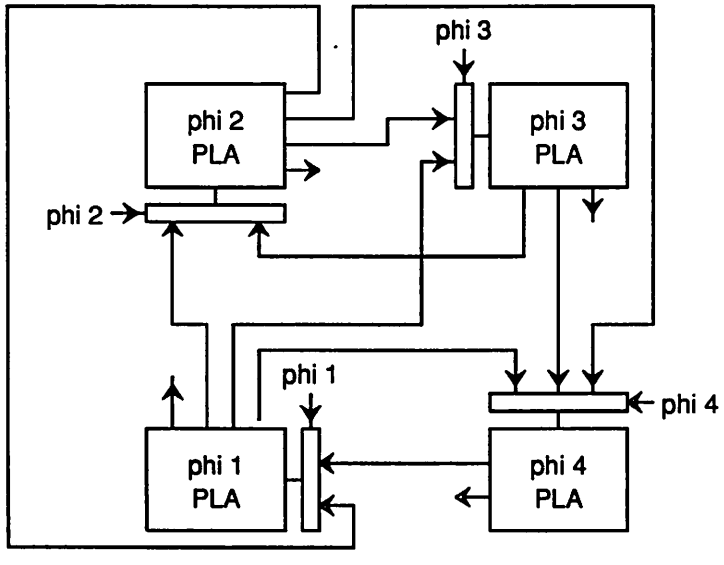


Figure 5.25 On-Chip Instruction Cache Controller in SPUR

	Voltage step			
	1V	0.5V	0.25V	0.1V
CPU time (sec)	13.7	19.4	31.7	69.8
Memory (M byte)	0.6	0.7	0.8	1.4

Table 5.7 CPU time and Memory Usage Comparisons of E-TV on The OIC Controller

	E-TV				SPICE (MOS1)
	Voltage step				
	1V	0.5V	0.25V	0.1V	
$\phi4(R)$–$\phi1(R)$ seg.	4.50ns	4.51ns	4.39ns	4.32ns	4.18ns
$\phi4(R)$–$\phi1(F)$ seg.	7.60ns	6.51ns	6.42ns	6.30ns	5.91ns
$\phi3(R)$–$\phi4(R)$ seg.	5.37ns	5.07ns	4.97ns	4.94ns	4.70ns
$\phi3(R)$–$\phi4(F)$ seg.	7.51ns	7.29ns	7.11ns	6.99ns	6.53ns
$\phi3(R)$–$\phi2(F)$ seg.	4.18ns	3.97ns	3.88ns	3.83ns	3.80ns
$\phi2(R)$–$\phi4(R)$ seg.	6.51ns	6.38ns	6.12ns	6.04ns	5.78ns
$\phi2(R)$–$\phi4(F)$ seg.	9.35ns	9.95ns	9.77ns	9.59ns	9.09ns
$\phi2(R)$–$\phi3(R)$ seg.	6.35ns	6.21ns	5.95ns	5.93ns	5.73ns
$\phi2(R)$–$\phi3(F)$ seg.	7.75ns	8.05ns	7.69ns	7.65ns	7.08ns
$\phi2(R)$–$\phi1(R)$ seg.	5.07ns	4.80ns	4.73ns	4.70ns	4.79ns
$\phi2(R)$–$\phi1(F)$ seg.	6.91ns	7.41ns	7.17ns	7.01ns	6.60ns
$\phi1(R)$–$\phi4(R)$ seg.	5.86ns	5.78ns	5.74ns	5.74ns	5.62ns
$\phi1(R)$–$\phi4(F)$ seg.	7.24ns	7.66ns	7.57ns	7.51ns	7.14ns
$\phi1(R)$–$\phi3(R)$ seg.	6.00ns	5.92ns	5.89ns	5.88ns	5.69ns
$\phi1(R)$–$\phi3(F)$ seg.	7.43ns	8.04ns	7.70ns	7.69ns	7.20ns
$\phi1(R)$–$\phi2(R)$ seg.	5.14ns	5.05ns	5.00ns	4.99ns	4.96ns
$\phi1(R)$–$\phi2(F)$ seg.	6.02ns	5.84ns	5.76ns	5.73ns	5.64ns

Table 5.8 Critical Path Delay Estimates Comparison on The OIC Controller Using E-TV and SPICE

them separately. For this circuit, when the voltage step does not exceed 0.5 V, the delay error of *E-TV* is typically less than 10% compared to SPICE MOS1, again an acceptable error in most cases.

The last test circuit is the SPUR ALU which was designed using the domino logic structure [31]. The circuit contains 4,286 MOS transistors and 2,258 nodes. *E-TV* visited 1,264 and 2,689 nodes to locate their worst rising and falling delay paths, respectively, and it evaluated the rising and the falling delays of 1,844 and 3,512 transistor chains, respectively. The results are shown in Tables 5.9 and 5.10. In this example, the delay estimates of *E-TV* are accurate to within 7.1% for all four voltage steps compared to SPICE MOS1.

From the experimental results, I have shown that *E-TV* verifies multiphase systems successfully, including those employing dynamic circuits. It has also been shown that the ELogic method analyzes a given circuit with good accuracy (typically less than 10% delay error when compared to SPICE using the same MOS dc model) with a voltage step which is not larger than 0.5V over a wide range of circuit design styles (both dynamic and static logic) and in a state-of-art CMOS technology. The analysis speed was shown to be fast enough for practical use.

6.3. EVALUATION OF THE E-TV MOS MODEL

In this section, the accuracy of the E-TV MOS model is evaluated using the SPICE program and it is shown that the MOS1-DC and LGC models introduced earlier have adequate accuracy to be used for the timing verification of most MOS digital systems. The SPICE MOS2 dc model and the Meyer capacitance model were used for comparison under the assumption that they are accurate enough to be used as references. The experiments have been performed in the following order, using the critical paths of the test circuits:

	Voltage step			
	1V	0.5V	0.25V	0.1V
CPU time (sec)	78.6	112.0	179.9	387.0
Memory (M byte)	2.6	3.1	4.0	6.7

Table 5.9 CPU time and Memory Usage Comparisons of E-TV on The SPUR ALU

	E-TV				SPICE
	Voltage step				(MOS1)
	1V	0.5V	0.25V	0.1V	
$\phi3(R)$–$\phi3(F)$ seg.	13.45ns	12.83ns	12.57ns	12.44ns	12.56ns
$\phi3(R)$–$\phi4(R)$ seg.	Sibling of $\phi3(R)$–$\phi3(F)$ seg.				
$\phi3(R)$–$\phi4(F)$ seg.	Sibling of $\phi3(R)$–$\phi3(F)$ seg.				
$\phi2(R)$–$\phi3(R)$ seg.	10.56ns	10.12ns	10.05ns	9.98ns	9.98ns
$\phi2(R)$–$\phi3(F)$ seg.	Sibling of $\phi2(R)$–$\phi3(R)$ seg.				
$\phi2(R)$–$\phi4(R)$ seg.	9.16ns	8.76ns	8.69ns	8.63ns	8.62ns
$\phi2(R)$–$\phi4(F)$ seg.	Sibling of $\phi2(R)$–$\phi4(R)$ seg.				

Table 5.10 Critical Path Delay Estimates Comparison on The SPUR ALU Using E-TV and SPICE

(1) Determine the value of KP for the MOS1-DC model from the given MOS2 parameters.

(2) Compare the accuracy of the MOS1-DC model and the MOS2 dc model by comparing their delay estimates for the critical-path of the test circuits.

(3) Compare the accuracy of the E-TV gate capacitance model (LGC model) and the Meyer gate capacitance model by comparing their delay estimates for the critical-path of the test circuits.

(4) Evaluate the aggregate effect of the E-TV MOS model (MOS1-DC and LGC models) compared to the MOS2 dc and Meyer capacitance models.

6.3.1. DETERMINATION OF THE MOS1 DC MODEL PARAMETERS

As mentioned before, it is necessary to determine KP from the given MOS2 parameters to use the MOS1 model. The effective value of KP for the MOS1 model depends on the dimension of MOS transistors. Thus, it is desirable to determine the value of KP for each different-sized MOS transistor so that MOS transistors with different dimension (channel length or width) can refer to a different model card. However, while the value of KP is very sensitive to the channel length of MOS transistors for short devices, it is less sensitive to the channel width. Therefore, KP was determined for each different channel length using the average channel width for devices of that length for the accuracy comparison of the MOS models.

The value of KP was determined according to the following steps:

Step 1: Determine KP by matching the DC characteristic of the MOS transistors as close as possible for each different channel length using inverters as described in Section 6.2 (see Appendix 2).

Step 2: Under the actual operating condition, the output of the inverter is not able to switch as fast as its input. Thus, KP was adjusted once more to match the delays through a 10-inverter chain with 100fF load at each node for the MOS1 and MOS2 models.

In **Step 1**, the values of KP for NMOS and PMOS transistors were adjusted separately, while they were scaled by the same constant in **Step 2** to match the inverter chain delays. The typical-sized transistors of the test circuits and their values of KP, obtained by **Steps 1** and **2**, are presented in Table 5.11.

Test Circuit	MOS Type	W/L (μ)	KP (μ)	
			Step 1	Step 2
SOAR	NMOS	5/3	21.9	20.99
	PMOS	5/3	4.21	4.06
	NMOS	5/7	19.0	19.63
	PMOS	5/7	3.95	4.08
SPUR	NMOS	11/1.6	40.8	36.0
	PMOS	6.4/1.6	14.5	12.78
	NMOS	11/2.4	41.6	37.6
	PMOS	6.4/2.4	14.8	13.39
	NMOS	11/3.2	41.6	38.6
	PMOS	6.4/3.2	14.8	13.75

Table 5.11 Effective Values of KP for The MOS1 Model for Test Circuits

6.3.2. THE E-TV MOS DC MODEL

In this section, the accuracy of the MOS1-DC model is compared to the MOS2 dc model. First, SPICE computed the delays through the critical paths presented in Section 6.2 using the MOS1-DC model with the KP as determined in Section 6.3.1 and it computed the delays again using the MOS2 dc model. Then the delays were compared. In order to compare the dc models only, capacitance parameters such as CGDO, CGSO, CGBO, CJ and CJSW were set at zero. Note that the gate capacitance can not be excluded for delay computations when the MOS2 model is used. If TOX (oxide thickness) is not specified, it is assumed to be 1000nm. On the other hand, if TOX is set to large value to reduce the gate capacitance, the surface mobility is affected. Thus, the Meyer model was used for both MOS1 and MOS2 models. The delays of the critical paths using the two MOS models are compared from Tables 5.12 to 5.16. The tables illustrate that the delay error of the MOS1-DC model was less than 10% for more than half of the critical paths and it was 18% at worst for the test circuits, compared to the MOS2 dc model. Note that it was not necessary to

Critical Path	Path Delay		Error of MOS1-DC compared to MOS2 dc
	MOS1-DC	MOS2 dc	
$\phi 3(R)$–$\phi 3(F)$ seg.	104.48n	88.4n	18.7%

Table 5.12 Delay Estimate Comparison on The Critical Path of The SOAR ALU Using MOS1-DC and MOS2 dc Models

Critical Path	Path Delay		Error of MOS1-DC compared to MOS2 dc	Clock skew
	MOS1-DC	MOS2 dc		
$\phi 3(R)$–$\phi 3(F)$ seg.	145.1n	131.2n	10.5%	2.3n
$\phi 2(R)$–$\phi 3(F)$ seg.	45.87n	41.97n	9.3%	1.43n
$\phi 1(R)$–$\phi 1(F)$ seg.	215.7n	190.3n	13.3%	7.7n

Table 5.13 Delay Estimates Comparison on The Critical Paths of The SOAR Circuit Using MOS1-DC and MOS2 dc Models

Critical Path	Path Delay		Error of MOS1-DC compared to MOS2 dc
	MOS1-DC	MOS2 dc	
$\phi 1(R)$–$\phi 2(F)$ seg.	7.016ns	7.206ns	-2.6%

Table 5.14 Delay Estimate Comparison on The Critical Path of The NuBus Interface in The SPUR Circuit Using MOS1-DC and MOS2 dc Models

Critical Path	Path Delay		Error of MOS1-DC compared to MOS2 dc
	MOS1-DC	MOS2 dc	
$\phi4(R)$–$\phi1(R)$ seg.	4.974ns	4.904ns	1.4%
$\phi4(R)$–$\phi1(F)$ seg.	7.011ns	6.970ns	0.6%
$\phi3(R)$–$\phi4(R)$ seg.	5.547ns	5.571ns	-0.4%
$\phi3(R)$–$\phi4(F)$ seg.	7.703ns	7.728ns	-0.3%
$\phi3(R)$–$\phi2(F)$ seg.	4.594ns	4.412ns	4.1%
$\phi2(R)$–$\phi4(R)$ seg.	6.765ns	6.798ns	-0.5%
$\phi2(R)$–$\phi4(F)$ seg.	10.81ns	10.90ns	-0.8%
$\phi2(R)$–$\phi3(R)$ seg.	6.775ns	7.420ns	-8.7%
$\phi2(R)$–$\phi3(F)$ seg.	8.566ns	8.578ns	-0.1%
$\phi2(R)$–$\phi1(R)$ seg.	5.692ns	6.068ns	-6.2%
$\phi2(R)$–$\phi1(F)$ seg.	7.811ns	7.777ns	0.4%
$\phi1(R)$–$\phi4(R)$ seg.	6.619ns	7.138ns	-7.3%
$\phi1(R)$–$\phi4(F)$ seg.	8.445ns	8.953ns	-5.7%
$\phi1(R)$–$\phi3(R)$ seg.	6.737ns	7.276ns	-7.4%
$\phi1(R)$–$\phi3(F)$ seg.	8.537ns	9.060ns	-5.8%
$\phi1(R)$–$\phi2(R)$ seg.	5.899ns	6.340ns	-7.0%
$\phi1(R)$–$\phi2(F)$ seg.	6.757ns	7.127ns	-5.2%

Table 5.15 Delay Estimates Comparison on The Critical Paths of The OIC Controller in The SPUR Circuit Using MOS1-DC and MOS2 dc Models

Critical Path	Path Delay		Error of MOS1-DC compared to MOS2 dc
	MOS1-DC	MOS2 dc	
$\phi3(R)$–$\phi3(F)$ seg.	15.79ns	14.07ns	12.2%
$\phi2(R)$–$\phi3(R)$ seg.	12.53ns	11.18ns	12.1%
$\phi2(R)$–$\phi4(R)$ seg.	10.76ns	9.458ns	13.8%

Table 5.16 Delay Estimates Comparison on The Critical Paths of The SPUR ALU Circuit Using MOS1-DC and MOS2 dc Models

"tune" the model parameters to the circuit to obtain this delay error. Existing switch-level delay models usually require extensive tuning of the parameters to a given circuit and to a given process during the circuit design to obtain usable accuracy, but even after that process they still suffer from poor accuracy as pointed out in Chapter 4. The *E-TV* delay error can be reduced further, if desired, by determining KP for each different channel width as well as each different channel length, or by adapting the MOS1-DC model to include more sophisticated geometric modeling. However, this was not done in *E-TV* since the error obtained with the efficient MOS1-DC model is small enough for practical purposes.

6.3.3. THE E-TV GATE CAPACITANCE MODEL

Figure 5.26 illustrates the nonlinear capacitances associated with MOS transistors, whose values vary with the node voltages at *G, D, S* and *B* which represent gate, drain, source and bulk terminals respectively. Figure 5.27 illustrates the LGC model that E-TV uses. The model assumes that the bulk terminal is connected to a constant voltage source. The user is required to specify the capacitances shown in the figure, where the capacitances are given as follows:

$$Cg = Kgate \times W \times (Cox \times (L - 2 \times LD) + CGDO + CGSO) \quad (5.7.a)$$

$$Cd = Kgate \times W \times CGDO + Keqd \times (Cj \times AD + Cjsw \times PD) \quad (5.7.b)$$

$$Cs = Kgate \times W \times CGSO + Keqs \times (Cj \times AS + Cjsw \times PS) \quad (5.7.c)$$

where

Kgate	:	parameter to obtain equivalent linear gate capacitance
W (L)	:	channel width (length) of a transistor
LD	:	lateral diffusion
Cox	:	gate capacitance per unit area
CGDO (CGSO)	:	gate-drain (gate-source) overlap capacitance
Cj (Cjsw)	:	junction bottom (sidewall) capacitance
AS (PS)	:	source area (perimeter)

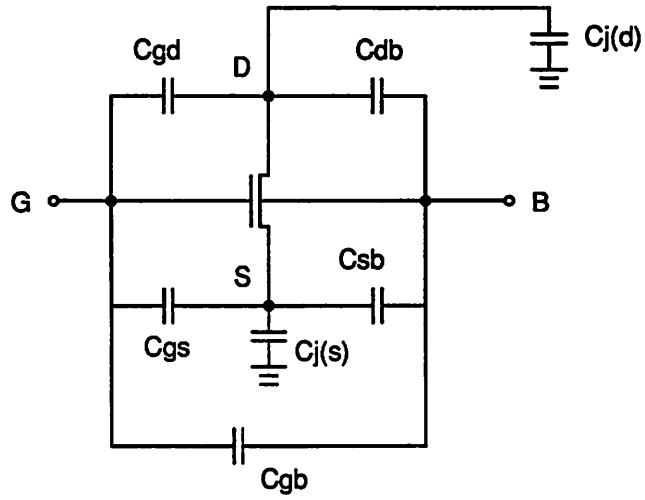


Figure 5.26 Nonlinear Capacitances Associated With MOS Transistors

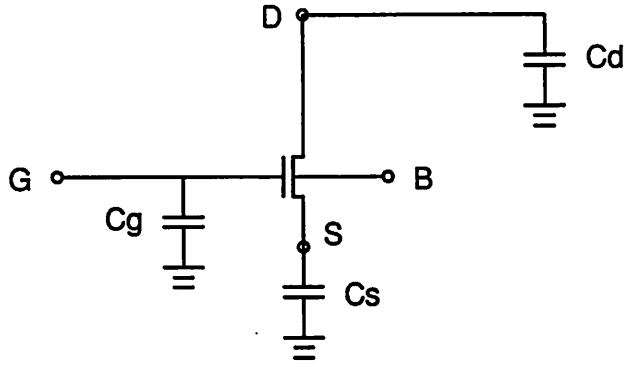


Figure 5.27 The Gate Capacitance Model E-TV Uses
(The Constant Grounded Capacitance (LGC) Model)

AD (PD) : drain area (perimeter)
Keqd (Keqs) : parameter to obtain equivalent linear junction capacitance

The suggested value of *Kgate* is 1.3, as is derived from the experimental results below.

The purpose of the experiment described in this section is to determine the value of *Kgate* by which the LGC model of Figure 5.27 can approximate the Meyer gate capacitance model best in terms of path delays, and to evaluate how good the resulting approximation is. The effect of junction capacitances on path delays is usually negligible compared to other capacitances. Thus, Equations (5.7.a), (5.7.b) and (5.7.c) can be rewritten as follows:

$$Cg = Kgate \times W \times (Cox \times (L - 2 \times LD) + CGDO + CGSO) \quad (5.8.a)$$

$$Cd = Kgate \times W \times CGDO \quad (5.8.b)$$

$$Cs = Kgate \times W \times CGSO \quad (5.8.c)$$

If the junction capacitances are not negligible for path delays, the suggested value of *Keqd* and *Keqs* for non-ground terminals whose node voltage switches between 0V and 5V is 0.54 from [29]. For the experiment, the MOS1 DC model was used to compare the LGC and Meyer models because if the MOS2 model is used the gate capacitance can not be excluded from SPICE simulation. Since the oxide thickness of the SOAR and SPUR circuits are 50nm and 25nm, the corresponding values of *Cox* are 0.69fF/μ^2 and 1.38/μ^2 respectively, and the constant grounded capacitances were computed using Equations (5.8.a) to (5.8.c). Tables 5.17 to 5.21 compare the delays through the critical paths, which were computed using the Meyer model and the LGC model with five different values of *Kgate*. The numbers in parentheses are the delay errors of the LGC model compared to the Meyer model. In the table, "Avg Abs Err" is the average of the absolute error, "Max Err" is the maximum error and "Min Err" is the minimum error, when *Kgate* of the corresponding column was used. From the tables, it is clear that 1.3 is the best choice as the value of *Kgate*. For the critical

Critical Path	Meyer Model	LGC Model				
		Kgate				
		1.0	1.1	1.2	1.3	1.4
$\phi3(R)$–$\phi3(F)$ seg.	151.6n	137.1n (-9.6%)	141.8n (-6.5%)	146.4n (-3.4%)	150.8n (-0.5%)	155.2n (2.4%)

Table 5.17 Delay Estimate Comparison on The Critical Path of The SOAR ALU Using The Meyer and LGC Models

Critical Path	Meyer Model	LGC Model				
		Kgate				
		1.0	1.1	1.2	1.3	1.4
$\phi3(R)$–$\phi3(F)$ seg.	167.7n	162.2n (-3.3%)	164.8n (-1.7%)	169.5n (1.1%)	171.3n (2.1%)	173.6n (3.5%)
$\phi2(R)$–$\phi3(F)$ seg.	53.4n	51.5n (-3.6%)	52.3n (-2.0%)	53.1n (-0.6%)	54.0n (1.1%)	54.8n (2.6%)
$\phi1(R)$–$\phi1(F)$ seg.	270.6n	251.1n (-7.2%)	257.8n (-4.7%)	262.9n (-2.8%)	269.2n (-0.5%)	273.0n (0.9%)
Avg Abs Err		4.7%	2.8%	1.5%	1.2%	2.3%
Max Err		-7.2%	-4.7%	-2.8%	2.1%	3.5%
Min Err		-3.3%	-1.7%	-0.6%	-0.5%	0.9%

Table 5.18 Delay Estimates Comparison on The Critical Paths of The SOAR circuits Using The Meyer and LGC Models

Critical Path	Meyer Model	LGC Model				
		Kgate				
		1.0	1.1	1.2	1.3	1.4
$\phi1(R)$–$\phi2(F)$ seg.	7.26n	7.1n (-2.2%)	7.153n (-1.5%)	7.2n (-0.8%)	7.26n (0%)	7.31n (0.7%)

Table 5.19 Delay Estimate Comparison on The Critical Path of NuBus interface in The SPUR Circuit Using The Meyer and LGC Models

Critical Path	Meyer Model	LGC Model				
		Kgate				
		1.0	1.1	1.2	1.3	1.4
$\phi4(R)$–$\phi1(R)$ seg.	5.19n	5.16n (-0.6%)	5.21n (0.4%)	5.27n (1.5%)	5.32n (2.5%)	5.37n (3.5%)
$\phi4(R)$–$\phi1(F)$ seg.	7.28n	7.09n (-2.6%)	7.14n (-1.9%)	7.20n (-1.1%)	7.26n (-0.3%)	7.32n (0.5%)
$\phi3(R)$–$\phi4(R)$ seg.	5.75n	5.67n (-1.4%)	5.72n (-0.5%)	5.77n (0.3%)	5.81n (1.0%)	5.86n (1.9%)
$\phi3(R)$–$\phi4(F)$ seg.	8.01n	7.89n (-1.5%)	7.97n (-0.5%)	8.02n (0.1%)	8.10n (1.1%)	8.16n (1.9%)
$\phi3(R)$–$\phi2(F)$ seg.	4.79n	4.68n (-2.3%)	4.72n (-1.5%)	4.77n (-0.4%)	4.81n (0.4%)	4.85n (1.3%)
$\phi2(R)$–$\phi4(R)$ seg.	7.00n	6.96n (-0.6%)	7.02n (0.3%)	7.07n (1.0%)	7.13n (1.9%)	7.19n (2.7%)
$\phi2(R)$–$\phi4(F)$ seg.	11.22n	10.90n (-2.9%)	10.99n (-2.0%)	11.07n (-1.3%)	11.15n (-0.6%)	11.24n (0.2%)
$\phi2(R)$–$\phi3(R)$ seg.	6.99n	6.78n (-3.0%)	6.81n (-2.6%)	6.84n (-2.1%)	6.87n (-1.7%)	6.90n (-1.3%)
$\phi2(R)$–$\phi3(F)$ seg.	8.84n	8.68n (-1.8%)	8.75n (-1.0%)	8.81n (-0.3%)	8.88n (0.5%)	8.95n (1.2%)
$\phi2(R)$–$\phi1(R)$ seg.	5.92n	5.73n (-3.2%)	5.77n (-2.5%)	5.8n (-2.0%)	5.83n (-1.5%)	5.86n (-1.0%)
$\phi2(R)$–$\phi1(F)$ seg.	8.09n	7.90n (-2.3%)	7.96n (-1.6%)	8.03n (-0.7%)	8.09n (0.0%)	8.15n (0.7%)
$\phi1(R)$–$\phi4(R)$ seg.	6.84n	6.60n (-3.5%)	6.64n (-2.9%)	6.67n (-2.5%)	6.70n (-2.0%)	6.73n (-1.6%)
$\phi1(R)$–$\phi4(F)$ seg.	8.71n	8.32n (-4.5%)	8.36n (-4.0%)	8.39n (-3.6%)	8.43n (-3.2%)	8.46n (-2.9%)
$\phi1(R)$–$\phi3(R)$ seg.	6.94n	6.73n (-3.0%)	6.77n (-2.4%)	6.80n (-2.0%)	6.83n (-1.6%)	6.86n (-1.2%)
$\phi1(R)$–$\phi3(F)$ seg.	8.77n	8.45n (-3.6%)	8.48n (-3.3%)	8.52n (-2.8%)	8.55n (-2.1%)	8.59n (-1.6%)
$\phi1(R)$–$\phi2(R)$ seg.	6.11n	5.88n (-3.8%)	5.92n (-3.1%)	5.95n (-2.6%)	5.98n (-2.1%)	6.01n (-1.6%)
$\phi1(R)$–$\phi2(F)$ seg.	7.08n	6.81n (-3.8%)	6.87n (-3.0%)	6.91n (-2.4%)	6.97n (-1.6%)	7.02n (-0.8%)
Avg Abs Err		2.6%	2.0%	1.6%	1.4%	1.6%
Max Err		-4.5%	-4.0%	-3.6%	-3.2%	3.5%
Min Err		-0.6%	0.3%	0.1%	0.0%	0.5%

Table 5.20 Delay Estimates Comparison on The Critical Paths of The OIC Controller in The SPUR Circuit Using The Meyer and LGC Models

paths of the test circuits, the delay error of the LGC model using the *Kgate* of 1.3 was less than 3.2% compared to the Meyer model. Thus, I conclude that the LGC model is accurate enough to be used for timing verification.

6.3.4. THE AGGREGATE ACCURACY OF THE E-TV MOS DC AND LGC MODELS

The accuracy of the E-TV MOS dc (MOS1-DC) and LGC models was evaluated in Sections 6.3.2 and 6.3.3 separately. In this section, the aggregate effect of the MOS1-DC and LGC models is presented. The delays through the critical paths of the test circuits were computed by SPICE; first using the MOS2 dc and Meyer capacitance models and then using the MOS1-DC and LGC (*Kgate* was set at 1.3) models. The results are compared in Tables 5.22 to 5.26. In the tables, the numbers in parentheses are the delay errors of the MOS1-DC and LGC models compared to the MOS2 dc and Meyer capacitance models.

As seen in Tables 5.22 and 5.23, the delay error using the MOS1-DC and LGC models was 10% to 12% for the critical paths of the SOAR circuit, compared to the MOS2 dc model with the Meyer capacitance model. From Tables 5.24 and 5.25, when the MOS1-DC and LGC models were used to compute the delays through the critical paths of the NuBus interface and the OIC controller of the SPUR circuit, the delay error ranged from -9% to 4%, compared to using the MOS2 dc and Meyer models. From Table 5.26, the MOS1-DC and LGC models overestimated the delays through the critical paths of the SPUR ALU by 12% to 17%, compared to the MOS2 dc and Meyer capacitance models. Therefore, for the test circuits, when the MOS1-DC and LGC models were used, the absolute delay error for critical paths was less than 17% compared to the MOS2 model using the Meyer capacitance model. Since much of the delay error comes from the MOS dc model rather than from the LGC model, as

Critical Path	Meyer Model	LGC Model				
		Kgate				
		1.0	1.1	1.2	1.3	1.4
$\phi3(R)$–$\phi3(F)$ seg.	16.73n	16.25n (-2.9%)	16.47n (-1.6%)	16.67n (-0.4%)	16.87n (0.8%)	17.08n (2.1%)
$\phi2(R)$–$\phi3(R)$ seg.	13.09n	13.04n (-0.4%)	13.19n (0.8%)	13.35n (2.0%)	13.50n (3.1%)	13.66n (4.3%)
$\phi2(R)$–$\phi4(R)$ seg.	11.22n	11.19n (-0.3%)	11.32n (0.9%)	11.45n (2.0%)	11.57n (3.1%)	11.70n (4.3%)
Avg Abs Err		1.2%	1.1%	1.47%	2.3%	3.6%
Max Err		-2.9%	-1.6%	2.0%	3.1%	4.3%
Min Err		-0.3%	0.8%	-0.4%	0.8%	2.1%

Table 5.21 Delay Estimates Comparison on The Critical Paths of The SPUR ALU Using The Meyer and LGC Models

Critical Path	MOS2 dc & Meyer	MOS1-DC & LGC (*Kgate*=1.3)
$\phi3(R)$–$\phi3(F)$ seg.	136.0n	150.8n (10.9%)

Table 5.22 Delay Estimate Comparison on The Critical Path of The SOAR ALU Using The MOS2 dc and Meyer Models, and The MOS1-DC and LGC Models

Critical Path	MOS2 dc & Meyer	MOS1-DC & LGC (*Kgate*=1.3)
$\phi3(R)$–$\phi3(F)$ seg.	152.9n	171.3n (12%)
$\phi2(R)$–$\phi3(F)$ seg.	48.6n	54.0n (11.1%)
$\phi1(R)$–$\phi1(F)$ seg.	241.9n	269.2n (11.3%)

Table 5.23 Delay Estimates Comparison on The Critical Paths of The SOAR circuit Using The MOS2 dc and Meyer Models, and The MOS1-DC and LGC Models

Critical Path	MOS2 dc & Meyer	MOS1-DC & LGC (*Kgate*=1.3)
$\phi1(R)$–$\phi2(F)$ seg.	7.46n	7.26n (-2.7%)

Table 5.24 Delay Estimate Comparison on The Critical Path of The SPUR NuBus interface Using The MOS2 dc and Meyer Models, and The MOS1-DC and LGC Models

Critical Path	MOS2 dc & Meyer	MOS1-DC & LGC (*Kgate*=1.3)
$\phi4(R)$–$\phi1(R)$ seg.	5.14n	5.32n (3.5%)
$\phi4(R)$–$\phi1(F)$ seg.	7.16n	7.26n (1.4%)
$\phi3(R)$–$\phi4(R)$ seg.	5.79n	5.81n (0.3%)
$\phi3(R)$–$\phi4(F)$ seg.	8.05n	8.10n (0.6%)
$\phi3(R)$–$\phi2(F)$ seg.	4.62n	4.81n (4.1%)
$\phi2(R)$–$\phi4(R)$ seg.	7.04n	7.13n (1.3%)
$\phi2(R)$–$\phi4(F)$ seg.	11.32n	11.15n (-1.5%)
$\phi2(R)$–$\phi3(R)$ seg.	7.66n	6.87n (-10.3%)
$\phi2(R)$–$\phi3(F)$ seg.	8.87n	8.88n (0.1%)
$\phi2(R)$–$\phi1(R)$ seg.	6.3n	5.83n (-7.5%)
$\phi2(R)$–$\phi1(F)$ seg.	8.08n	8.09n (0.1%)
$\phi1(R)$–$\phi4(R)$ seg.	7.36n	6.70n (-9.0%)
$\phi1(R)$–$\phi4(F)$ seg.	9.23n	8.43n (-8.7%)
$\phi1(R)$–$\phi3(R)$ seg.	7.49n	6.86n (-8.4%)
$\phi1(R)$–$\phi3(F)$ seg.	9.3n	8.55n (-8.1%)
$\phi1(R)$–$\phi2(R)$ seg.	6.58n	5.98n (-9.1%)
$\phi1(R)$–$\phi2(F)$ seg.	7.46n	6.97n (-6.6%)

Table 5.25 Delay Estimates Comparison on The Critical Paths of The SPUR OIC Controller Using The MOS2 dc and Meyer Models, and The MOS1-DC and LGC Models

Critical Path	MOS2 dc & Meyer	MOS1-DC & LGC (*Kgate*=1.3)
$\phi3(R)$–$\phi3(F)$ seg.	14.94n	16.87n (12.9%)
$\phi2(R)$–$\phi3(R)$ seg.	11.74n	13.50n (15.0%)
$\phi2(R)$–$\phi4(R)$ seg.	9.91n	11.57n (16.7%)

Table 5.26 Delay Estimates Comparison on The Critical Paths of The SPUR ALU Using The MOS2 dc and Meyer Models, and The MOS1-DC and LGC Models

shown in Sections 6.3.2 and 6.3.3, the delay error is expected to reduce by tuning the parameters such as KP and VTO for different channel widths as well as for different channel lengths and using design experience.

6.3.5. CONCLUSIONS

The purpose of using timing verifiers is to find problematic signal paths in a digital system due to path delays that are either too long or too short, and which the designer has difficulty finding because of the complexity of the circuit under analysis. That is, rather than being used as a *substitute* for a circuit analysis program such as SPICE [5] or ASTAP [6] or for a logic simulator such as LOGIS [9] or ILOGS [10], a timing verifier is used to detect the *possibility* of timing errors and to locate the associated paths in a digital system so that those suspected signal paths can be analyzed in more detail using accurate circuit-level analysis programs. Therefore, while the MOS model to be used by timing verifiers must be accurate enough to locate the problematic sigal paths, it does not require the same accuracy as the model used by circuit analysis programs. Instead, the MOS model for timing verifiers should be fast enough to examine the millions of paths in the system in a reasonable amount of CPU time, while accurate enough to identify the critical paths correctly. Note that, in general, a MOS model becomes less efficient as it becomes more accurate and vice versa. Therefore, it is important to choose the proper compromise between accuracy and efficiency.

As described in Chapter 4, there are two kinds accuracy associated with delay models to consider for timing verification: *absolute* accuracy and *relative* accuracy. The absolute measure represents the accuracy of the delay estimates by a delay model compared to the true delay values (as determined by SPICE, for example), while the relative accuracy represents the ability of the delay model to determine the slower and faster paths relative to one another.

Thus, if a delay model has good absolute accuracy, it also has good relative accuracy while the opposite is not true. In order that a timing verifier may detect a possible timing error, the delay model needs good absolute accuracy, while the delay model only needs good relative accuracy to order signal paths in terms of relative delay.

- The existing switch-level RC delay models are very efficient, but they have very poor absolute accuracy. The parameters for the RC delay models are usually obtained by estimating the delays through a small set of static gates using the delay models and SPICE, and then matching the estimated delays by adjusting some coarse, circuit-specific parameters. It is the SPUR designers' experience that when those parameters are adjusted optimally, the typical delay error of the timing verifiers using the RC delay models (e.g., [23, 24]) is about 30 % for paths made of static gate chains only, about 100 % for typical signal paths made of static gates and pass transistor chains, and much greater (up to 300 % to 400 % in the worst case) for pass transistor chains [112], when compared to the delays estimated by SPICE. Certainly, this large delay error is not acceptable for a delay model to be used for the detection of the possibility of timing errors. Even though, as with any other delay model, the error of such RC delay models can be reduced by adjusting parameters repeatedly after delay comparisons with the SPICE program on a circuit-specific basis (the delay estimates by the RC delay models presented in Chapter 4 were obtained after very careful parameter tuning to achieve the best results by comparing them to the SPICE estimates), it is not meaningful to use such an approach because the user already has the SPICE delay estimates to begin with! On the other hand, it has been illustrated that the maximum delay error of the *E-TV* MOS model (MOS1-DC and LGC models) was only 17% for the critical paths of the test circuits *without* repeated tuning of MOS parameters, when compared to the MOS2 dc and Meyer capacitance models. Thus, I conclude that the *E-TV* MOS model has sufficient accuracy to detect the pos-

sible timing errors in systems of the types described in this dissertation. For some timing verifier applications (e.g., locating the slowest paths), the delay model may only need good relative accuracy. However, as illustrated in Chapter 4, the RC delay models do not have good relative accuracy either. They could not list signal paths in the SOAR circuit in the correct relative delay order and hence failed to locate the slowest path. On the other hand, the ELogic delay model located the path successfully. To conclude, the MOS1-DC and LGC models have far superior accuracy compared to those of the RC delay models used in existing timing verifiers. While the MOS2 dc and Meyer models might provide more accurate delay estimates, they are computationally much less efficient than the MOS1-DC and LGC models and do not provide better relative timing estimates. For these reasons, the MOS1-DC and LGC models are provided in the *E-TV* program.

CHAPTER 7

CONCLUSIONS

Timing verification programs are very important CAD tools for improving the operating speed, reducing the silicon area and power consumption, and detecting the possibility of timing errors in a digital system. Even though simulation can be used to analyze a circuit for the same purposes, the necessary work is exponential in the number of input nodes. Hence timing verification, which carries out only one analysis (or a small number of analyses) and provides the same or more information for the improvement or correction of the design much more quickly, is used in preference.

This dissertation summarized the timing constraints which synchronous systems must satisfy to operate properly, including those for specific design styles using precharging and predischarging as well as those for signal paths between pairs of clocked storage elements. Algorithms for the verification of a synchronous system which uses both edge-triggered and transparent clocked storage elements in the same design have been presented.

Timing verifiers are available at switch level, block level, and behavioral level. Among them, switch-level timing verifiers are the most appropriate for MOS designs. Unfortunately, the existing switch-level delay models suffer from poor accuracy. This dissertation presented the ELogic delay model which is not only accurate but also fast enough for practical use. The numerical properties of the ELogic algorithm such as stability and accuracy have been investigated. Since the voltage change a node can make is bounded, the algorithm has proven to remain in a stable region when solving a linear RC test circuit. The ELogic algorithm can also be used for simulation.

E-TV is an accurate timing verifier developed for MOS designs and uses the ELogic delay model. It detects the possibility of timing errors and locates the critical paths in synchronous and combinational MOS digital systems. *E-TV* allows the designer to use both transparent and edge-triggered clocked storage elements in the same synchronous system.

Since *E-TV* traverses a circuit in the topological order of nodes, its running time is almost linear with circuit size. Like some other switch-level timing verifiers [24,57], *E-TV* uses a rule-based approach to derive transistor signal flow. An approach and a set of rules developed for *E-TV* were presented. By using the idea of biconnected transistor groups, the chance of the signal-flow assignment has been improved. A novel loop-breaking algorithm which is simple but greatly reduces the number of blocked forward paths was presented. An efficient switch-level simulation algorithm using topological order of nodes is used to propagate fixed nodes in *E-TV*.

E-TV approximates the clocked storage element using two types of element: clocked transparent element and memory element. This approach eliminates problems that arise from treating any clocked storage element as a pass transistor register, as other switch-level timing verifiers do. By comparing the transitions at internal nodes of clocked storage elements directly, *E-TV* does not need to compute setup times, and clock skew is taken into account for the detection of time errors automatically.

The experimental results indicate that *E-TV* can verify multiphase systems successfully, including dynamic circuits such as precharged modules, domino, and NORA logic. *E-TV* with 0.5V step required about 2.7 hours on a VAX 8800 to verify a large system with 34,410 MOS transistors. The current implementation of the *E-TV* program uses the MOS1 DC and linear grounded capacitance models for MOS transistors. When the voltage step is equal to or smaller than 0.5V, the delay error of *E-TV* was typically less than 10% (as the voltage step

gets smaller, so does the delay error), compared to SPICE using the same MOS DC model. Thus, the ELogic analysis method was shown to be not only efficient but also accurate enough for timing verification. When compared to the SPICE2 DC and Meyer gate capacitance models, the *E-TV* MOS model were accurate with less than 17% of delay error for the critical paths of test circuits, and hence has sufficient accuracy to be used for timing verification. Existing switch-level timing verifiers use RC delay models. Note that, while they are efficient, they are not sufficiently accurate. On other hand, while the MOS2 DC and Meyer models are more accurate, they are not efficient enough to examine the uncountable paths in a design.

Future work to be done remains in following areas. First, it is worthwhile to apply the concept of "slacks" for MOS source/drain channels to provide the user with a measure of the severity of the timing problem.

Second, the interface of *E-TV* to the design data base such as OCT [113, 114] will provide many benefits. OCT is a CAD/VLSI data representation system which offers an interface for storing information about the various aspects of an evolving chip design. Some benefits for using such a data base includes the following. Because the higher level circuit information is available, easy solutions may be found for some problems such as identifying clocked storage elements. The program can also be enhanced to be more user-friendly. For example, the program may highlight the worst delay path on a layout using a graphic editor such as VEM [113, 115] so that the designers can locate it easily. Another possible benefit is that the program can be easily used for the optimization of a design by logic synthesis tools that use the same data base.

One final area of future work is due to the fact *E-TV* may occasionally report unrealistic worst delay paths. This is a generic limitation of all timing verifiers using the value-

independent approach. As mentioned in Chapter 3, there have been some attempts [42, 43] to identify such false paths automatically. However, they are not efficient enough for practical use, and do not take the effect of the propagation delay into account properly in determining whether a particular path is sensitizable. Developing an efficient algorithm or method which can locate these false paths and exclude them from consideration is still one of the "open" research areas in timing verification.

REFERENCES

1. C. Ho and S. Hansen, "SUPREM III - A program for integrated circuit process modeling and simulation," *Stanford Electronics Lab. Tech. Report*, July 1983.
2. S. Selberherr, A.Schutz, and H.W.Potzl, "MINIMOS - A two dimensional MOS transistor analyzer," *IEEE Trans. Electron Devices*, vol. Vol.ED-27, p. 1540, 1980.
3. E. Butula, P. Cottrell, B. Grossman, and K. Salsburg, "Finite-element analysis of semiconductor devices: The FIELDAY program," *IBM J.Res.Develp.*, vol. Vol.25, pp. 218-231,, July 1981.
4. M. Pinto, C. Rafferty, and R. Dutton, "PISCES-II: Poisson and continuity equation solver," *Stanford Electronics Lab. Tech. Report*, Sept. 1984.
5. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *UCB/ERL M75/520*, University of California, Berkeley, May 1975. Ph.D. Dissertation
6. W. Weeks, A. Jimenez, G. Mahoney, D. Mehta, H. Qassemzadeh, and T.R. Scott, "Algorithms for ASTAP - A network analysis program," *IEEE Trans. on Circuit Theory*, vol. CT-20, pp. 628-634, Nov. 1973.
7. R. Bryant, "An Algorithm for MOS Logic Simulation," *LAMBDA*, pp. 46-53, 4th Quarter 1980.
8. Z. Barzilai, D. Beece, L. Huisman, V. Iyengar, and G. Silberman, "SLS-A Fast Switch-Level Simulator," *IEEE Trans. on CAD of ICAS*, vol. Vol. 7, No. 8, pp. 838-849, Aug. 1988.
9. *LOGIS: User's Manual Version 4*, ISD Corporation, 1980.
10. F. Jenkins, *ILOGS: User's Manual*, Simutec, 1982.
11. A. Insinga, "Behavioral Modeling in a Structural Logic Simulator," *ICCAD-84*, pp. 42-44, Santa Clara, CA, Nov. 1984.
12. R. Lathrop and R. Kirk, "An Extensible Object-Oriented Mixed-Mode Functional Simulation System," *Proc. of the 22nd Design Automation Conference*, pp. 630-636, June 1985.

13. M. Lightner, et al, "CSIM: The Evolution of a Behavioral level Simulator from a Functional Simulator: Implementation Issues and Performance Measurements," *ICCAD-85*, pp. 350-352, Santa Clara, CA, Nov. 1985.

14. B. Crawford, "Design Rules Checking for Integrated Circuits Using Graphical Operators," *Computer Graphics, 9:1*, pp. 168-176, 1975.

15. C. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs," *Lambda, 1:3*, pp. 22-30, 4th Quarter 1980 .

16. M. Arnold and J. Ousterhout, "Lyra: A New Approach to Geometric Layout Rule Checking," *Proc. of the 19th Design Automation Conference*, pp. 530-536, June 1982.

17. R. Spickelmier and A. R. Newton, "Critic: A Knowledge-Based Program for Critiquing Circuit Designs," *Proc. of ICCD*, pp. 324-327, Rye Brook, NY, Oct. 1988.

18. H. Barrow, "VERIFY: A Program for Proving Correctness of Digital Hardware Designs," *Artificial Intelligence, 24:1-3*, pp. 437-491, Dec. 1084.

19. M. Gordon, "HOL-A Machine Oriented Formulation of Higher Order Logic," *Univ. of Cambridge Computer Lab., tech. report 68*, July 1985.

20. S. German and Y. Wang, "Formal Verification of Parameterized Hardware Designs," *Proc. of ICCD*, pp. 549-552, Oct. 1985.

21. T. Kirkpatrick and N. Clark, "PERT as an Aid to Logic Design," *IBM Jour. of Research and Development*, pp. 135-141, March 1966.

22. T. McWilliams, "Verification of Timing Constraints on Large Digital Systems," *Proc. of the 17th Design Automation Conference*, pp. 139-147, IEEE, 1980.

23. J. Ousterhout, "Crystal:A Timing Analyzer for NMOS VLSI Circuits," *Proc. of the third Caltech VLSI Conference*, pp. 57-59, 1983.

24. N. Jouppi, "Timing Analysis and Performance Improvement of MOS VLSI Designs," *IEEE Trans. on CAD of ICAS*, vol. Vol. CAD-6, No. 4, pp. 650-665, July 1987.

25. C. Mead and L. Conway, *Introduction To VLSI Systems*, Addison-Wesley, 1980.

26. G. Langdon, *Computer Design*, Computeach Press Inc. , 1982.

27. N. Weste and K. Eshraghian, *Principals of CMOS VLSI Design. A systems Perspective*, Addison Wesley, 1985.

28. L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison Wesley, 1985.

29. D. Hodges and H. Jackson, *Analysis and Design of Digital integrated Circuits*, McGraw-Hill, 1983.

30. N. Weiner, *Hummingbird*, University of California, Berkeley, Dec. 1987.

31. R. Krambeck, C. Lee, and H. Law, "High Speed Compact Circuits with CMOS," *IEEE Journal of Solid State Circuits*, vol. Vol. SC-17 No.3, pp. 614-619, 1982.

32. N. Gardner and H. De Man, "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures," *IEEE J. Solid-State Circuits*, pp. 261-266, 1983.

33. L. Hartung and J. Morgan, *PERT/PEP-A Dynamic Project Control Method*, Report 61-816-2005, IBM FSD Space Guidance Center, Owego, NY, 1961.

34. N. Loeber, "PERT for Small Projects," *Machine Design*, pp. 134-139, 1962.

35. M. Wold, "Design Verification and Performance Analysis," *Proc. of the 15th Design Automation Conference*, pp. 264-270, IEEE, Las Vegas, 1978.

36. T. Sasaki, A. Yamada, T. Aoyama, K. Hasegawa, S. Kato, and S. Sato, "Hierarchical Design Verification for Large Digital Systems," *Proc. of the 18th Design Automation Conference*, pp. 105-112, IEEE, Nachville, 1981.

37. R. Hitchcock, "Timing Verification and the Timing Analysis Program," *Proc. of the 19th Design Automation Conference*, pp. 594-604, IEEE, June 1982.

38. E. Tamura, K. Ogawa, and T. Nakano, "Path Delay Analysis for Hierarchical Building Block Layout System," *Proc. of the 20th Design Automation Conference*, pp. 403-410, IEEE, June 1983.

39. R. Saleh, J. Kleckner, and A. R. Newton, "Iterated Timing Analysis in SPLICE1," *Digest 1983 Int. Conf. on CAD, IEEE*, Santa Clara, CA, Sept. 1983.

40. J. White and A. Sangiovanni-Vincentelli, "RELAX2: A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits," *Proc. 1983 Int. Symp on Circ. and*

Sys., Newport Beach, May 1983.

41. T. Quarles, A. R. Newton, D. Pederson, and A. Sangiovanni-Vincentelli, *SPICE 3B1 User's Guide*, University of California, Berkeley, Apr. 27 1987.

42. D. Brand and V. Iyengar, "Timing Analysis using Functional Relationships," *ICCAD-86*, pp. 126-129, Santa Clara, CA, Nov. 1986.

43. J. Benkoski, E. Meersch, and H. De Man, "Efficient Algorithms for Solving the False Path problem in Timing Verification," *ICCAD-87*, pp. 44-47, Santa Clara, CA, Nov. 1987.

44. H. Ma, S. Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli, "Test Generation for Sequential Finite State Machines," *Digest 1987 Int. Conf. on CAD, IEEE*, pp. 288-291, Santa Clara, CA, Nov. 1987.

45. R. Lisanke, F. Frglez, A. de Geus, and D. Gregory, "Testability-Driven Random Pattern Generation," *Digest 1986 Int. Conf. on CAD, IEEE*, pp. 144-147, Santa Clara, CA, Nov. 1986.

46. A. de Geus, J. Reed, M. Rekhson, and G. Wikle, "IDA: Interconnect Delay Analysis for Integrated Circuits," *Proc. of the 21st Design Automation Conference*, pp. 536-541, IEEE, June 1984.

47. J. Shelly and D. Tryon, "Statistical Techniques of Timing Verification," *Proc. of the 20th Design Automation Conference*, pp. 396-402, IEEE, June 1983.

48. D. Wallace, "ATV: An Abstract Timing Verifier," *Proc. of the 25th Design Automation Conference*, pp. 154-159, IEEE, Anaheim, 1988.

49. D. Coelho and N. Chalapathy, "Timing Verification Using A Generâl Behavioral Simulator," , 1984.

50. D. Coelho and W. VanCleemput, "Helix: A tool for Multilevel Simulation of VLSI Systems," *The third International Conference on Semi-Custom IC's*, London, England, Nov. 1983.

51. A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading,MA, 1974.

52. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Inc, 1978.

53. R. Tarjan, *Data Structure and Network Algorithms*, Bell Laboratories, Englewood Cliffs, N.J., 1983.

54. P. Bottorff, "Computer Aids to Testing - An Overview," in *Computer Design Aids for VLSI Circuits*, P. Antognetti, D. Pederson and H. De Man, eds., Nato Advanced Study Series, Sijthoff & Noordhoff, Rockville, Maryland, 1981.

55. D. Knuth, *The Art of Computer programming, Vol. 1: Fundamental Algorithms, 2nd ed.*, Addison-Wesley, Inc, Reading,MA, 1973.

56. T. Szymanski, "LEADOUT: A Static Timing Analysis for MOS Circuits," *ICCAD-86*, pp. 130-133, Santa Clara, CA, Nov. 1986.

57. J. Cherry, "Pearl: A CMOS Timing Verifier," *Proc. of the 25th Design Automation Conference*, pp. 148-153, IEEE, Anaheim, 1988.

58. J. Kleckner, "Advanced Mixed-Mode Simulation Techniques," *UCB/ERL M84/48*, University of California, Berkeley, June 1984. Ph.D. Dissertation

59. Y. Kim, J. Kleckner, R. Saleh, and A. R. Newton, "Electrical-Logic Simulation," *Digest 1984 Int. Conf. on CAD, IEEE*, Santa Clara, CA, Nov 1984.

60. Y. Kim, "ELOGIC: A Relaxation-Based Switch-Level Simulation Technique," *UCB/ERL M86/2*, University of California, Berkeley, Dec 1986.

61. Y. Kim, S. Hwang, and A. R. Newton, "Electrical-Logic Simulation and Its Application," *IEEE Trans. on CAD of ICAS*, To be published in Jan. 1989.

62. C. Desoer and E. Kuh, *Basic Circuit Theory*, McGraw Hill, Inc, 1969.

63. L. Chua and P. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1975.

64. P. Penfield and J. Rubenstein, "Signal Delay in RC Tree Networks," *Proc. of the 18th Design Automation Conference*, pp. 613-617, IEEE, Nachville, June 1981.

65. D. Pilling and J. Skalnik, "A Circuit Model for Predicting Transient Delays in LSI Logic Systems," *Proc. of the 6th Asilomar Conference on Circuit and Systems*, pp.

424-428, 1972.

66. H. De Man, et al., "LOGMOS: A Transistor Oriented Logic Simulator With Assignable Delays," *International Conference on Circuits and Computers*, NY, 1982.

67. B. Chawla, H. Gummel , and P. Kozak, "MOTIS - An MOS timing simulator," *IEEE Trans. on Circ. and Sys.*, vol. CAS-22, pp. 901-909, Dec. 1975.

68. S. Fan, M. Hsueh, A. R. Newton, and D. Pederson, "MOTIS-C: A New Circuit Simulator for MOS LSI Circuits," *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 700-703, April 1977.

69. G. Arnout and H. De Man, "The Use of Threshold Functions and Boolean-Controlled Network Elements for Macromodelling of LSI Circuits," *IEEE J. of Solid-State Circuits*, vol. Vol. SC-13, pp. 326-332, June 1978.

70. A. R. Newton, "Techniques for the Simulation of Large-Scale Integrated Circuits," *Trans. IEEE*, vol. Vol. CAS-26, No.9, pp. 741-749, Sept. 1979.

71. D. Tsao and C. Chen, "A Fast Timing Simulation for Digital MOS Circuits," *Digest 1985 Int. Conf. on CAD, IEEE*, pp. 185 - 187, Santa Clara, CA, Nov. 1985.

72. A. R. Newton, "Timing, Logic, and Mixed-Mode Simulation for Large MOS Integrated Circuits," in *Computer Design Aids for VLSI Circuits*, P. Antognetti, D. Pederson and H. De Man, eds., Nato Advanced Study Series, Sijthoff & Noordhoff, Rockville, Maryland, 1981.

73. P. Stevens and G. Arnout, "BIMOS, an MOS oriented multi-level logic simulator," *Proc. of the 20th Design Automation Conference*, pp. 100-106, Miami Beach, Fl, 1983.

74. C. Terman, "RSIM - A Logic-Level Timing Simulator," *Proc. of the IEEE International Conference on Computer Design*, pp. 437-440, 1983.

75. A. de Geus, "SPECS: Simulation Program for Electronic Circuits and Systems," *Proc. 1984 Int. Symp. on Circ and Sys., IEEE*, pp. 534 - 537, Montreal, Canada, May 1984.

76. L. Vidigal, S. Nassif, and S. Director, "CINNAMON: Coupled Integration and Nodal Analysis of MOS Network," *Proc. of the 23rd Design Automation Conference*, pp. 179-185, 1986.

77. P. Odryna, "Simulator merges speed of logic verifiers with circuit-level accuracy," *Electronic Design*, pp. 97 - 105, Nov. 1985.

78. C. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1971.

79. G. Dahlquist and A. Bjorck, *Numerical Methods*, Prentice-Hall, Inc, 1974.

80. R. Saleh, "Nonlinear Relaxation Algorithm For Circuit Simulation," *UCB/ERL M87/21*, University of California, Berkeley, April 1987. Ph.D. Dissertation

81. G. Michelli and A. Sangiovanni-Vincentelli, "Characterization of Integration Algorithms for Timing Analysis of MOS VLSI Circuits," *Circuit Theory and Applications*, vol. Vol. 10, pp. 299-309, 1982.

82. G. Michelli, A. R. Newton, and A. Sangiovanni-Vincentelli, "Symmetric Displacement Algorithms for the Timing Analysis of Large Scale Circuits," *IEEE Trans. on CAD of ICAS*, vol. CAD-2 No.3, pp. 167-180, July 1983.

83. J. White, "The Multirate Integration Properties of Waveform Relaxation, with Applications to Circuit Simulation and Parappel Computation," *UCB/ERL M85/90*, University of California, Berkeley, Nov. 1985.

84. A. R. Newton, "The Simulation of Large-Scale Integrated Circuits," *Memo UCB/ERL M78/52*, University of California, Berkeley, July 1978. Ph.D. Dissertation.

85. G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 1983.

86. S. Szygenda and E. Thompson, "Digital Logic Simulation in a Time-Based, Table-Driven Environment. Part 1. Design Verification," *IEEE Computer*, pp. 24-36, March 1975.

87. E. Ulrich, "Event Manipulation for Discrete Simulations Requiring Large Numbers of Events," *Comm. of ACM*, pp. 777-785, Sept. 1978.

88. K. Sakallah, "Mixed Simulation of Electronic Integrated Circuits," *DRC-02-07-81*, Carnegie Mellon University, Nov. 1981.

89. E. Ince, *Ordinary Differential Equations*, Dover, New York, 1956.

90. L. Chua, C. Desoer, and E. Kuh, *Linear and Nonlinear Circuits*, McGraw Hill, Inc, 1987.

91. C. Ho, A. Ruehli, and P. Brennan, "The Modified Nodal Approach to Network Analysis," *Proc. IEEE ISCAS*, pp. 505-509, SF, CA, April 1974.

92. H. Shichman and D. Hodges, "Modeling and Simulation of Insulated Gate Field-Effect Transistor Switching Circuits," *IEEE Journ. on Solid State Circuits*, vol. Vol. SC-3, pp. 285-289, Sept. 1968.

93. A. R. Newton and D. Pederson, "Analysis Time, Accuracy and Memory Requirement Tradeoffs in SPICE2," *11th Asilomar Conference on Circuits, Systems and Computers*, pp. 6-9, Asilomar, CA, Nov. 1977.

94. J. Burns, A. R. Newton, and D. Pederson, "Active Device Table Look-up Models For Circuit Simulation," *Proc. 1983 Int. Symp. on Circ and Sys., IEEE*, May 1983.

95. T. Shima, S. Tsutomu, S. Moriyama, and H. Yamada, "Three-Dimensional Table Look-up MOSFET Model for Precise Circuit Simulation," *IEEE Journal of Solid-State Circuits*, vol. Vol. SC-17 No.3, pp. 449-453, June 1983.

96. T. Shima, H. Yamada, and Dang, "Table Look-Up MOSFET Modeling System Using a 2-D Device Simulator and Monotonic Piecewise Cubic Interpolation," *IEEE Trans. on CAD of ICAS*, vol. Vol. CAD-2, No.2, pp. 121-125, April 1983.

97. S. Hwang, Y. Kim, and A. R. Newton, *An Accurate Delay Modeling Technique for Switch-Level Timing Verification*, pp. 227-233, Las Vegas, Nevada, June 1986.

98. C. Marino, *Smalltalk on a RISC - CMOS Implementation*, University of California, May 1985. M.S. Report

99. N. Jouppi, "Derivation of Signal Flow Direction in MOS VLSI," *IEEE Trans. on CAD of ICAS*, vol. Vol. CAD-6, No. 3, pp. 480-490, May 1987.

100. H. Shin, "Electronic Switching Techniques for Fiber Optic Networks," *UCB/ERL M88/15*, University of California, Berkeley, Feb. 1988. Ph.D. Dissertation

101. A. Gupta, "ACE: A Circuit Extractor," *Proc. of the 20th Design Automation Conference*, pp. 721-725, 1983.

102. M. Hofmann and V. Lauther, "HEX: An Instruction Driven Approach to Feature Extraction," *Proc. of the 20th Design Automation Conference*, pp. 331-336, 1983.

103. Dan Fitzpatrick, "mextra - Manhattan Circuit Extractor," *Berkeley CAD Tools User's Manual*, University of California, Berkeley, 1983.

104. E. van Willigen and R. Nouta, "A VLSI Artwork Analysis System Using Dedicated Hardware and Syntactic Pattern Recognition," *Digest 1986 Int. Conf. on CAD, IEEE*, pp. 400-403, Santa Clara, CA, Nov. 1986.

105. M. Hill, et al., "Design Decisions in SPUR," *IEEE Computer*, vol. Vol. 19 No 10, pp. 8-24, Nov. 1986.

106. A. Vladimirescu, K. Zhang, A. R. Newton, D. Pederson, and A. Sangiovanni-Vincentelli, *SPICE Version 2G User's Guide*, University of California, Berkeley, August 1981.

107. A. Vladimirescu and S. Liu, "The Simulation of MOS Integrated Circuits Using SPICE2," *UCB/ERL M80/7*, University of California, Berkeley, Feb. 1980.

108. "NU MACHINE Technical Summary," *Texas Instruments*, Irvine, CA, 1982.

109. D. Jeong, et al., "A Memory Management Unit and Cache Controller for a Multiprocessor Workstation," *ISSCC*, Submitted.

110. J. Meyer, "MOS Models and Circuit Simulation," *RCA Review*, vol. 32, March 1972.

111. K. Sakallah, Y. Yen, and S. Greenberg, "The Meyer Model Revisited: Explaining and Correcting the Charge Non-Conservation Problem," *ICCAD-87*, pp. 204-207, Santa Clara, CA, Nov. 1987.

112. D. Lee, *Private communication*, Nov. 1988.

113. D. Harrison, P. Moore, R. Spickelmier, and A.R. Newton, "Data Management and Graphics Editing in the Berkeley Design Environment," *ICCAD-86*, pp. 24-27, Santa Clara, CA, Nov. 1986.

114. P. Moore, "The General Structure of OCT," in *Oct Tools Distribution 2.1*, ed. R. Spickelmier, ERL, University of California, Berkeley, March 1988.

115. D. Harrison, "Customizing VEM," in *Oct Tools Distribution 2.1*, ed. R. Spickelmier, ERL, University of California, Berkeley, March 1988.

APPENDIX 1

MOS MODEL PARAMETERS

This appendix contains MOS model parameters that were used for *E-TV* and SPICE to analyze the test circuits described in Section 5.13.

[1] MOS model parameters used for the analysis of the SOAR circuit.

[1.1] MOS1 parameters for *E-TV* and SPICE

```
* MOS1 parameters for SOAR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695
+  lambda=0.025 ld=0.24u
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514
+  lambda=0.0527 ld=0.51u
```

[1.2] MOS2 parameters for SPICE

```
* MOS2 parameters for SOAR
.model nmos nmos(level=2 tox=50n nsub=10e15 vto=0.93 xj=0.45u
+  ld=0.24u js=1.24e-4 pb=0.80 uo=381 ucrit=99e4 uexp=0.001
+  utra=0 lambda=0.025 cgbo=4.0e-10 tpg=1
+  cgdo=5.2e-10 cgso=5.2e-10 cj=3.2e-4 cjsw=9.0e-10
+  vmax=5.5e4 neff=1.0e-2 rsh=25 delta=1.47 nfs=3.73e11)
.model pmos pmos(level=2 tox=50n nsub=2.97e14 vto=-0.844 xj=0.0258u
+  ld=0.51u js=7.75e-5 pb=0.88 uo=100 ucrit=18500 uexp=0.145
+  gamma=0.723 lambda=0.0527 cgbo=4.0e-10 tpg=-1
+  cgdo=4.0e-10 cgso=4.0e-10 cj=2.0e-4 cjsw=4.0e-10
+  vmax=10e4 neff=.01 rsh=95 delta=2.19 nfs=1.62e12)
```

[2] MOS model parameters used for the analysis of the SPUR circuit.

[2.1] MOS1 parameters for *E-TV* and SPICE

```
* MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
```

[2.2] MOS2 parameters for SPICE

```
* MOS2 parameters for SPUR.
.model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
+ tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
+ js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*
.model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
+ tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
+ js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
```

APPENDIX 2

MOS1 MODEL PARAMETER ADJUSTMENT

Model parameters for the SOAR and SPUR circuits were available for the MOS2 model only. Among the five device parameters of the MOS1 model [106, 107] (*VTO*, *KP*, *GAMMA*, *PHI* and *LAMBDA*), all parameters except *KP* were used without adjustment. In this appendix, the method which is used to determine the value of *KP* for the MOS1 model from MOS2 model parameters is described. Since the process for SOAR and SPUR is different, their *KP* was determined separately. A key idea of the method is to adjust the value of *KP* of MOS1 to match the dc transfer curve of an inverter shown in Figure A2.1 and to match the value of current flowing through the inverter to the input change, as much as possible using the MOS1 and MOS2 models. Since the value of *KP* to be determined for MOS1 depends on the transistor size used, the average channel length and width of transistors in a circuit was used. The transistor sizes used are illustrated in Table A2.1 The dc transfer curve and the value of current (*I*) are compared for both circuits in Figures A2.2, A2.3, A2.4 and A2.5, using the model parameters presented in Appendix 1.

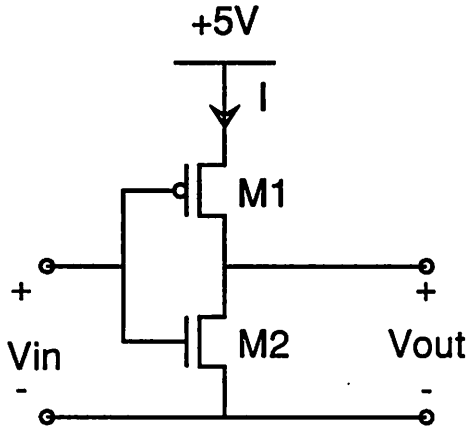


Figure A2.1 An Inverter Used To Determine The Values of *KP* of The MOS1 Model From MOS2 Model Parameters

	SOAR	SPUR
M1	14μm/4.4μm	11μm/1.6μm
M2	10μm/3μm	6.4μm/1.6μm

Table A2.1 The Size of Transistors (W/L) Used To Determine The Values of *KP* of The MOS1 Model From MOS2 Model Parameters

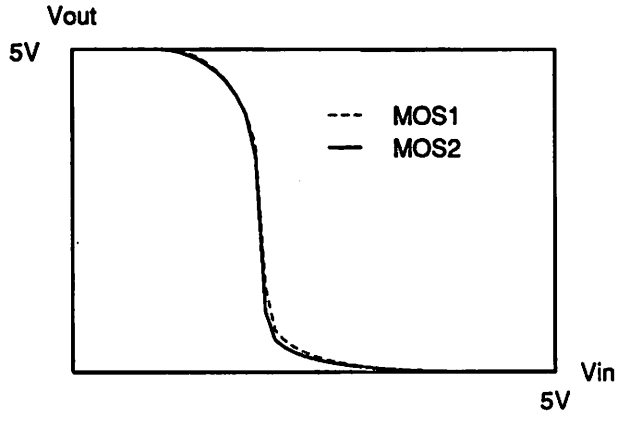


Figure A2.2 DC Transfer Curve of The Inverter in Figure A2.1

Using The MOS1 and MOS2 Models for SOAR

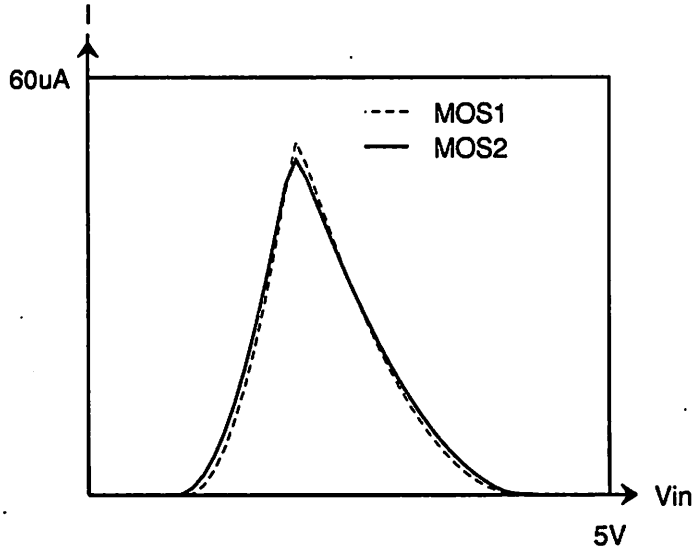


Figure A2.3 Current Through The Inverter Shown in Figure A2.1

Using The MOS1 and MOS2 Models for SOAR

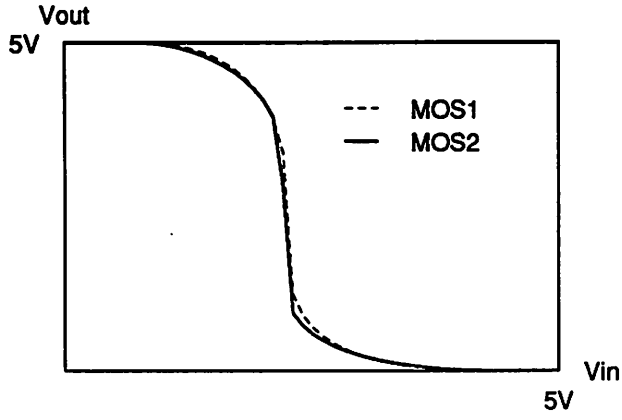


Figure A2.4 DC Transfer Curve of The Inverter in Figure A2.1

Using The MOS1 and MOS2 Models for SPUR

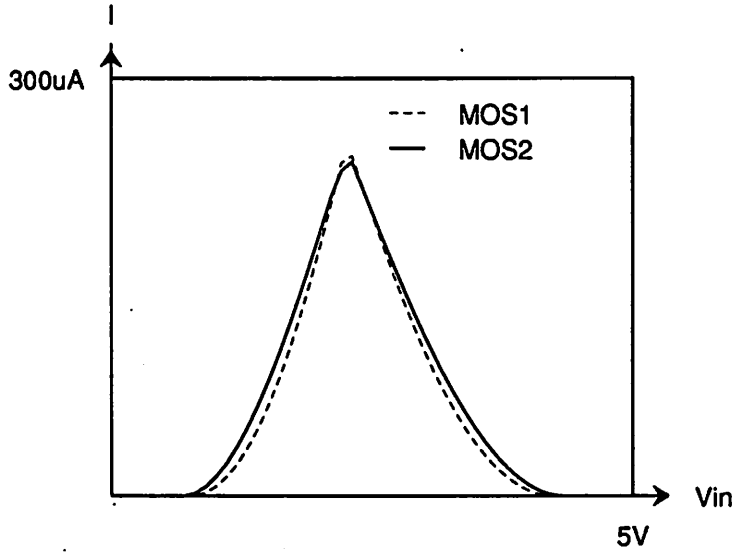


Figure A2.5 Current Through The Inverter Shown in Figure A2.1

Using The MOS1 and MOS2 Models for SPUR

APPENDIX 3

SPICE INPUT DATA

FOR THE CRITICAL PATHS OF EXAMPLE CIRCUITS

This appendix contains the input data used by SPICE for the analysis of the SOAR and SPUR examples of Chapter 6.

[1] SPICE input data for the critical path of the SOAR ALU ($\phi 3(R)$–$\phi 3(F)$ segment)

```
* SPICE input data for the critical path of the SOAR ALU
* (φ3(R)–φ3(F) segment)
*
.opt acct opts nomod nopage limpts=500
.tran 1ns 150ns
.print tran v(1016)
*
*************************************************************************
*           MOS1 parameters for SOAR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695
+ lambda=0.025 ld=0.24u
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514
+ lambda=0.0527 ld=0.51u
*************************************************************************
*           MOS2 parameters for SOAR
* .model nmos nmos(level=2 tox=50n nsub=10e15 vto=0.93 xj=0.45u
* + ld=0.24u js=1.24e-4 pb=0.80 uo=381 ucrit=99e4 uexp=0.001
* + utra=0 lambda=0.025 cgbo=4.0e-10 tpg=1
* + cgdo=5.2e-10 cgso=5.2e-10 cj=3.2e-4 cjsw=9.0e-10
* + vmax=5.5e4 neff=1.0e-2 rsh=25 delta=1.47 nfs=3.73e11)
* .model pmos pmos(level=2 tox=50n nsub=2.97e14 vto=-0.844 xj=0.0258u
* + ld=0.51u js=7.75e-5 pb=0.88 uo=100 ucrit=18500 uexp=0.145
* + gamma=0.723 lambda=0.0527 cgbo=4.0e-10 tpg=-1
* + cgdo=4.0e-10 cgso=4.0e-10 cj=2.0e-4 cjsw=4.0e-10
* + vmax=10e4 neff=.01 rsh=95 delta=2.19 nfs=1.62e12)
* .options method=gear gmin=10p reltol=0.01 abstol=10p vntol=10u
* .options chgtol=1p
*************************************************************************
*
vdd 1 0 dc 5
vphi_3 15 0 pulse 0 5 0 1n 1n 100n 200n
*
c45 45 0 1ff
c1710 44 0 77ff
c1712 707 0 78ff
c1718 15 0 62ff
c1719 15 0 63ff
c1723 15 0 67ff
c1728 15 0 61ff
c1735 762 0 122ff
c1751 67 0 77ff
c1758 778 0 79ff
c1762 15 0 62ff
c1764 15 0 63ff
```

c1768 15 0 67ff
c1770 111 0 63ff
c1774 15 0 61ff
c1782 108 0 122ff
c1784 842 0 72ff
c1802 860 0 78ff
c1805 147 0 79ff
c1809 15 0 62ff
c1811 15 0 63ff
c1815 15 0 67ff
c1821 15 0 61ff
c1829 864 0 122ff
c1831 201 0 72ff
c1849 222 0 78ff
c1852 888 0 79ff
c1856 15 0 62ff
c1858 15 0 63ff
c1862 15 0 67ff
c1868 15 0 61ff
c1876 215 0 122ff
c1878 947 0 70ff
c1897 967 0 79ff
c1900 257 0 79ff
c1904 15 0 62ff
c1906 15 0 63ff
c1910 15 0 66ff
c1916 15 0 61ff
c1925 1016 0 89ff
c1926 303 0 72ff
c1933 15 0 50ff
c1948 326 0 78ff
c1952 15 0 62ff
c1954 15 0 67ff
c1962 388 0 122ff
c1972 408 0 79ff
c1975 15 0 63ff
c1978 15 0 61ff
c1985 412 0 72ff
c1996 445 0 77ff
c1999 15 0 62ff
c2001 15 0 67ff
c2009 500 0 158ff
c2032 520 0 79ff
c2035 15 0 63ff
c2038 15 0 61ff
c2045 572 0 72ff
c2056 597 0 78ff
c2059 15 0 62ff
c2061 15 0 67ff

```
c2069 656 0 122ff
c2079 674 0 79ff
c2082 15 0 63ff
c2085 15 0 61ff
c2092 678 0 72ff
*
m59 0 15 45 0 nmos w = 4u l = 3u
m41 45 40 44 0 nmos w = 4u l = 3u
v40 40 0 dc 5
m110 44 98 68 0 nmos w = 4u l = 3u
v98 98 0 dc 5
m68 68 63 67 0 nmos w = 4u l = 3u
v63 63 0 dc 5
m251 67 113 111 0 nmos w = 4u l = 3u
v113 113 0 dc 5
m161 1 111 128 1 pmos w = 7u l = 3u
m152 0 128 127 0 nmos w = 7u l = 3u
m151 127 19 108 0 nmos w = 7u l = 3u
v19 19 0 dc 5
m195 144 108 1 1 pmos w = 7u l = 3u
m207 148 144 147 0 nmos w = 4u l = 3u
m301 205 147 1 1 pmos w = 7u l = 3u
m285 201 205 196 0 nmos w = 4u l = 3u
m343 235 201 1 1 pmos w = 7u l = 3u
m355 237 235 222 0 nmos w = 4u l = 3u
m323 223 222 1 1 pmos w = 7u l = 3u
m425 215 223 286 0 nmos w = 4u l = 3u
m487 323 215 1 1 pmos w = 7u l = 3u
m499 259 323 257 0 nmos w = 4u l = 3u
m467 311 257 1 1 pmos w = 7u l = 3u
m451 303 311 295 0 nmos w = 4u l = 3u
m511 335 303 1 1 pmos w = 7u l = 3u
m523 337 335 326 0 nmos w = 4u l = 3u
m616 392 326 1 1 pmos w = 7u l = 3u
m596 388 392 383 0 nmos w = 4u l = 3u
m659 423 388 1 1 pmos w = 7u l = 3u
m671 425 423 408 0 nmos w = 4u l = 3u
m639 409 408 1 1 pmos w = 7u l = 3u
m749 412 409 478 0 nmos w = 4u l = 3u
m808 441 412 1 1 pmos w = 7u l = 3u
m697 446 441 445 0 nmos w = 4u l = 3u
m788 507 445 1 1 pmos w = 7u l = 3u
m768 500 507 495 0 nmos w = 4u l = 3u
m929 521 500 1 1 pmos w = 7u l = 3u
m815 522 521 520 0 nmos w = 4u l = 3u
m909 579 520 1 1 pmos w = 7u l = 3u
m893 572 579 567 0 nmos w = 4u l = 3u
m955 606 572 1 1 pmos w = 7u l = 3u
m967 608 606 597 0 nmos w = 4u l = 3u
```

```
m1059 659 597 1 1 pmos w = 7u l = 3u
m1039 656 659 653 0 nmos w = 4u l = 3u
m1096 688 656 1 1 pmos w = 7u l = 3u
m1108 690 688 674 0 nmos w = 4u l = 3u
m1076 675 674 1 1 pmos w = 7u l = 3u
m1185 678 675 740 0 nmos w = 4u l = 3u
m1242 703 678 1 1 pmos w = 7u l = 3u
m1131 708 703 707 0 nmos w = 4u l = 3u
m1222 769 707 1 1 pmos w = 7u l = 3u
m1202 762 769 757 0 nmos w = 4u l = 3u
m1259 788 762 1 1 pmos w = 7u l = 3u
m1271 790 788 778 0 nmos w = 4u l = 3u
m1360 846 778 1 1 pmos w = 7u l = 3u
m1344 842 846 837 0 nmos w = 4u l = 3u
m1404 876 842 1 1 pmos w = 7u l = 3u
m1416 878 876 860 0 nmos w = 4u l = 3u
m1384 861 860 1 1 pmos w = 7u l = 3u
m1487 864 861 925 0 nmos w = 4u l = 3u
m1544 889 864 1 1 pmos w = 7u l = 3u
m1434 890 889 888 0 nmos w = 4u l = 3u
m1524 954 888 1 1 pmos w = 7u l = 3u
m1508 947 954 942 0 nmos w = 4u l = 3u
m1565 976 947 1 1 pmos w = 7u l = 3u
m1577 978 976 967 0 nmos w = 4u l = 3u
m1656 1019 967 1 1 pmos w = 7u l = 3u
m1637 1016 1019 1011 0 nmos w = 4u l = 3u
.ic v(1016)=5 v(1019)=0 v(967)=5 v(976)=0 v(947)=5 v(954)=0
.ic v(888)=5 v(889)=0 v(864)=5 v(861)=0 v(860)=5 v(876)=0
.ic v(842)=5 v(846)=0 v(778)=5 v(788)=0 v(762)=5 v(769)=0
.ic v(707)=5 v(703)=0 v(678)=5 v(675)=0 v(674)=5 v(688)=0
.ic v(656)=5 v(659)=0 v(597)=5 v(606)=0 v(572)=5 v(579)=0
.ic v(520)=5 v(521)=0 v(500)=5 v(507)=0 v(445)=5 v(441)=0
.ic v(412)=5 v(409)=0 v(408)=5 v(423)=0 v(388)=5 v(392)=0
.ic v(326)=5 v(335)=0 v(303)=5 v(311)=0 v(257)=5 v(323)=0
.ic v(215)=5 v(223)=0 v(222)=5 v(235)=0 v(201)=5 v(205)=0
.ic v(147)=5 v(144)=0 v(108)=5 v(127)=5 v(128)=0 v(111)=5
.ic v(67)=5  v(68)=5  v(44)=5  v(45)=5  v(15)=0
*
* evaluation logic of n-type dynamic logic gates
ma148 148 100148 101148 0 nmos w = 4u l = 3u
v148 100148 0 dc 5
c148 148 0 1ff
.ic v(148)=5
c101148 101148 0 1ff
.ic v(101148)=5
mb148 101148 15 0 0 nmos w = 4u l = 3u
ma196 196 100196 101196 0 nmos w = 4u l = 3u
v196 100196 0 dc 5
c196 196 0 1ff
```

```
.ic v(196)=5
c101196 101196 0 1ff
.ic v(101196)=5
mb196 101196 15 0 0 nmos w = 4u l = 3u
ma237 237 100237 101237 0 nmos w = 4u l = 3u
v237 100237 0 dc 5
c237 237 0 1ff
.ic v(237)=5
c101237 101237 0 1ff
.ic v(101237)=5
mb237 101237 15 0 0 nmos w = 4u l = 3u
ma286 286 100286 101286 0 nmos w = 4u l = 3u
v286 100286 0 dc 5
c286 286 0 1ff
.ic v(286)=5
c101286 101286 0 1ff
.ic v(101286)=5
mb286 101286 15 0 0 nmos w = 4u l = 3u
ma259 259 100259 101259 0 nmos w = 4u l = 3u
v259 100259 0 dc 5
c259 259 0 1ff
.ic v(259)=5
c101259 101259 0 1ff
.ic v(101259)=5
mb259 101259 15 0 0 nmos w = 4u l = 3u
ma295 295 100295 101295 0 nmos w = 4u l = 3u
v295 100295 0 dc 5
c295 295 0 1ff
.ic v(295)=5
c101295 101295 0 1ff
.ic v(101295)=5
mb295 101295 15 0 0 nmos w = 4u l = 3u
ma337 337 100337 101337 0 nmos w = 4u l = 3u
v337 100337 0 dc 5
c337 337 0 1ff
.ic v(337)=5
c101337 101337 0 1ff
.ic v(101337)=5
mb337 101337 15 0 0 nmos w = 4u l = 3u
ma383 383 100383 101383 0 nmos w = 4u l = 3u
v383 100383 0 dc 5
c383 383 0 1ff
.ic v(383)=5
c101383 101383 0 1ff
.ic v(101383)=5
mb383 101383 15 0 0 nmos w = 4u l = 3u
ma425 425 100425 101425 0 nmos w = 4u l = 3u
v425 100425 0 dc 5
c425 425 0 1ff
```

```
.ic v(425)=5
c101425 101425 0 1ff
.ic v(101425)=5
mb425 101425 15 0 0 nmos w = 4u l = 3u
ma478 478 100478 101478 0 nmos w = 4u l = 3u
v478 100478 0 dc 5
c478 478 0 1ff
.ic v(478)=5
c101478 101478 0 1ff
.ic v(101478)=5
mb478 101478 15 0 0 nmos w = 4u l = 3u
ma446 446 100446 101446 0 nmos w = 4u l = 3u
v446 100446 0 dc 5
c446 446 0 1ff
.ic v(446)=5
c101446 101446 0 1ff
.ic v(101446)=5
mb446 101446 15 0 0 nmos w = 4u l = 3u
ma495 495 100495 101495 0 nmos w = 4u l = 3u
v495 100495 0 dc 5
c495 495 0 1ff
.ic v(495)=5
c101495 101495 0 1ff
.ic v(101495)=5
mb495 101495 15 0 0 nmos w = 4u l = 3u
ma522 522 100522 101522 0 nmos w = 4u l = 3u
v522 100522 0 dc 5
c522 522 0 1ff
.ic v(522)=5
c101522 101522 0 1ff
.ic v(101522)=5
mb522 101522 15 0 0 nmos w = 4u l = 3u
ma567 567 100567 101567 0 nmos w = 4u l = 3u
v567 100567 0 dc 5
c567 567 0 1ff
.ic v(567)=5
c101567 101567 0 1ff
.ic v(101567)=5
mb567 101567 15 0 0 nmos w = 4u l = 3u
ma608 608 100608 101608 0 nmos w = 4u l = 3u
v608 100608 0 dc 5
c608 608 0 1ff
.ic v(608)=5
c101608 101608 0 1ff
.ic v(101608)=5
mb608 101608 15 0 0 nmos w = 4u l = 3u
ma653 653 100653 101653 0 nmos w = 4u l = 3u
v653 100653 0 dc 5
c653 653 0 1ff
```

```
.ic v(653)=5
c101653 101653 0 1ff
.ic v(101653)=5
mb653 101653 15 0 0 nmos w = 4u l = 3u
ma690 690 100690 101690 0 nmos w = 4u l = 3u
v690 100690 0 dc 5
c690 690 0 1ff
.ic v(690)=5
c101690 101690 0 1ff
.ic v(101690)=5
mb690 101690 15 0 0 nmos w = 4u l = 3u
ma740 740 100740 101740 0 nmos w = 4u l = 3u
v740 100740 0 dc 5
c740 740 0 1ff
.ic v(740)=5
c101740 101740 0 1ff
.ic v(101740)=5
mb740 101740 15 0 0 nmos w = 4u l = 3u
ma708 708 100708 101708 0 nmos w = 4u l = 3u
v708 100708 0 dc 5
c708 708 0 1ff
.ic v(708)=5
c101708 101708 0 1ff
.ic v(101708)=5
mb708 101708 15 0 0 nmos w = 4u l = 3u
ma757 757 100757 101757 0 nmos w = 4u l = 3u
v757 100757 0 dc 5
c757 757 0 1ff
.ic v(757)=5
c101757 101757 0 1ff
.ic v(101757)=5
mb757 101757 15 0 0 nmos w = 4u l = 3u
ma790 790 100790 101790 0 nmos w = 4u l = 3u
v790 100790 0 dc 5
c790 790 0 1ff
.ic v(790)=5
c101790 101790 0 1ff
.ic v(101790)=5
mb790 101790 15 0 0 nmos w = 4u l = 3u
ma837 837 100837 101837 0 nmos w = 4u l = 3u
v837 100837 0 dc 5
c837 837 0 1ff
.ic v(837)=5
c101837 101837 0 1ff
.ic v(101837)=5
mb837 101837 15 0 0 nmos w = 4u l = 3u
ma878 878 100878 101878 0 nmos w = 4u l = 3u
v878 100878 0 dc 5
c878 878 0 1ff
```

```
.ic v(878)=5
c101878 101878 0 1ff
.ic v(101878)=5
mb878 101878 15 0 0 nmos w = 4u l = 3u
ma925 925 100925 101925 0 nmos w = 4u l = 3u
v925 100925 0 dc 5
c925 925 0 1ff
.ic v(925)=5
c101925 101925 0 1ff
.ic v(101925)=5
mb925 101925 15 0 0 nmos w = 4u l = 3u
ma890 890 100890 101890 0 nmos w = 4u l = 3u
v890 100890 0 dc 5
c890 890 0 1ff
.ic v(890)=5
c101890 101890 0 1ff
.ic v(101890)=5
mb890 101890 15 0 0 nmos w = 4u l = 3u
ma942 942 100942 101942 0 nmos w = 4u l = 3u
v942 100942 0 dc 5
c942 942 0 1ff
.ic v(942)=5
c101942 101942 0 1ff
.ic v(101942)=5
mb942 101942 15 0 0 nmos w = 4u l = 3u
ma978 978 100978 101978 0 nmos w = 4u l = 3u
v978 100978 0 dc 5
c978 978 0 1ff
.ic v(978)=5
c101978 101978 0 1ff
.ic v(101978)=5
mb978 101978 15 0 0 nmos w = 4u l = 3u
ma1011 1011 1001011 1011011 0 nmos w = 4u l = 3u
v1011 1001011 0 dc 5
c1011 1011 0 1ff
.ic v(1011)=5
c1011011 1011011 0 1ff
.ic v(1011011)=5
mb1011 1011011 15 0 0 nmos w = 4u l = 3u
* missing cap
c1019 1019 0 1ff
c976 976 0 1ff
c954 954 0 1ff
c889 889 0 1ff
c861 861 0 1ff
c876 876 0 1ff
c846 846 0 1ff
c788 788 0 1ff
c769 769 0 1ff
```

```
c703 703 0 1ff
c675 675 0 1ff
c688 688 0 1ff
c659 659 0 1ff
c606 606 0 1ff
c579 579 0 1ff
c521 521 0 1ff
c507 507 0 1ff
c441 441 0 1ff
c409 409 0 1ff
c423 423 0 1ff
c392 392 0 1ff
c335 335 0 1ff
c311 311 0 1ff
c323 323 0 1ff
c223 223 0 1ff
c235 235 0 1ff
c205 205 0 1ff
c144 144 0 1ff
c127 127 0 1ff
c128 128 0 1ff
c68 68 0 1ff
.end
```

[2] SPICE input data for the critical path of the SOAR circuit, excluding the ALU

[2.1] $\phi 3(R)$–$\phi 3(F)$ segment (SOAR)

```
* SPICE input data for the critical path of the SOAR circuit, excluding the ALU
* φ3(R)–φ3(F) segment
*
.tran 1n 200n
.print tran v(9298)
*
**********************************************************************
*            MOS1 parameters for SOAR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695
+ lambda=0.025 ld=0.24u
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514
+ lambda=0.0527 ld=0.51u
**********************************************************************
*            MOS2 parameters for SOAR
* .model nmos nmos(level=2 tox=50n nsub=10e15 vto=0.93 xj=0.45u
* + ld=0.24u js=1.24e-4 pb=0.80 uo=381 ucrit=99e4 uexp=0.001
* + utra=0 lambda=0.025 cgbo=4.0e-10 tpg=1
* + cgdo=5.2e-10 cgso=5.2e-10 cj=3.2e-4 cjsw=9.0e-10
* + vmax=5.5e4 neff=1.0e-2 rsh=25 delta=1.47 nfs=3.73e11)
* .model pmos pmos(level=2 tox=50n nsub=2.97e14 vto=-0.844 xj=0.0258u
* + ld=0.51u js=7.75e-5 pb=0.88 uo=100 ucrit=18500 uexp=0.145
* + gamma=0.723 lambda=0.0527 cgbo=4.0e-10 tpg=-1
* + cgdo=4.0e-10 cgso=4.0e-10 cj=2.0e-4 cjsw=4.0e-10
* + vmax=10e4 neff=.01 rsh=95 delta=2.19 nfs=1.62e12)
**********************************************************************
*
vdd 111111 0 dc 5
* clock skew is 5.1n for rising
vphi_3 11073 0 pulse 0 5 4.6n 1n 1n 1500n 2000n
* clock skew is 2.3n for falling
vphi_3_bar 11284 0 pulse 5 0 1.8n 1n 1n 1500n 2000n
.ic v(9298)=5 v(9075)=5 v(9252)=5 v(9253)=0 v(5345)=5 v(10167)=0
.ic v(10540)=0 v(10793)=5 v(12184)=0 v(12291)=5 v(12335)=5 v(12334)=5
.ic v(12331)=0 v(12333)=5 v(12330)=0 v(12328)=5 v(12286)=0 v(12508)=0
.ic v(12621)=0 v(12620)=5 v(12535)=0 v(12533)=5 v(12491)=0 v(12400)=0
.ic v(12594)=0 v(12595)=0 v(12381)=5
*
c12594 12594 0 0.1ff
c12595 12595 0 0.1ff
```

```
c9252 9252 0 0.1ff
c9253 9253 0 0.1ff
c34493 12491 0 64ff
c34660 12184 0 65ff
c34936 12334 0 95ff
c34939 12333 0 78ff
c34943 12328 0 95ff
c34997 12331 0 55ff
c34998 12330 0 55ff
c35398 12335 0 70ff
c35448 12621 0 188ff
c35449 12620 0 212ff
c35958 12291 0 76ff
c36157 10540 0 86ff
c36906 12286 0 926ff
c36958 9075 0 1251ff
c37113 12535 0 295ff
c37446 5345 0 286ff
c37601 10167 0 51ff
c37945 12400 0 153ff
c37977 9298 0 73ff
c38064 12381 0 225ff
c38625 12508 0 221ff
c38638 10793 0 987ff
c38684 12533 0 73ff
*
m24183 9075 5781 9298 111111 pmos w = 4u l = 3u
v5781 5781 0 dc 0
m24248 9075 5284 9298 0 nmos w = 4u l = 3u
v5284 5284 0 dc 5
m23932 9075 5346 9252 0 nmos w = 7u l = 3u
v5346 5346 0 dc 5
m23783 0 9253 9252 0 nmos w = 15u l = 3u
m24027 9253 119252 111111 111111 pmos w = 4u l = 8u
v9252 119252 0 dc 0
m23933 9253 5345 9255 0 nmos w = 7u l = 3u
c37673 9255 0 1494
.ic v(9255)=0
m24265 0 9099 9255 0 nmos w = 26u l = 3u
v9099 9099 0 dc 5
m26511 0 10167 5345 0 nmos w = 31u l = 3u
m27491 10167 10541 10540 0 nmos w = 4u l = 3u
v10541 10541 0 dc 5
m28143 10540 10793 111111 111111 pmos w = 6u l = 3u
m32416 10793 12184 0 0 nmos w = 4u l = 3u
m33319 12184 12291 111111 111111 pmos w = 11u l = 3u
m33289 12291 12473 12335 0 nmos w = 4u l = 3u
v12473 12473 0 dc 5
m33304 12291 12475 12335 111111 pmos w = 4u l = 3u
```

```
v12475 12475 0 dc 0
m32856 12335 12292 12334 111111 pmos w = 4u l = 3u
v12292 12292 0 dc 0
m33027 12334 12331 0 0 nmos w = 4u l = 3u
m32961 12331 12333 111111 111111 pmos w = 8u l = 3u
m33026 0 12330 12333 0 nmos w = 4u l = 3u
m32958 12330 12328 111111 111111 pmos w = 8u l = 3u
m33061 12328 12286 0 0 nmos w = 4u l = 3u
m33762 12286 12610 12508.0 nmos w = 4u l = 3u
v12610 12610 0 dc 5
m33850 111111 12621 12508 0 nmos w = 4u l = 3u
m33839 111111 12620 12508 111111 pmos w = 7u l = 3u
m33849 111111 12620 12621 111111 pmos w = 58.9u l = 2.8u
m33863 12620 12535 0 0 nmos w = 27u l = 3u
m33469 111111 12422 12552 111111 pmos w = 27u l = 3u
v12422 12422 0 dc 0
.ic v(12552)= 5
c12552 12552 0 0.1ff
m33468 12552 12533 12535 111111 pmos w = 27u l = 3u
m33430 0 12491 12533 0 nmos w = 4u l = 3u
m33396 12400 12502 12491 0 nmos w = 4u l = 3u
v12502 12502 0 dc 5
m33465 12400 12549 12491 111111 pmos w = 7u l = 3u
v12549 12549 0 dc 0
m33652 12594 10371 12400 111111 pmos w = 24u l = 3u
v10371 10371 0 dc 0
m33653 12595 12422 12594 111111 pmos w = 24u l = 3u
m33654 111111 12381 12595 111111 pmos w = 24u l = 3u
*
m33089 0 11667 12381 0 nmos w = 4u l = 3u
.ic v(12381)=5
m31273 11667 11073 11666 0 nmos w = 4u l = 3u
m30688 11667 11284 11666 111111 pmos w = 7u l = 3u
.ic v(11667)=0
c11667 11667 0 250ff
* 11073 is phi3 and 11284 is phi3-bar
m31274 11655 11601 11666 0 nmos w = 4u l = 3u
m30689 11655 11668 11666 111111 pmos w = 7u l = 3u
.ic v(11666)=5
c11666 11666 0 0.1ff
vg11601 11601 0 dc 5
vg11668 11668 0 dc 0
m30678 11654 11492 11655 111111 pmos w = 4u l = 3u
.ic v(11655)=5
c36920 11655 0 113ff
v11492 11492 0 dc 0
m30715 11654 11694 11653 111111 pmos w = 4u l = 3u
.ic v(11654)=5
c11654 11654 0 0.1ff
```

```
v11694 11694 0 dc 0
m30676 11652 11592 11653 111111 pmos w = 4u l = 3u
.ic v(11653)=5
c11653 11653 0 0.1ff
v11592 11592 0 dc 0
m30675 11652 10379 11651 111111 pmos w = 4u l = 3u
.ic v(11652)=5
c11652 11652 0 0.1ff
v10379 10379 0 dc 0
m30674 11650 11591 11651 111111 pmos w = 4u l = 3u
.ic v(11651)=5
c11651 11651 0 0.1ff
v11591 11591 0 dc 0
m30673 11650 10312 111111 111111 pmos w = 4u l = 3u
.ic v(11650)=5
c11650 11650 0 0.1ff
v10312 10312 0 dc 0
.end
```

[2.2] $\phi 2(R)$–$\phi 3(F)$ segment (SOAR)

```
* SPICE input data for the critical path of the SOAR circuit, excluding the ALU
* phi2(R)-phi3(F) segment
*
.tran 1n 100n
.print tran v(9298)
*
*********************************************************************
*           MOS1 parameters for SOAR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695
+ lambda=0.025 ld=0.24u
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514
+ lambda=0.0527 ld=0.51u
*********************************************************************
*           MOS2 parameters for SOAR
* .model nmos nmos(level=2 tox=50n nsub=10e15 vto=0.93 xj=0.45u
* + ld=0.24u js=1.24e-4 pb=0.80 uo=381 ucrit=99e4 uexp=0.001
* + utra=0 lambda=0.025 cgbo=4.0e-10 tpg=1
* + cgdo=5.2e-10 cgso=5.2e-10 cj=3.2e-4 cjsw=9.0e-10
* + vmax=5.5e4 neff=1.0e-2 rsh=25 delta=1.47 nfs=3.73e11)
* .model pmos pmos(level=2 tox=50n nsub=2.97e14 vto=-0.844 xj=0.0258u
* + ld=0.51u js=7.75e-5 pb=0.88 uo=100 ucrit=18500 uexp=0.145
* + gamma=0.723 lambda=0.0527 cgbo=4.0e-10 tpg=-1
* + cgdo=4.0e-10 cgso=4.0e-10 cj=2.0e-4 cjsw=4.0e-10
* + vmax=10e4 neff=.01 rsh=95 delta=2.19 nfs=1.62e12)
*********************************************************************
*
* clock skew is 2n for rising, 2.34n for falling transitions.
vphi1 12285 0 pulse 0 5 1.5n 1n 1n 100n 200n
* clock skew is 1.9n for rising, 1.43n for falling transitions.
vphi1_bar 12295 0 pulse 5 0 0.93n 1n 1n 100n 200n
vdd 111111 0 dc 5
.ic v(9298)=5 v(9075)=5 v(9252)=5 v(9253)=0 v(5345)=0 v(10167)=5
.ic v(10540)=5 v(10802)=5 v(10793)=0 v(12184)=5 v(12291)=0
c37977 9298 0 73ff
c36958 9075 0 1251ff
c9252 9252 0 0.1ff
c9253 9253 0 0.1ff
c37446 5345 0 286ff
c37601 10167 0 51ff
c36157 10540 0 86ff
c10802 10802 0 0.1ff
c38638 10793 0 987ff
c34660 12184 0 65ff
c35958 12291 0 76ff
```

```
*
m24248 9075 5284 9298 0 nmos w = 4u l = 3u
v5284 5284 0 dc 5
m24183 9075 5781 9298 111111 pmos w = 4u l = 3u
v5781 5781 0 dc 0
m23932 9075 5346 9252 0 nmos w = 7u l = 3u
v5346 5346 0 dc 5
m23783 0 9253 9252 0 nmos w = 15u l = 3u
m23933 9253 5345 9255 0 nmos w = 7u l = 3u
m23973 111111 5272 9255 111111 pmos w = 8u l = 3u
c37673 9255 0 1494
.ic v(9255)=5
v5272 5272 0 dc 0
m27028 111111 10167 5345 111111 pmos w = 68u l = 3u
m27491 10167 10541 10540 0 nmos w = 4u l = 3u
v10541 10541 0 dc 5
m28215 10802 10785 10540 0 nmos w = 7u l = 3u
v10785 10785 0 dc 5
m28293 0 10793 10802 0 nmos w = 7u l = 3u
m32415 10793 12184 111111 111111 pmos w = 8u l = 3u
m33329 12184 12291 0 0 nmos w = 11u l = 3u
m32765 12292 12285 12291 0 nmos w = 4u l = 3u
m32793 12292 12295 12291 111111 pmos w = 4u l = 3u
c37989 12292 0 127ff
m32821 111111 12314 12292 111111 pmos w = 23u l = 3u
v12314 12314 0 dc 0
.ic v(12292)=5
.end
```

[2.3] $\phi 1(R)$–$\phi 1(F)$ segment (SOAR)

```
* SPICE input data for the critical path of the SOAR circuit, excluding the ALU
* phi1(R)-phi1(F) segment
*
.tran 2n 300n
.print tran v(4785)
*
*********************************************************************
*           MOS1 parameters for SOAR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695
+ lambda=0.025 ld=0.24u
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514
+ lambda=0.0527 ld=0.51u
*********************************************************************
*           MOS2 parameters for SOAR
* .model nmos nmos(level=2 tox=50n nsub=10e15 vto=0.93 xj=0.45u
* + ld=0.24u js=1.24e-4 pb=0.80 uo=381 ucrit=99e4 uexp=0.001
* + utra=0 lambda=0.025 cgbo=4.0e-10 tpg=1
* + cgdo=5.2e-10 cgso=5.2e-10 cj=3.2e-4 cjsw=9.0e-10
* + vmax=5.5e4 neff=1.0e-2 rsh=25 delta=1.47 nfs=3.73e11)
* .model pmos pmos(level=2 tox=50n nsub=2.97e14 vto=-0.844 xj=0.0258u
* + ld=0.51u js=7.75e-5 pb=0.88 uo=100 ucrit=18500 uexp=0.145
* + gamma=0.723 lambda=0.0527 cgbo=4.0e-10 tpg=-1
* + cgdo=4.0e-10 cgso=4.0e-10 cj=2.0e-4 cjsw=4.0e-10
* + vmax=10e4 neff=.01 rsh=95 delta=2.19 nfs=1.62e12)
*********************************************************************
*
* clock skew is 7.67n and 7.95n for rising and falling transitions.
vphi1 179 0 pulse 0 5 7.2n 1n 1n 500n 1000n
* clock skew 9.21n and 8.74n for rising and falling transitions.
vphi1_bar 613 0 pulse 5 0 8.25n 1n 1n 500n 1000n
Vdd 101010 0 dc 5
.ic v(4785)=5 v(4784)=5 v(5002)=0 v(5006)=0 v(4514)=0 v(4757)=0
.ic v(4500)=5 v(4501)=5 v(4488)=0 v(4717)=0 v(4299)=0 v(4300)=0
.ic v(4296)=5 v(4295)=5 v(4193)=0 v(4198)=0 v(3896)=0 v(4145)=0
.ic v(3878)=5 v(3879)=5 v(3939)=0 v(4100)=0 v(3701)=5 v(3698)=0
.ic v(3697)=0 v(3653)=5 v(3573)=5 v(3464)=5 v(3465)=0 v(3097)=5
.ic v(3094)=0 v(3093)=0 v(3052)=5 v(2979)=5 v(2869)=0 v(2870)=5
.ic v(2503)=0 v(2504)=0 v(2500)=5 v(2499)=5 v(2399)=0 v(2403)=0
.ic v(2286)=0 v(2359)=0 v(2132)=5 v(2131)=5 v(2117)=0 v(2307)=0
.ic v(2065)=5 v(1805)=0 v(1518)=5 v(1501)=0 v(1502)=0 v(1556)=5
.ic v(1722)=5 v(1512)=0 v(1752)=0 v(1513)=5 v(1499)=0
c2424 2424 0 0.1ff
c2980 2980 0 0.1ff
c2978 2978 0 0.1ff
```

c3574 3574 0 0.1ff
c4791 4791 0 0.1ff
c34696 3896 0 52ff
c34727 4500 0 55ff
c34743 4488 0 243ff
c34867 4717 0 81ff
c35101 2399 0 243ff
c35106 2403 0 81ff
c35192 4514 0 52ff
c35276 2500 0 55ff
c35379 2065 0 53ff
c35471 3052 0 243ff
c35482 2979 0 81ff
c35597 2503 0 53ff
c35694 3094 0 55ff
c36293 3653 0 243ff
c36311 3573 0 81ff
c36400 3097 0 53ff
c36543 3698 0 55ff
c36754 4193 0 243ff
c36761 4198 0 81ff
c36808 1501 0 55ff
c36810 1513 0 86ff
c36818 1556 0 243ff
c36887 1722 0 736ff
c36916 3701 0 53ff
c36959 4296 0 55ff
c37137 2132 0 55ff
c37178 2117 0 243ff
c37332 2307 0 81ff
c37427 1518 0 52ff
c37437 5002 0 243ff
c37456 5006 0 81ff
c37641 4299 0 53ff
c37816 2286 0 52ff
c38248 2869 0 52ff
c38505 3464 0 52ff
c38579 3878 0 55ff
c38598 3939 0 243ff
c38686 4100 0 81ff
c1499 1499 0 0.1ff
c1502 1502 0 0.1ff
c1512 1512 0 0.1ff
c1752 1752 0 0.1ff
c1805 1805 0 0.1ff
c2131 2131 0 0.1ff
c2359 2359 0 0.1ff
c2499 2499 0 0.1ff
c2504 2504 0 0.1ff

```
c2870 2870 0 0.1ff
c3093 3093 0 0.1ff
c3465 3465 0 0.1ff
c3697 3697 0 0.1ff
c3879 3879 0 0.1ff
c4145 4145 0 0.1ff
c4295 4295 0 0.1ff
c4300 4300 0 0.1ff
c4501 4501 0 0.1ff
c4757 4757 0 0.1ff
c4784 4784 0 0.1ff
c4785 4785 0 0.1ff
*
m12130 4785 614 4784 101010 pmos w = 7u l = 3u
m12578 4784 182 4785 0 nmos w = 4u l = 3u
m12599 4784 5002 0 0 nmos w = 12u l = 3u
m12343 5002 557 5006 101010 pmos w = 4u l = 3u
v557 557 0 dc 0
m12381 5006 4514 4789 0 nmos w = 4u l = 3u
m12382 5006 4791 4789 101010 pmos w = 4u l = 3u
* inverter made from 4514 to 4791
mzz 4791 4514 0 0 nmos w = 4u l = 4u
mzz1 101010 4514 4791 101010 pmos w = 4u l = 4u
.ic v(4791)=5
*
m12133 101010 4790 4789 101010 pmos w = 8u l = 3u
v4790 4790 0 dc 0
.ic v(4789)=5
c4789 4789 0 0.1ff
m12043 4757 4513 4514 101010 pmos w = 16u l = 3u
m12044 101010 4500 4757 101010 pmos w = 16u l = 3u
m11361 4501 182 4500 0 nmos w = 4u l = 3u
m12033 4501 614 4500 101010 pmos w = 7u l = 3u
m11402 4501 4488 0 0 nmos w = 12u l = 3u
m11802 4717 557 4488 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
m11756 4512 4299 4717 0 nmos w = 4u l = 3u
m11757 4512 4513 4717 101010 pmos w = 4u l = 3u
* inverter made from 4299 to 4513
mzzz 4513 4299 0 0 nmos w = 4u l = 4u
mzzz1 101010 4299 4513 101010 pmos w = 4u l = 4u
.ic v(4513)=5
c4513 4513 0 0.1ff
*
m12026 4512 4487 101010 101010 pmos w = 8u l = 3u
v4487 4487 0 dc 0
.ic v(4512)=5
c4512 4512 0 0.1ff
m10704 4300 4220 4299 101010 pmos w = 16u l = 3u
```

```
m10705 101010 4296 4300 101010 pmos w = 16u l = 3u
m10939 4295 182 4296 0 nmos w = 4u l = 3u
v182 182 0 dc 5
m10700 4296 614 4295 101010 pmos w = 7u l = 3u
v614 614 0 dc 0
m11313 4295 4193 0 0 nmos w = 12u l = 3u
m10464 4193 557 4198 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
m10490 4198 3896 4219 0 nmos w = 4u l = 3u
m10491 4198 4220 4219 101010 pmos w = 4u l = 3u
m10703 101010 4197 4219 101010 pmos w = 8u l = 3u
v4197 4197 0 dc 0
.ic v(4219)=5
c4219 4219 0 0.1ff
* inverter made from 3896 to 4220
mzzzz 4220 3896 0 0 nmos w = 4u l = 4u
mzzzz1 101010 3896 4220 101010 pmos w = 4u l = 4u
.ic v(4220)=5
c4220 4220 0 0.1ff
*
m10331 4145 3895 3896 101010 pmos w = 16u l = 3u
v3895 3895 0 dc 0
m10332 101010 3878 4145 101010 pmos w = 16u l = 3u
m9718 3879 182 3878 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m10321 3879 614 3878 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m9783 3879 3939 0 0 nmos w = 12u l = 3u
m10108 4100 557 3939 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
m10536 4100 3895 3893 0 nmos w = 4u l = 3u
* v3895 3895 0 dc 5
m10535 4100 3701 3893 101010 pmos w = 4u l = 3u
m10586 101010 1010878 3893 101010 pmos w = 30u l = 3u
c38697 3893 0 157ff
v1010878 1010878 0 dc 0
.ic v(3893)=5
m9698 0 3698 3701 0 nmos w = 4u l = 3u
m9686 3697 182 3698 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m9049 3698 614 3697 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m9253 3697 3653 101010 101010 pmos w = 30u l = 3u
m9270 3653 557 3573 101010 pmos w = 4u l = 3u
m8665 3610 3574 3573 0 nmos w = 4u l = 3u
m8667 3610 3464 3573 101010 pmos w = 4u l = 3u
* inverter made from 3464 to 3574
mzzzzz 3574 3464 0 0 nmos w = 4u l = 4u
mzzzzz1 101010 3464 3574 101010 pmos w = 4u l = 4u
```

```
.ic v(3574)=0
*
* m9667 0 3698 3610 0 nmos w = 12 l = 3
* v3698 3698 0 dc 5
m9667 0 13698 3610 0 nmos w = 12u l = 3u
v13698 13698 0 dc 5
c36290 3610 0 157ff
.ic v(3610)=0
m8286 3464 3465 0 0 nmos w = 4u l = 3u
m8482 101010 3097 3465 101010 pmos w = 8u l = 3u
m7709 0 3094 3097 0 nmos w = 4u l = 3u
m7719 3093 182 3094 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m7427 3094 614 3093 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m7690 3093 3052 101010 101010 pmos w = 30u l = 3u
m7675 3052 557 2979 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
m7028 2979 2869 2978 0 nmos w = 4u l = 3u
m7029 2979 2980 2978 101010 pmos w = 4u l = 3u
m7706 2978 3009 0 0 nmos w = 4u l = 3u
v3009 3009 0 dc 5
* inverter made from 2869 to 2980
mxzz 2980 2869 0 0 nmos w = 4u l = 4u
mxzz1 101010 2869 2980 101010 pmos w = 4u l = 4u
.ic v(2980)=5
*
m6890 2917 2870 2869 101010 pmos w = 16u l = 3u
m6891 101010 2709 2917 101010 pmos w = 16u l = 3u
v2709 2709 0 dc 0
.ic v(2917)=5
c2917 2917 0 0.1ff
m6682 0 2503 2870 0 nmos w = 4u l = 3u
m5856 2504 112424 2503 101010 pmos w = 16u l = 3u
v112424 112424 0 dc 0
m5857 101010 2500 2504 101010 pmos w = 16u l = 3u
m6109 2499 182 2500 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m5853 2500 614 2499 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m6126 2499 2399 0 0 nmos w = 12u l = 3u
m5618 2399 557 2403 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
m5644 2403 2286 2423 0 nmos w = 4u l = 3u
m5645 2403 2424 2423 101010 pmos w = 4u l = 3u
m5855 101010 2402 2423 101010 pmos w = 8u l = 3u
* inverter made from 2286 to 2424
mxxzz 2424 2286 0 0 nmos w = 4u l = 4u
mxxzz1 101010 2286 2424 101010 pmos w = 4u l = 4u
```

```
.ic v(2424)=5
*
v2402 2402 0 dc 0
.ic v(2423)=5
c2423 2423 0 0.1ff
m5492 2359 2287 2286 101010 pmos w = 16u l = 3u
v2287 2287 0 dc 0
m5493 101010 2132 2359 101010 pmos w = 16u l = 3u
m5255 2131 182 2132 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m5482 2131 614 2132 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m4929 2131 2117 0 0 nmos w = 12u l = 3u
m5267 2307 557 2117 101010 pmos w = 4u l = 3u
* v557 557 0 dc 0
* m5694 2307 2287 2116 0 nmos w = 4 l = 3
m5693 2307 2065 2116 101010 pmos w = 4u l = 3u
m5738 101010 12132 2116 101010 pmos w = 30u l = 3u
.ic v(2116)=5
c37392 2116 0 157ff
v12132 12132 0 dc 0
m4513 2065 1805 0 0 nmos w = 4u l = 3u
m4060 101010 1518 1805 101010 pmos w = 8u l = 3u
m3326 0 1501 1518 0 nmos w = 4u l = 3u
m3309 1502 182 1501 0 nmos w = 4u l = 3u
* v182 182 0 dc 5
m3943 1502 614 1501 101010 pmos w = 7u l = 3u
* v614 614 0 dc 0
m3961 1502 1556 101010 101010 pmos w = 30u l = 3u
m3718 1722 559 1556 101010 pmos w = 4u l = 3u
v559 559 0 dc 0
m4000 0 1512 1722 0 nmos w = 40u l = 3u
m3947 1512 348 1752 101010 pmos w = 10u l = 3u
v348 348 0 dc 0
m3740 101010 1513 1752 101010 pmos w = 20u l = 3u
m3336 0 1499 1513 0 nmos w = 12u l = 3u
m3308 1500 179 1499 0 nmos w = 4u l = 3u
m3942 1500 613 1499 101010 pmos w = 7u l = 3u
m3960 1500 1557 101010 101010 pmos w = 30u l = 3u
.ic v(1500)=5
c1500 1500 0 0.1ff
v1557 1557 0 dc 0
.end
```

[3] SPICE input data for the critical path of the NuBus Interface Controller in the SPUR circuit

```
* SPICE input data for the critical path of the NuBus Interface
* Controller in the SPUR circuit
* (φ1(R)–φ2(F) segment)
.tran 0.5n 10n
.print tran v(265)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vphi1 331 0   pulse 0 5 0 0.01n 0.01n 25.0n 100n
vphi1_1 290 0 pulse 5 0 0 0.01n 0.01n 25.0n 100n
v268 268 0 dc 5
v246 246 0 dc 0
v309 309 0 dc 0
vdd 100 0 dc 5
.ic v(265)=0 v(223)=0 v(120)=5 v(143)=0
.ic v(202)=5 v(201)=0 v(245)=5 v(331)=0
c807 265 0 45ff
c848 223 0 46ff
c951 120 0 158ff
c929 143 0 186ff
c869 202 0 183ff
c870 201 0 355ff
c827 245 0 86ff
c741 331 0 267ff
m625 265 268 223 0 nmos w = 3.2u l = 1.6u
m594 265 246 223 100 pmos w = 3.2u l = 1.6u
```

```
m572 100 120 223 100 pmos w = 6.4u l = 1.6u
m325 120 143 0 0 nmos w = 3.2u l = 1.6u
m44 143 0 100 100 pmos w = 3.2u l = 3.2u
m318 143 202 0 0 nmos w = 3.2u l = 1.6u
m548 202 201 0 0 nmos w = 8.8u l = 1.6u
m597 100 245 201 100 pmos w = 13.6u l = 1.6u
m719 309 331 245 0 nmos w = 3.2u l = 1.6u
m679 309 290 245 100 pmos w = 3.2u l = 1.6u
.end
```

[4] SPICE input data for the critical path of the On-Chip Instruction Cache Controller in the SPUR circuit

[4.1] $\phi 4(R)$–$\phi 1(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ4(R)–φ1(R) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
vdd 2 0 dc 5
c1437 192 0 97ff
c1439 190 0 803ff
c1489 101 0 585ff
c1540 85 0 60ff
c1541 87 0 128ff
c1543 84 0 41ff
c1475 154 0 80ff
m248 84 0 2 2 pmos w = 3.2u l = 3.2u
m268 0 85 84 0 nmos w = 3.2u l = 1.6u
m251 0 87 85 0 nmos w = 3.2u l = 1.6u
m288 2 101 87 2 pmos w = 13.6u l = 1.6u
m541 101 192 0 0 nmos w = 8u l = 1.6u
m534 192 194 154 2 pmos w = 3.2u l = 1.6u
m528 192 190 154 0 nmos w = 3.2u l = 1.6u
```

```
m392 154 0 2 2 pmos w = 3.2u l = 3.2u
.print tran v(84)
.ic v(84)=0 v(85)=5 v(87)=0 v(101)=5 v(192)=0 v(190)=0 v(154)=5
vphi4   190 0 pulse  0 5 0 1n 1n 14n 80n
vphi4_b 194 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.2] $\phi 4(R)$–$\phi 1(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ4(R)–φ1(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1217 393 0 97ff
c1438 191 0 96ff
c1439 190 0 803ff
c1493 100 0 587ff
c1537 90 0 65ff
c1539 89 0 127ff
c1545 82 0 50ff
c1476 153 0 75ff
m1178 393 129 82 2 pmos w = 3.2u l = 1.6u
v129 129 0 dc 0
m1181 393 131 82 0 nmos w = 3.2u l = 1.6u
v131 131 0 dc 5
m246 82 0 2 2 pmos w = 3.2u l = 3.2u
m272 0 90 82 0 nmos w = 3.2u l = 1.6u
m255 0 89 90 0 nmos w = 3.2u l = 1.6u
m284 2 100 89 2 pmos w = 13.6u l = 1.6u
m540 0 191 100 0 nmos w = 8u l = 1.6u
m533 191 194 153 2 pmos w = 3.2u l = 1.6u
m527 191 190 153 0 nmos w = 3.2u l = 1.6u
m391 153 0 2 2 pmos w = 3.2u l = 3.2u
.print tran v(393)
```

```
.ic v(393)=0 v(82)=0 v(90)=5 v(89)=0 v(100)=5 v(191)=0 v(153)=5 v(190)=0
vphi4   190 0 pulse  0 5 0 1n 1n 14n 80n
vphi4_b 194 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.3] $\phi 3(R)$–$\phi 4(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ3(R)–φ4(R) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1352 80 0 572ff
c1361 262 0 100ff
c1362 261 0 720ff
c1549 59 0 135ff
c1577 35 0 92ff
c1604 8 0 84ff
c1434 195 0 93ff
m6 8 0 2 2 pmos w = 3.2u l = 3.2u
m175 8 35 0 0 nmos w = 3.2u l = 1.6u
m163 0 59 35 0 nmos w = 3.2u l = 1.6u
m244 2 80 59 2 pmos w = 13.6u l = 1.6u
m855 0 262 80 0 nmos w = 8u l = 1.6u
m827 262 261 195 0 nmos w = 3.2u l = 1.6u
m842 262 270 195 2 pmos w = 3.2u l = 1.6u
m543 195 0 2 2 pmos w = 3.2u l = 2.4u
.print tran v(8)
.ic v(8)=0 v(35)=5 v(59)=0 v(80)=5 v(262)=0 v(195)=5 v(261)=0
vphi3   261 0 pulse  0 5 0 1n 1n 14n 80n
vphi3_b 270 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.4] $\phi 3(R)$–$\phi 4(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ3(R)–φ4(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1207 403 0 113ff
c1351 71 0 786ff
c1360 263 0 99ff
c1362 261 0 720ff
c1555 70 0 107ff
c1564 47 0 109ff
c1599 12 0 112ff
c1603 9 0 79ff
c1433 196 0 93ff
m1205 9 190 403 0 nmos w = 6.4u l = 1.6u
v190 190 0 dc 5
m153 0 12 9 0 nmos w = 3.2u l = 1.6u
m10 12 0 2 2 pmos w = 3.2u l = 3.2u
m95 12 47 0 0 nmos w = 3.2u l = 1.6u
m217 47 70 0 0 nmos w = 8.8u l = 1.6u
m228 2 71 70 2 pmos w = 13.6u l = 1.6u
m856 71 263 0 0 nmos w = 8u l = 1.6u
m828 263 261 196 0 nmos w = 3.2u l = 1.6u
m843 263 270 196 2 pmos w = 3.2u l = 1.6u
m544 196 0 2 2 pmos w = 3.2u l = 2.4u
.print tran v(403)
```

```
.ic v(403)=5 v(9)=5 v(12)=0 v(47)=5 v(70)=0 v(71)=5
.ic v(263)=0 v(196)=5 v(261)=0
vphi3   261 0 pulse  0 5 0 1n 1n 14n 80n
vphi3_b 270 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.5] $\phi 3(R)$–$\phi 2(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ3(R)–φ2(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1351 71 0 786ff
c1360 263 0 99ff
c1511 111 0 12ff
c1521 109 0 116ff
c1362 261 0 720ff
c1433 196 0 93ff
m308 109 112 111 2 pmos w = 11.2u l = 1.6u
v112 112 0 dc 0
m307 111 71 2 2 pmos w = 11.2u l = 1.6u
m856 71 263 0 0 nmos w = 8u l = 1.6u
m828 263 261 196 0 nmos w = 3.2u l = 1.6u
m843 263 270 196 2 pmos w = 3.2u l = 1.6u
m544 196 0 2 2 pmos w = 3.2u l = 2.4u
.print tran v(109)
.ic v(109)=0 v(111)=0 v(71)=5 v(263)=0 v(196)=5 v(261)=0
vphi3   261 0 pulse  0 5 0 1n 1n 14n 80n
vphi3_b 270 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.6] $\phi 2(R)$–$\phi 4(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ4(R) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1250 66 0 835ff
c1252 123 0 892ff
c1257 357 0 100ff
c1569 43 0 157ff
c1585 27 0 87ff
c1606 6 0 84ff
c1298 319 0 59ff
m4 6 0 2 2 pmos w = 3.2u l = 3.2u
m168 0 27 6 0 nmos w = 3.2u l = 1.6u
m109 0 43 27 0 nmos w = 3.2u l = 1.6u
m212 2 66 43 2 pmos w = 13.6u l = 1.6u
m1121 0 357 66 0 nmos w = 8u l = 1.6u
m1102 357 123 319 0 nmos w = 3.2u l = 1.6u
m1113 357 112 319 2 pmos w = 3.2u l = 1.6u
m1028 319 0 2 2 pmos w = 3.2u l = 3.2u
.print tran v(6)
.ic v(6)=0 v(27)=5 v(43)=0 v(66)=5 v(357)=0 v(319)=5 v(123)=0
vphi2   123 0 pulse  0 5 0 1n 1n 14n 80n
vphi2_b 112 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.7] $\phi 2(R)$–$\phi 4(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ4(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1303 317 0 97ff
c1343 278 0 94ff
c1334 286 0 91ff
c1335 287 0 136ff
c1307 116 0 527ff
c1518 117 0 116ff
c1249 69 0 773ff
c1256 358 0 100ff
c1252 123 0 892ff
c1297 320 0 68ff
c1509 118 0 18ff
m1013 317 190 278 0 nmos w = 3.2u l = 1.6u
m1019 317 194 278 2 pmos w = 3.2u l = 1.6u
v190 190 0 dc 5
v194 194 0 dc 0
m864 278 0 2 2 pmos w = 3.2u l = 3.2u
m944 278 286 0 0 nmos w = 3.2u l = 1.6u
m876 0 287 286 0 nmos w = 3.2u l = 1.6u
m983 2 116 287 2 pmos w = 13.6u l = 1.6u
m328 0 117 116 0 nmos w = 5.6u l = 1.6u
m315 2 119 118 2 pmos w = 16.8u l = 1.6u
```

```
v119 119 0 dc 0
m314 118 69 117 2 pmos w = 16.8u l = 1.6u
m1122 69 358 0 0 nmos w = 8u l = 1.6u
m1103 358 123 320 0 nmos w = 3.2u l = 1.6u
m1114 358 112 320 2 pmos w = 3.2u l = 1.6u
m1029 320 0 2 2 pmos w = 3.2u l = 3.2u
.print tran v(317)
.ic v(317)=0 v(278)=0 v(286)=5 v(287)=0 v(116)=5 v(118)=5
.ic v(117)=0 v(69)=5 v(358)=0 v(320)=5 v(123)=0
vphi2   123 0 pulse  0 5 0 1n 1n 14n 80n
vphi2_b 112 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 15n
.end
```

[4.8] $\phi 2(R)$–$\phi 3(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ3(R) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
vdd 2 0 dc 5
c1363 73 0 979ff
c1374 255 0 136ff
c1406 223 0 86ff
c1428 201 0 100ff
c1519 114 0 101ff
c1520 113 0 116ff
c1510 115 0 12ff
m549 201 0 2 2 pmos w = 3.2u l = 2.4u
m760 0 223 201 0 nmos w = 3.2u l = 1.6u
m740 0 255 223 0 nmos w = 3.2u l = 1.6u
m836 2 73 255 2 pmos w = 13.6u l = 1.6u
m325 0 113 73 0 nmos w = 3.2u l = 1.6u
m312 113 112 115 2 pmos w = 11.2u l = 1.6u
m311 115 114 2 2 pmos w = 11.2u l = 1.6u
v114 114 0 dc 0
.print tran v(201)
.ic v(201)=0 v(223)=5 v(255)=0 v(73)=5 v(113)=0 v(115)=5 v(114)=0
vphi2_b 112 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.9] $\phi 2(R)$–$\phi 3(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ3(F) segment)
*
**************************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**************************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**************************************************************************
*
vdd 2 0 dc 5
c1252 123 0 892ff
c1358 265 0 100ff
c1363 73 0 979ff
c1374 255 0 136ff
c1377 252 0 164ff
c1425 203 0 98ff
c1431 198 0 123ff
c1520 113 0 116ff
c1507 124 0 2ff
m830 265 261 198 0 nmos w = 3.2u l = 1.6u
v261 261 0 dc 5
m845 265 270 198 2 pmos w = 3.2u l = 1.6u
v270 270 0 dc 0
m546 198 0 2 2 pmos w = 3.2u l = 2.4u
m697 0 203 198 0 nmos w = 3.2u l = 1.6u
m695 203 252 0 0 nmos w = 3.2u l = 1.6u
m835 252 255 2 2 pmos w = 14.4u l = 1.6u
m820 255 73 0 0 nmos w = 8.8u l = 1.6u
m310 2 113 73 2 pmos w = 11.2u l = 1.6u
m327 113 123 124 0 nmos w = 3.2u l = 1.6u
m326 124 114 0 0 nmos w = 3.2u l = 1.6u
```

```
v114 114 0 dc 5
.print tran v(265)
.ic v(265)=0 v(198)=0 v(203)=5 v(252)=0 v(255)=5 v(73)=0
.ic v(113)=5 v(124)=0v(123)=0
vphi2   123 0 pulse  0 5 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.10] $\phi 2(R)$–$\phi 1(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ1(R) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1239 379 0 96ff
c1240 377 0 109ff
c1363 73 0 979ff
c1520 113 0 116ff
c1519 114 0 101ff
c1510 115 0 12ff
m1159 379 377 0 0 nmos w = 4u l = 1.6u
m1146 2 110 378 2 pmos w = 16.8u l = 1.6u
v110 110 0 dc 0
m1145 378 73 377 2 pmos w = 16.8u l = 1.6u
m325 0 113 73 0 nmos w = 3.2u l = 1.6u
m312 113 112 115 2 pmos w = 11.2u l = 1.6u
m311 115 114 2 2 pmos w = 11.2u l = 1.6u
v114 114 0 dc 0
*
.print tran v(379)
.ic v(379)=5 v(378)=5 v(377)=0 v(73)=5 v(113)=0 v(115)=5 v(114)=0
vphi2_b 112 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.11] $\phi 2(R)$–$\phi 1(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ2(R)–φ1(F) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
vdd 2 0 dc 5
c1217 393 0 97ff
c1250 66 0 835ff
c1252 123 0 892ff
c1257 357 0 100ff
c1524 99 0 120ff
c1538 88 0 65ff
c1545 82 0 50ff
c1298 319 0 59ff
m1178 393 129 82 2 pmos w = 3.2u l = 1.6u
v129 129 0 dc 0
m1181 393 131 82 0 nmos w = 3.2u l = 1.6u
v131 131 0 dc 5
m246 82 0 2 2 pmos w = 3.2u l = 3.2u
m270 82 88 0 0 nmos w = 3.2u l = 1.6u
m269 0 99 88 0 nmos w = 3.2u l = 1.6u
m304 2 66 99 2 pmos w = 13.6u l = 1.6u
m1121 0 357 66 0 nmos w = 8u l = 1.6u
m1102 357 123 319 0 nmos w = 3.2u l = 1.6u
m1113 357 112 319 2 pmos w = 3.2u l = 1.6u
m1028 319 0 2 2 pmos w = 3.2u l = 3.2u
.print tran v(393)
```

```
.ic v(393)=0 v(82)=0 v(88)=5 v(99)=0 v(66)=5 v(357)=0 v(319)=5 v(123)=0
vphi2   123 0 pulse  0 5 0 1n 1n 14n 80n
vphi2_b 112 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.12] $\phi 1(R)$–$\phi 4(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ4(R) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
vdd 2 0 dc 5
c1370 64 0 730ff
c1489 101 0 585ff
c1490 143 0 116ff
c1586 26 0 91ff
c1595 16 0 178ff
c1606 6 0 84ff
c1488 144 0 12ff
m4 6 0 2 2 pmos w = 3.2u l = 3.2u
m167 6 26 0 0 nmos w = 3.2u l = 1.6u
m70 0 16 26 0 nmos w = 3.2u l = 1.6u
m204 2 64 16 2 pmos w = 13.6u l = 1.6u
m372 0 143 64 0 nmos w = 3.2u l = 1.6u
m370 143 129 144 2 pmos w = 11.2u l = 1.6u
m369 144 101 2 2 pmos w = 11.2u l = 1.6u
v101 101 0 dc 0
.print tran v(6)
.ic v(6)=0 v(26)=5 v(16)=0 v(64)=5 v(143)=0
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.13] $\phi 1(R)$–$\phi 4(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ4(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1436 193 0 96ff
c1474 155 0 85ff
c1471 158 0 75ff
c1445 183 0 178ff
c1370 64 0 730ff
c1490 143 0 116ff
c1488 144 0 12ff
c1489 101 0 585ff
m529 193 190 155 0 nmos w = 3.2u l = 1.6u
v190 190 0 dc 5
m535 193 194 155 2 pmos w = 3.2u l = 1.6u
v194 194 0 dc 0
m393 155 0 2 2 pmos w = 3.2u l = 3.2u
m451 155 158 0 0 nmos w = 3.2u l = 1.6u
m443 0 183 158 0 nmos w = 3.2u l = 1.6u
m531 2 64 183 2 pmos w = 13.6u l = 1.6u
m372 0 143 64 0 nmos w = 3.2u l = 1.6u
m370 143 129 144 2 pmos w = 11.2u l = 1.6u
m369 144 101 2 2 pmos w = 11.2u l = 1.6u
v101 101 0 dc 0
.print tran v(193)
```

```
.ic v(193)=0 v(155)=0 v(158)=5 v(183)=0 v(64)=5 v(143)=0 v(144)=5
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.14] $\phi 1(R)$–$\phi 3(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ3(R) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1428 201 0 100ff
c1396 233 0 91ff
c1423 207 0 244ff
c1370 64 0 730ff
c1490 143 0 116ff
c1488 144 0 12ff
c1489 101 0 585ff
m549 201 0 2 2 pmos w = 3.2u l = 2.4u
m769 201 233 0 0 nmos w = 3.2u l = 1.6u
m663 0 207 233 0 nmos w = 3.2u l = 1.6u
m790 2 64 207 2 pmos w = 13.6u l = 1.6u
m372 0 143 64 0 nmos w = 3.2u l = 1.6u
m370 143 129 144 2 pmos w = 11.2u l = 1.6u
m369 144 101 2 2 pmos w = 11.2u l = 1.6u
v101 101 0 dc 0
.print tran v(201)
.ic v(201)=0 v(233)=5 v(207)=0 v(64)=5 v(143)=0 v(144)=5
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.15] $\phi 1(R)$–$\phi 3(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ3(F) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
vdd 2 0 dc 5
c1358 265 0 100ff
c1370 64 0 730ff
c1415 213 0 93ff
c1423 207 0 244ff
c1431 198 0 123ff
c1489 101 0 585ff
c1490 143 0 116ff
c1488 144 0 12ff
m830 265 261 198 0 nmos w = 3.2u l = 1.6u
v261 261 0 dc 5
m845 265 270 198 2 pmos w = 3.2u l = 1.6u
v270 270 0 dc 0
m546 198 0 2 2 pmos w = 3.2u l = 2.4u
m716 0 213 198 0 nmos w = 3.2u l = 1.6u
m570 0 207 213 0 nmos w = 3.2u l = 1.6u
m790 2 64 207 2 pmos w = 13.6u l = 1.6u
m372 0 143 64 0 nmos w = 3.2u l = 1.6u
m370 143 129 144 2 pmos w = 11.2u l = 1.6u
m369 144 101 2 2 pmos w = 11.2u l = 1.6u
v101 101 0 dc 0
.print tran v(265)
```

```
.ic v(265)=0 v(198)=0 v(213)=5 v(207)=0 v(64)=5 v(143)=0 v(144)=5
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.16] $\phi 1(R)$–$\phi 2(R)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ2(R) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1258 75 0 744ff
c1276 337 0 153ff
c1288 329 0 67ff
c1296 321 0 50ff
c1480 150 0 111ff
c1481 149 0 116ff
c1479 151 0 12ff
m1030 321 0 2 2 pmos w = 3.2u l = 3.2u
m1058 0 329 321 0 nmos w = 3.2u l = 1.6u
m1057 0 337 329 0 nmos w = 3.2u l = 1.6u
m1115 2 75 337 2 pmos w = 13.6u l = 1.6u
m388 0 149 75 0 nmos w = 3.2u l = 1.6u
m386 149 129 151 2 pmos w = 11.2u l = 1.6u
m385 151 150 2 2 pmos w = 11.2u l = 1.6u
v150 150 0 dc 0
.print tran v(321)
.ic v(321)=0 v(329)=5 v(337)=0 v(75)=5 v(149)=0 v(151)=5
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[4.17] $\phi 1(R)$–$\phi 2(F)$ segment (SPUR On-Chip Instruction Cache Controller)

```
* SPICE input data for the critical path of the On-Chip Instruction
* Cache Controller in the SPUR circuit
* (φ1(R)–φ2(F) segment)
*
*********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
*********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
*********************************************************************
*
vdd 2 0 dc 5
c1253 361 0 114ff
c1264 65 0 729ff
c1278 345 0 107ff
c1282 335 0 58ff
c1293 324 0 63ff
c1294 323 0 51ff
c1485 103 0 584ff
c1486 146 0 116ff
c1484 147 0 12ff
m1106 361 123 323 0 nmos w = 6.4u l = 1.6u
v123 123 0 dc 5
m1047 0 324 323 0 nmos w = 3.2u l = 1.6u
m1033 324 0 2 2 pmos w = 3.2u l = 3.2u
m1044 324 335 0 0 nmos w = 3.2u l = 1.6u
m1074 335 345 0 0 nmos w = 8.8u l = 1.6u
m1085 2 65 345 2 pmos w = 13.6u l = 1.6u
m380 0 146 65 0 nmos w = 3.2u l = 1.6u
m378 146 129 147 2 pmos w = 11.2u l = 1.6u
m377 147 103 2 2 pmos w = 11.2u l = 1.6u
v103 103 0 dc 0
.print tran v(361)
```

```
.ic v(361)=5 v(323)=5 v(324)=0 v(335)=5
.ic v(345)=0 v(65)=5  v(146)=0 v(147)=5
vphi1_b 129 0 pulse  5 0 0 1n 1n 14n 80n
.tran 0.1n 10n
.end
```

[5] SPICE input data for the critical path of the SPUR ALU

[5.1] $\phi 3(R)$–$\phi 3(F)$ segment (SPUR ALU)

```
* SPICE input data for the critical path of the SPUR ALU
* (φ3(R)–φ3(F) segment)
*
**********************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**********************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**********************************************************************
*
.tran 0.1n 20n
.print tran v(662)
.opt nopage nomod
vphi3 176 0 pulse 0 5 0 1n 1n 100n 200n
vdd 5 0 dc 5
.ic v(662)=0 v(664)=0 v(660)=5 v(632)=0 v(631)=5 v(423)=0
.ic v(422)=5 v(214)=0 v(213)=5 v(217)=5 v(216)=0 v(940)=5
.ic v(860)=0 v(906)=5 v(867)=0 v(874)=5 v(875)=5 v(876)=5
.ic v(176)=0
c5892 662 0 225ff
c5890 664 0 15ff
c5896 660 0 197ff
c5712 632 0 306ff
c5917 631 0 260ff
c5927 423 0 306ff
c6126 422 0 260ff
c6136 214 0 306ff
c6335 213 0 260ff
```

```
c6338 217 0 3ff
c6340 216 0 55ff
c5606 940 0 63ff
c5617 860 0 148ff
c5643 906 0 63ff
c5644 867 0 108ff
c5676 876 0 3ff
c5677 875 0 3ff
c5680 874 0 80ff
c5709 176 0 7502ff
m881 662 8 664 5 pmos w = 13.6u l = 1.6u
v8 8 0 dc 0
m880 664 660 5 5 pmos w = 13.6u l = 1.6u
m1073 660 632 807 0 nmos w = 6.4u l = 1.6u
c5741 807 0 3ff
.ic v(807)=5
m1072 807 806 805 0 nmos w = 6.4u l = 1.6u
c5742 805 0 3ff
v806 806 0 dc 5
.ic v(805)=5
m1071 805 176 0 0 nmos w = 6.4u l = 1.6u
m838 632 631 5 5 pmos w = 16u l = 1.6u
m841 631 423 635 0 nmos w = 6.4u l = 1.6u
c5920 635 0 3ff
.ic v(635)=5
m840 635 634 633 0 nmos w = 6.4u l = 1.6u
c5921 633 0 3ff
v634 634 0 dc 5
.ic v(633)=5
m839 633 176 0 0 nmos w = 6.4u l = 1.6u
m556 423 422 5 5 pmos w = 16u l = 1.6u
m559 422 214 426 0 nmos w = 6.4u l = 1.6u
c6129 426 0 3ff
.ic v(426)=5
m558 426 425 424 0 nmos w = 6.4u l = 1.6u
v425  425 0 dc 5
c6130 424 0 3ff
.ic v(424)=5
m557 424 176 0 0 nmos w = 6.4u l = 1.6u
m274 214 213 5 5 pmos w = 16u l = 1.6u
m277 213 180 217 0 nmos w = 6.4u l = 1.6u
v180 180 0 dc 5
m276 217 216 215 0 nmos w = 6.4u l = 1.6u
c6339 215 0 3ff
.ic v(215)=5
m275 215 176 0 0 nmos w = 6.4u l = 1.6u
m1325 216 940 5 5 pmos w = 8u l = 1.6u
m1319 941 860 940 0 nmos w = 6.4u l = 1.6u
c5605 941 0 3ff
```

```
.ic v(941)=5
m1320 943 942 941 0 nmos w = 6.4u l = 1.6u
v942 942 0 dc 5
c5604 943 0 3ff
.ic v(943)=5
m1321 0 176 943 0 nmos w = 6.4u l = 1.6u
m1269 860 906 5 5 pmos w = 8u l = 1.6u
m1263 907 867 906 0 nmos w = 6.4u l = 1.6u
c5642 907 0 3ff
.ic v(907)=5
m1264 908 893 907 0 nmos w = 6.4u l = 1.6u
c5641 908 0 3ff
v893 893 0 dc 5
.ic v(908)=5
m1265 0 176 908 0 nmos w = 6.4u l = 1.6u
m1210 867 874 5 5 pmos w = 8u l = 1.6u
m1214 875 3 874 0 nmos w = 6.4u l = 1.6u
v3 3 0 dc 5
m1215 876 30 875 0 nmos w = 6.4u l = 1.6u
v30 30 0 dc 5
m1216 0 176 876 0 nmos w = 6.4u l = 1.6u
.end
```

[5.2] $\phi 2(R)$–$\phi 3(R)$ segment (SPUR ALU)

```
* SPICE input data for the critical path of the SPUR ALU
* (φ2(R)–φ3(R) segment)
*
**************************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**************************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**************************************************************************
*
.tran 0.05n 15n
.print tran v(1451) v(178)
vphi2 1934 0 pulse 0 5 0 1n 1n 100n 200n
vphi2_bar 1937 0 pulse 5 0 0 1n 1n 100n 200n
vdd 5 0 dc 5
.ic v(178)=0  v(849)=5  v(848)=0  v(864)=5  v(3)=0 v(1459)=5
.ic v(1462)=5 v(1456)=0 v(1451)=5 v(1448)=5 v(1936)=0
.ic v(1935)=5 v(1934)=0
*
c6376 178  0 85ff
c5697 849  0 85ff
c5698 848  0 93ff
c5684 864  0 85ff
c6543 3    0 326ff
c5093 1459 0 144ff
c5085 1462 0 3ff
c5092 1456 0 181ff
c5096 1451 0 145ff
c5098 1448 0 181ff
c4611 1936 0 196ff
c4613 1935 0 100ff
c4427 1934 0 965ff
*
```

```
m1140 178 849 5 5 pmos w = 7.2u l = 1.6u
m1135 849 848 0 0 nmos w = 3.2u l = 1.6u
m1188 848 864 5 5 pmos w = 7.2u l = 1.6u
m1183 864 3 0 0 nmos w = 3.2u l = 1.6u
m2321 3 1459 5 5 pmos w = 18.4u l = 1.6u
m2327 1459 1454 1462 0 nmos w = 6.4u l = 1.6u
v1454 1454 0 dc 5
m2326 1462 1456 0 0 nmos w = 6.4u l = 1.6u
m2312 5 1451 1456 5 pmos w = 8u l = 1.6u
m2310 1451 1450 1448 5 pmos w = 3.2u l = 1.6u
v1450 1450 0 dc 0
m3395 0 1936 1448 0 nmos w = 8u l = 1.6u
m3398 5 1935 1936 5 pmos w = 9.6u l = 1.6u
m3391 1935 1934 1933 0 nmos w = 3.2u l = 1.6u
m3396 1935 1937 1933 5 pmos w = 8u l = 1.6u
c4612 1933 0 73ff
.ic v(1933)=0
m4032 1933 2128 0 0 nmos w = 3.2u l = 1.6u
v2128 2128 0 dc 5
.end
```

[5.3] $\phi 2(R)$–$\phi 4(R)$ segment (SPUR ALU)

```
* SPICE input data for the critical path of the SPUR ALU
* (φ2(R)–φ4(R) segment)
*
**************************************************************************
*           MOS1 parameters for SPUR
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto= 0.75 kp=39.5u gamma=0.40 lambda=0.025
+  phi=0.771 ld=0.2u
.model pmos pmos vto=-0.75 kp=15.0u gamma=0.50 lambda=0.045
+  phi=0.735 ld=0.05u
**************************************************************************
*           MOS2 parameters for SPUR.
*  .model nmos nmos level=2 vto= 0.75 kp=76.0u gamma=.40 lambda=.025
*  + tox=25n nsub=4e16 tpg=+1 xj=.25u ld=.20u uexp=.16 vmax=5.5e4
*  + js=1000u cgso=220p cgdo=220p cj=230u cjsw=260p cgbo=400p
** phi F = -0.3855862 -> phi=0.771
*  .model pmos pmos level=2 vto=-0.75 kp=27.0u gamma=.50 lambda=.045
*  + tox=25n nsub=2.0e16 tpg=-1 xj=.20u ld=.05u uexp=.15 vmax=9.0e4
*  + js=1000u cgso=220p cgdo=220p cj=670u cjsw=215p cgbo=400p
** phi F = 0.3675644 -> phi=0.735
**************************************************************************
*
.tran 0.05n 15n
.print tran v(92)
.opt nopage nomod limpts=800
vphi2 1934 0 pulse 0 5 0 1n 1n 100n 200n
vphi2_bar 1937 0 pulse 5 0 0 1n 1n 100n 200n
vdd 5 0 dc 5
.ic v(92)=5   v(93)=0   v(1523)=5 v(1526)=5 v(1520)=0
.ic v(1515)=5 v(1513)=5 v(1961)=0 v(1960)=5 v(1934)=0
c6458 92 0 112ff
c6459 93 0 327ff
c5033 1523 0 144ff
c5025 1526 0 3ff
c5032 1520 0 181ff
c5036 1515 0 145ff
c5038 1513 0 181ff
c4587 1961 0 196ff
c4589 1960 0 100ff
c4427 1934 0 965ff
m113 0 93 92 0 nmos w = 3.2u l = 1.6u
m2453 93 1523 5 5 pmos w = 18.4u l = 1.6u
m2459 1523 1518 1526 0 nmos w = 6.4u l = 1.6u
v1518 1518 0 dc 5
m2458 1526 1520 0 0 nmos w = 6.4u l = 1.6u
```

```
m2444 5 1515 1520 5 pmos w = 8u l = 1.6u
m2442 1515 1450 1513 5 pmos w = 3.2u l = 1.6u
v1450 1450 0 dc 0
m3475 0 1961 1513 0 nmos w = 8u l = 1.6u
m3478 5 1960 1961 5 pmos w = 9.6u l = 1.6u
m3471 1960 1934 1959 0 nmos w = 3.2u l = 1.6u
m3476 1960 1937 1959 5 pmos w = 8u l = 1.6u
c4588 1959 0 73ff
.ic v(1959)=0
m4064 1959 2144 0 0 nmos w = 3.2u l = 1.6u
v2144 2144 0 dc 5
.end
```

APPENDIX 4

EXAMPLE RUNS

This appendix contains the netlist of a simple example, 1 bit of the SOAR ALU, and the output from the *E-TV* program. Copies of all software, and the examples described in this dissertation, are available from the following address:

The Software Office

Industrial Liaison Program

Department of Electrical Engineering and Computer Sciences,

University of California at Berkeley

Berkeley, CA 94720

[1] E-TV input data for the SOAR 1bit circuit

```
*** 1 bit of soar alu
* voltage step
.step 0.1
* analysis request card
.phase high PHI3
*************** MOS1 parameters for soar ******************************
* Obtained from MOS2. kp was adjusted to match dc characteristic.
.model nmos nmos vto=0.93 kp=21.8e-6 gamma=0.834 phi=0.695 lambda=0.025 ld=0.24
.model pmos pmos vto=-0.844 kp=4.45e-6 gamma=0.723 phi=0.514 lambda=0.0527 ld=0.51
*
vDD 1 0 dc 5
vsum 33 0 dc 5
vand 32 0 dc 5
vxor 34 0 dc 5
vor 31 0 dc 5
*
* clock signal
data clock PHI3 PHI3 15
.model PHI3 pulse 0 5 30e-9 32e-9 80e-9 82e-9 100e-9
*
* MOS transistors. The default unit of W and L is um.
m190 136 135 1 1 pmos w = 7 l = 3 Flow 1
m191 1 138 137 1 pmos w = 7 l = 3 Flow 1
m193 1 141 140 1 pmos w = 7 l = 3 Flow 1
m194 1 143 142 1 pmos w = 7 l = 3 Flow 1
m196 136 144 145 0 nmos w = 4 l = 3 Flow 136
m197 0 135 136 0 nmos w = 7 l = 3 Flow 0
m198 140 31 138 0 nmos w = 4 l = 3 Flow 140
m199 142 32 138 0 nmos w = 4 l = 3 Flow 142
m201 0 141 140 0 nmos w = 7 l = 3 Flow 0
m202 0 143 142 0 nmos w = 7 l = 3 Flow 0
m204 0 138 137 0 nmos w = 4 l = 3 Flow 0
m205 145 33 138 0 nmos w = 4 l = 3 Flow 145
m206 135 34 138 0 nmos w = 4 l = 3 Flow 135
m208 150 149 135 0 nmos w = 4 l = 3 Flow 150
m209 152 151 150 0 nmos w = 4 l = 3 Flow 152
m210 154 153 141 0 nmos w = 4 l = 3 Flow 154
m211 155 42 138 0 nmos w = 4 l = 3 Flow 155
m215 160 153 143 0 nmos w = 4 l = 3 Flow 160
m216 161 153 135 0 nmos w = 4 l = 3 Flow 161
m220 135 108 145 0 nmos w = 4 l = 3 Flow 135
m223 165 155 141 0 nmos w = 4 l = 3 Flow 165
m224 166 155 160 0 nmos w = 4 l = 3 Flow 166
m225 167 155 161 0 nmos w = 4 l = 3 Flow 167
m231 0 15 154 0 nmos w = 4 l = 3 Flow 0
m232 0 15 165 0 nmos w = 4 l = 3 Flow 0
m233 0 15 166 0 nmos w = 4 l = 3 Flow 0
m234 0 15 167 0 nmos w = 4 l = 3 Flow 0
m235 0 15 152 0 nmos w = 4 l = 3 Flow 0
```

```
m284 138 55 189 0 nmos w = 9 l = 3 Flow 189
m309 138 15 1 1 pmos w = 7 l = 3 Flow 1
m312 1 15 141 1 pmos w = 7 l = 3 Flow 1
m313 143 15 1 1 pmos w = 7 l = 3 Flow 1
m315 135 15 1 1 pmos w = 7 l = 3 Flow 1
* Capacitance. The default unit is fF.
c1792 135 0 124
c150 150 0 1
c152 152 0 1
c161 161 0 1
c 167 167 0 1
c1803 138 0 139
c137 137 0 1
c190 136 0 59
c145 145 0 1
c1804 143 0 67
c1807 142 0 56
c160 160 0 1
c166 166 0 1
c1797 141 0 80
c154 154 0 1
c165 165 0 1
c140 140 0 1
.end
```

[2] Output of the *E-TV* program for the SOAR 1bit circuit

```
E-TV :
-------------------------------------------------------------------------
|            [ CLOCK  SKEW  VALUES ]
|            ==========================
|
|    The clock signal is the one used to synchronize the
|    orderly and controlled flow of information in the system.
|    Primary clocks 'phi' and 'phi-bar' are defined to have
|    the same rising and falling slopes. They are defined to
|    cross each other at the inverting voltage of inverter (Vinv)
|    which can be set by the user.
|    'phi(-bar)' nodes are compared to primary clock 'phi(-bar)'.
|    Rising (falling) skew is the time difference of rising
|    (falling) waveforms of the clock node and primary signal.
|          node_name(rising_skew/falling_skew)
-------------------------------------------------------------------------


[1] clock 'PHI3' :
      rises at 31.00ns, falls at 81.00ns with a period of 100.00ns

   ['PHI3' clock nodes]
     (1) 15 (0.00ns/0.00ns) : primary
   ['PHI3-bar' clock nodes] : None

No logic-control signals are given in the input file.


-------------------------------------------------------------------------
|            [ TIMING  CONSTRAINT  CHECK ]
|            ================================
|
|                  SYNCHRONOUS SYSTEMS
|    Logic paths from input nodes/clocked storage elements to
|    output nodes/clocked storage elements, that are not inter-
|    posed by other clocked storage elements, are verified using
|    actual arriving logic-control/clock signals.
|    Paths ending at clocked storage elements are reported in the
|    order of the worst-case evaluation-time margin,
|    while paths ending at output nodes (blocked nodes) are
|    reported in order of the worst-case delay.
-------------------------------------------------------------------------


            <<  PHI3 section  >>


( Timing Constraint Check against Clock Signals )
```

[Segment 1] PHI3(R)->PHI3(F) : Clock Separation OK.

Primary clock information :
PHI3 rises at 31.00ns and PHI3 falls at 81.00ns. Sep.=50.00ns

(1) Node 138 falls at 44.94ns (Delay=13.94ns) while a limiting clock node 15 (PHI3) falls at 81.00ns (F Skew=0.00ns)
Separation margin : 36.06ns

Node 138 falls at 44.94ns thru (m205)
Node 145 falls at 35.11ns thru (m220)
Node 135 falls at 35.03ns thru (m216)
Node 161 falls at 30.90ns thru (m225)
Node 167 falls at 30.78ns thru (m234)
Node 15 (clock PHI3) rises at 31.00ns

(2) Node 135 falls at 35.03ns (Delay=4.03ns) while a limiting clock node 15 (PHI3) falls at 81.00ns (F Skew=0.00ns)
Separation margin : 45.97ns

Node 135 falls at 35.03ns thru (m216)
Node 161 falls at 30.90ns thru (m225)
Node 167 falls at 30.78ns thru (m234)
Node 15 (clock PHI3) rises at 31.00ns

(3) Node 143 falls at 33.26ns (Delay=2.26ns) while a limiting clock node 15 (PHI3) falls at 81.00ns (F Skew=0.00ns)
Separation margin : 47.74ns

Node 143 falls at 33.26ns thru (m215)
Node 160 falls at 30.90ns thru (m224)
Node 166 falls at 30.78ns thru (m233)
Node 15 (clock PHI3) rises at 31.00ns

(4) Node 141 falls at 32.80ns (Delay=1.80ns) while a limiting clock node 15 (PHI3) falls at 81.00ns (F Skew=0.00ns)
Separation margin : 48.20ns

Node 141 falls at 32.80ns thru (m223)
Node 165 falls at 30.78ns thru (m232)
Node 15 (clock PHI3) rises at 31.00ns

[Segment 2] PHI3(R)->'BLOCKED nodes' :

Primary clock information : PHI3 rises at 31.00ns

[Rising transitions]

(1) Node 137 (Delay = 10.60ns)

Node 137 rises at 41.60ns thru (m191)
Node 138 falls at 44.94ns thru (m205)
Node 145 falls at 35.11ns thru (m220)

```
    Node 135 falls at 35.03ns thru (m216)
    Node 161 falls at 30.90ns thru (m225)
    Node 167 falls at 30.78ns thru (m234)
    Node 15 (clock PHI3) rises at 31.00ns

 [ Falling transitions ]

   (1) Node 137 (Delay = -46.17ns)

    Node 137 falls at -15.17ns thru (m204)
    Node 138 precharges at -10.26ns thru (m206)
    Node 135 precharges at -16.56ns thru (m315)
    Node 15 (clock PHI3) falls at 81.00ns

( Check for pre-(dis)charging delay ) : All OK or NONE
-------------------------------------

( Cycles in circuit ) : None
---------------------

-----------------------------------------------------------------------
|          ANALYSIS PARAMETERS and STATISTICS
|          ==================================
-----------------------------------------------------------------------

 [1] Global voltage step : 0.1V
 [2] .gcapdef : 0.1fF
 [3] MOS model : small-signal companion
 [4] Inverting voltage : 2.5V
 [5] Max depth of transistor in chain for delay evaluation : 8
 [6] Fixed nodes were propagated
 [7] Total mosfet number  : 33
 [8] Total net number  : 33
 [9] Number of stages for delay evaluation:
      Total:37  Rising Transition=14  Falling Transition=23
[10] Number of nodes for delay evaluation:
      Total:17  Rising Transition=9  Falling Transition=17
[11] Peak memory used  : 149 kbyte
[12] CPU time usage
    Readin and setup data struct. : 0.12sec
    Initialization            : 0.04sec
    Finding data in/out nodes    : 0.12sec
    Finding control/clock nodes  : 0sec
    Topological ordering all nodes: 0.02sec
    Waveform generation for control/clock nodes  : 0.01sec
    Propagation of set-to-high(low) nodes       : 0sec
    Pruning uninteresting nodes  : 0sec
    Finding pre(dis)charged nodes : 0.01sec
    Finding latches           : 0.01sec
    Timing verification between pairs of latches : 2.55sec
    Design reference check       : 0sec
    TOTAL                     : 2.89sec
```