National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

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

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

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

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

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

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

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

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Rapid Current Analysis for CMOS Digital Circuits

Na-Han CHAN

McGill University, Montreal

July 1994

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering.

ISBN 0-315-99960-8

Abstract

A versatile and efficient computer-aided analysis tool, CUREST, has been developed for the analysis of supply currents in CMOS digital circuits. It is based on Nabavi-Lishi's semi-analytical model for computing the current and delay in a CMOS logic gate which, when compared to HSPICE running the level-3 MOSFET model, is more than three orders of magnitude faster, and accurate to within 10%. CUREST is built on top of the timing analyser TAMIA and, in particular, uses its circuit parser and its data structure to store the circuit topology and primary input pattern.

Extensive tests on benchmark circuits containing up to 555 gates, which were analysed with CUREST using thousands of primary input patterns, demonstrate that the current analysis time is in the range of 1ms per gate per input pattern, using a SUN4/490 workstation with 32 Mb of main memory, running the SUN OS 4.103 operating system. The peak value of the total supply current, the current rise-time, and the time at which the peak occurs are usually computed to within 10% of HSPICE. However, appreciable errors often occur in the average current. This is because at the moment we do not have a good model for dealing with incomplete transitions associated with glitches in a CMOS gate.

Résumé

CUREST est un outil versatile et efficace d'analyse assistée par ordinateur qui a été conçu pour l'analyse des courants d'alimentation dans les circuits digitaux CMOS. L'outil est basé sur les modèles semi-analytiques de Nabavi-Lishi pour le calcul des délais et courants des portes logiques CMOS. En comparaison à HSPICE (au *3ième* niveau de complexité des modèles MOSFET), ces nouveaux modèles permettent des calculs de l'ordre de milliers de fois plus rapides que HSPICE. De plus, les calculs sont précis à 10% de la valeur exacte. CUREST a été implanté à partir du système d'analyse temporelle TAMIA. En particulier CUREST se sert du programme d'analyse lexicographique et des structures de données de TAMIA pour la mise en mémoire des circuits et de leur topologie ainsi que les signaux d'entrée.

Plusieurs tests, ayant des milliers de signaux d'entrée chacun, ont été effectués sur des circuits étalons comportant jusqu'à 555 portes logiques. Les résultats indiquent une performance moyenne d'une milliseconde multiplier par le nombre de portes logiques et par le nombre de signaux d'entrée. Cette formule s'applique à un système d'exploitation SUN OS 4.103 en opération sur un SUN4/490 possédant 32 MegaOctets. Les résultats pour le courant maximal, la durée de transit du courant, et le temps lorsque le courant atteint son maximum sont tous à l'intérieur de 10% des valeurs obtenues par HSPICE. Cependant le calcul du courant moyen est souvent inexact. Ceci est dû à une carence de nos modèles pour l'analyse des transitions incomplètes causées par les soubresauts des portes logiques CMOS.

Acknowledgements

I would like to thank my thesis supervisor, Prof. Nicholas Rumin, for his support and guidance throughout my Master's research studies. His encouragement and patience were greatly appreciated. In addition, I am thankful to A. Nabavi-Lishi, S. Gaiotti, Prof. M. Dagenais and Prof. J. Rajski for their valuable help in my research work.

I would also like to take this opportunity to acknowledge the help of the staff of the Microelectronic and Computer System(MACS) laboratory. In particular, thanks to Jacek Slaboszewicz, and Christine Marquis for all their help. I have also enjoyed many fruitful discussions with my fellow graduate students in the MACS laboratory and I cherish the friendships that have been built between us.

This work was supported by research grants from the Natural Sciences and Engineering Research Council(NSERC) of Canada as well as by FCAR Scholarship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As VLSI circuit densities continue to increase, accurate information on the current distribution in the chip power and ground buses is needed by the designer. The reasons include the need to minimize the sizes of the power rails without, at the same time, producing current densities high enough to cause premature failure due to electromigration, or performance degradation due to excessive voltage drops. The current information is also needed to determine the power requirements of the chip. In this chapter, a number of existing current and power estimation techniques are reviewed and, in Section 1.3 our own approach is introduced.

The existing current and power estimation techniques can be grouped into two major categories: pattern-dependent and pattern-independent. The pattern-dependent method evaluates the current waveform of a circuit according to one particular primary input pattern. On the other hand, the pattern-independent approach produces the average current waveform corresponding to a large number of input patterns. In order to tackle the excessive heat dissipation, electromigration, and excessive voltage drop problems, it is the true pattern-dependent current waveforms that are required to determine the worst case conditions. However, these are impractical for circuits with more than approximately 20 inputs. Hence, pattern-independent methods, which

are more efficient, are often used. In the following section, the pattern-independent methods are discussed which is followed by the pattern-dependent methods. Finally, the objective and outline of this dissertation are presented.

1.1 Pattern-independent current estimation techniques

In general, pattern-independent methods adopt a statistical approach to estimate the current and power dissipation in a circuit. This method does not provide the information on the instantaneous current waveform associated with a particular input pattern, but it is fast in computing the average current and power over a large number of input test patterns.

Najm *et al.* computed the expected current waveform of a circuit by carrying out an event-driven probabilistic simulation of the circuit [1][2][3]. First, they replaced logic values and transitions of primary inputs of a circuit by signal probabilities [4] and transition probabilities respectively. Then, they derived the corresponding probability waveforms at internal circuit nodes and propagated these waveforms towards the primary outputs. The expected current pulses of each individual gates are summed to create the expected current waveform of the circuit. In particular, the expected current waveform at each subcircuit is approximated by a right-angled triangular pulse that starts with a peak of $E[i(t)]$ at time t and decays linearly to zero at time $(t+\tau)$. The expected current value $E[i(t)]$ is computed by assuming the transistor behaves as a resistor and operates in its saturation region, and τ is evaluated as a function of $E[i(t)]$. In [5], the authors extended the probabilistic simulation approach to include the computation of the variance waveform of the circuit which is useful in obtaining a better estimation of the median time-to-failure(MTF) of a circuit.

The current model used in the above approach ignores the short circuit current

and assumes the input transitions are step functions. In addition, the authors in [5] assume that there is no signal correlation between internal circuit nodes which results in overestimating the total current drawn by the circuit. Although Kriplani *et al.* [6] develop an algorithm that resolves the signal correlation problem as mention above, their approach complicates the algorithm and hence, slows down the analysis.

In [7], Kriplani *et al* proposed a method to predict an upper bound for the current waveform that a circuit can draw. For each of the excitations, low, high, high→low, and low→high, the authors stored a list of intervals during which a node might carry that excitation. The internal node transition information is derived by merging these intervals. It is assumed that there are only two possible current waveforms associated with every gate; one due to low→high transitions and the other due to high→low transitions. The maximum of the two current waveforms at every time point is considered to be the worst case supply current. A right-angled triangular current pulse is used to approximate the current waveform of a switching gate, where the peak of the current is user specified, and the duration is computed by charge conservation. Once all the gates' currents have been computed, the total current waveform is determined by adding together the individual gate contributions.

In this approach, the authors assumed in each combinational block the inputs switch simultaneously. Moreover, they assumed the delay of each gate is fixed and specified ahead of time.

Burch *et al.* [8] proposed to combine the simulation-based approaches with the probabilistic approaches to estimate the power of a circuit. The basic idea of their method is to apply randomly generated input vectors to the circuit and monitor, with a simulator, the resulting power value. Since this approach uses a finite number of patterns to estimate the average power, which really depends on the complete set of possible input patterns, this method belongs to the general class of so-called *Monte Carlo* methods. The total power is computed as the sum of the average power dissipated

at every internal node in the circuit. The average power dissipated at node i during the time interval T is evaluated as:

$$\frac{1}{2}V_{dd}^2 C_i \frac{n_i(T)}{T}$$

where $n_i(T)$ is the number of transitions at node i and C_i is the total capacitance at i. To make sure that the measured power is typical, the circuit has to be in its steady state. This is achieved by simulating the circuit for a period of time, which is called the *setup time*, before evaluating the power. A number of randomly generated input patterns are applied to the circuit until the power value obtained reaches a desired accuracy which is specified by the user. The authors are able to achieve excellent speed performance with this method; however, the computational cost increases as higher accuracy is desired. There is a specific problem in handling sequential circuits, where it is not clear how to obtain the *setup time*.

The CAD tool PowerPlay [9] developed by Krodel is designed to compute the instantaneous power waveform of a VLSI circuit design based on standard cells. First, analog simulation is employed to accurately compute the energy and the power waveform associated with each cell-type by taking into account the input transition and loading capacitances. Then, the power waveform is derived by approximating the original analog power waveform with a rectangle, where the area equals the corresponding energy and the height meets the peak power value. These power waveforms are stored in a data-base which is used together with the timing and transition information provided by a logic simulator to obtain the instantaneous power waveform of a circuit. The speed-up of PowerPlay is more than four orders of magnitude faster than SPICE and good accuracy is claimed for average power computation.

In [10], Dresig *et al* developed a method to compute the average dynamic power of a CMOS circuit based on the following equation derived in [11].

$$P_D = \frac{1}{2}V_{DD}^2 \cdot \sum_i f_i C_i$$

where f_i is the frequency of signal transitions at the output of gate i, and C_i is its load capacitance. It is assumed that the wiring capacitance is negligible, and hence C_i is the sum of the output capacitance C_o of the driving gate and the input capacitances C_{in_i} of the driven gates. The authors also assumed that the average output capacitance and input capacitance of all gates are constants. Therefore,

$$C_i = C_{out} + fanout_i \cdot C_{in}$$

where $fanout_i$ is the gate fanout and C_{out}, C_{in} are constants determined from technology parameters.

To relate f_i and the switching frequency, f, of the circuit, the authors defined a_i as follow:

$$a_i = \frac{f_i}{2f}$$

where a_i can be interpreted as the mean switching probability at node i during a clock cycle. As a result, the total power can be expressed as:

$$P_D = V_{DD}^2 \cdot f \cdot C_{in} \sum_i a_i (\frac{C_{out}}{C_{in}} + fanout_i) \tag{1.1}$$

The switching activity a_i is derived as the total number of switching A_i at node i divided by the number of cycles n, where A_i and n can be obtained through gate level simulations. Since the fanout of each internal node is known, the total average power is readily computed by Equation(1.1). The limitation of this approach is that short circuit current is not taken into account and heuristic is needed to tackle the signal correlation problem.

In [12], Vanoostende *et al* proposed to compute the maximal currents and maximal current derivatives of CMOS circuits using *activity waveforms*. The *activity waveform* consists of a set of parameter pairs describing the periods of time that a node can switch. To capture the switching activities associated with a node, the authors assumed that only one input of a gate switches at a certain time. Then, the input patterns

that correspond to the fastest or slowest switching output is determined. Using the derived input patterns, the gate is simulated using the traditional circuit simulation method. The peak current observed during the simulation is used as the amplitude of the maximal current waveform of the gate. A trapezoid is used to approximate the maximal current waveform with a duration determined by the output activity intervals which are also computed from the simulation. The maximal current derivative waveform is approximated using a square shape, with a shorter duration than for the maximal current waveform. The total current waveform is obtained by adding the contributions of all logic gates in the circuit.

Landman *et al* proposed a power estimation technique for high level system architectures in [13]. The power consumption is determined by the system input statistics, where the input signal patterns are described by using three statistical parameters: mean, variance and correlation coefficient. From the input statistics , the statistics of its outputs are computed. By propagating statistics in this fashion, the statistical parameters for each internal bus are derived. Once the internal bus statistics are known, the bit transition probabilities(i.e. gate input transition probabilities) can be derived based on a stochastic model developed by the authors. Finally, the energy required to drive data with the given statistics onto a bus is computed from these bit probabilities. At present, the proposed method is limited to compute power for datapath components.

1.2 Pattern-dependent current estimation techniques

In general, this approach involves using event driven simulation. The results of the timing analysis of the circuit are combined with the supply current waveform of each CMOS gate to obtain the total supply current. In principle, SPICE[14] is the most accurate approach. However, in order to find the worst case conditions, we have to

examine all possible combinations of the input signals. This is extremely expensive if it takes a considerable amount of CPU time, like SPICE, to compute each current waveform. Numerous methods have been developed to trade accuracy for speed. These are summarized in the following paragraphs.

In [15], the power dissipation of a single logic gate is computed by reducing the CMOS gate to an inverter circuit containing resistances and capacitances. The non-switching transistors are replaced with resistors and capacitors whose values are determined by a set of parameters. These parameters are obtained by circuit simulations or by direct measurements. For more than one input switching, the switching transistors are combined and the equivalent input start time, and transition time, and transistor size are computed. The resulting nonlinear equations are solved by a method analogous to that used in SPICE but optimized for the inverter. The speed of this approach is claimed to be two orders of magnitude faster than conventional circuit simulators with an average error of 10% per logic cell.

In [16] and [17] methods are developed to find the worst case voltage drop in the power bus network. The program Hercules designed by Tyagi [16] used a table-look-up method to estimate the current. The authors used SPICE simulation to find the current information of a logic gate based on its input transition time, β ratio, and its loading capacitances. These factors are later combined into one variable called *rise-time-ratio* which is used to index the look-up tables. To compute the current drawn by a circuit, the author decomposed it into *stages* which consist of a chain of transistors leading from a strong voltage source to an output node or a gate. The current drawn from a *stage* is computed using the tables and the worst case current drawn from a *stage* is obtained by assuming all transistors are fully on except the trigger transistor.

In [17], Stark and Horowitz proposed using the simulation tool Ariel to find the worst case voltage drop and current density in the power bus network of CMOS circuits. First, a resistor network is extracted from the circuit description using a Magic-

based[17] resistance extractor. Then, a transistor level simulator RSIM[17] is used to generate node transition information. This procedure identifies the transistors that inject charge to the supply network and computes the inject current as

$$\frac{C_{load} \cdot V_{DD}}{T_{in}}$$

where T_{in} is the node transition time and C_{load} is the total capacitance at that node. From the analysis of RSIM, the current distribution of the resistive network is obtained. Finally a linear-tree-based algorithm is used to solve the branch currents and node voltages in the resistive network. In this approach, the short-circuit current is ignored which accounts for as high as 20% of the total supply current [18].

Similar to Hercules, SPIDER [19] is a CAD tool used to adjust the line width of a power bus. The transient current waveform at each identified node in the power bus is obtained through analysis of the individual subcircuits where SPICE is used to determine the subcircuit current waveforms. As a result, this process becomes very expensive as the number of inputs increase. Besides, it is hard to find the timing to cascade subcircuit outputs as inputs to its loading subcircuits.

Chowdhury and Barkatullah [20] used Shockley's transistor models to estimate the maximum current in CMOS logic circuits. For a general CMOS logic gate, the authors reduced it to an equivalent inverter depending on the sizes of the transistors in the pull-up and pull-down transistor sub-groups. Then, the current drawn by the inverter is obtained by solving the Kirchoff's current equations at the output node using the Forward-Euler method. The linear portion of the output transition is taken as input(s) to the loading gate(s) and the current associated with the loading gate is obtained similarly as the driving gate.

The above approach is implemented in the procedure CEST [20] to compute the current in a CMOS circuit in response to a particular input pattern. First, the logic gates in a given CMOS circuit are divided into stages. Then, stage numbers are assigned such that no gate in stage i is driven by a gate whose stage number is greater than or

equal to i. This ensure that the inputs of a gate are known as functions of time before the gate is processed. The order of processing gates within the same stage does not matter. Therefore, the current and output voltage of each gate in the CMOS circuit can be computed in the order of its stage number, starting with the lowest stage number. The total current drawn from the circuit is obtained by adding the individual current associated with each gate which are switching.

To find the input pattern corresponding to the maximum current drawn in a CMOS circuit, the authors use a branch-and-bound algorithm. In order to reduce the computational cost of this algorithm, a heuristic method is developed. However, the solution obtained from the heuristic is only a local optimal. Besides, the above current model tends to increase delay in series networks and tends to decrease delay in a parallel networks because the authors assumed that all inputs of a gate are identical in arrival time and rise time. Finally, the computational cost of this method is high, because numerical integration is used to obtain the maximum current.

In [21], [22], [23], [24] and [25], methods are proposed to compute the current waveform of a circuit. In SIMCURRENT [21], a database of analog current waveforms is used to estimate the final current waveform of a circuit. This database keep records of both inverting and non-inverting current waveforms of a reference gate. Furthermore, simulated current waveforms of the reference gate with zero load up to maximum load is also kept in the database. The current waveform of a gate is determined from the database according to the total loading capacitance and the *switching capacitance* of the gate, where the *switching capacitance* C_{sw} of the processed gate is defined as the mean of the current consumption of the rising and falling edges of the output. The speed of this method is claimed to be three orders of the magnitude faster than SPICE. Its database is huge.

CURRANT [22] is a switch-level simulator that can generate current waveforms for VLSI circuits. The authors use a simple RC model to compute the peak drain source

current of a transistor and model the decaying waveform as an exponential function.

$$I = I_o(e^{\frac{-t}{RC}})$$

where R and C are the resistance and capacitance of the transistor and I_o is its saturation current. The gate current is accounted for as the summation of the drain source current that flows to and from the node that a gate is connected to. CURRANT comprises three major components which work together to generate drain source current waveforms for each transistor in a circuit. First, the authors use RTL[22], a switch-level simulator, to simulate digital MOS circuits and obtain the timing and logic information. Then, they use EXTRACT[22], an interface module, to interface the RTL simulator with the waveform generating module GENERATE[22] which produces the drain source current waveforms for each transistor in the circuit, based upon the information that is extracted from RTL. The accuracy of this method is claimed to be within 10% of SPICE for a single logic gate.

Deng et al. [23] used a symmetric triangular current pulse to approximate the current waveform of a logic gate. This triangular pulse is uniquely specified by a triplet($t_{peak}, I_{peak}, \Delta T$) where t_{peak} is the time that I_{peak} occurs, and ΔT is the time duration of the triangular pulse. ΔT is calculated from the precharacterized switch-level delay library based on the input rise/fall time and the RC delay parameter. t_{peak} is computed as:

$$t_{peak} = e.time + \frac{\Delta T}{2}$$

where $e.time$ is the start time of the input transition. The current peak I_{peak} is computed as:

$$I_{peak} = 2 \times \frac{charge}{\Delta T}$$

where *charge* is the product of output node voltage swing and load capacitances at the output node. When the output load is large, the triangular pulse becomes asymmetric and better estimation can be achieved by using an asymmetric triangular pulse

characterized by $(t_{peak}, I_{peak}, \Delta T_1, \Delta T_2)$ where ΔT_1 and ΔT_2 are respectively the rising and falling times of the pulse. The speed of this method is claimed to be three orders of magnitude faster than SPICE with 20% accuracy.

Wang *et al.* [24] adopted a more complex model to approximate the current waveform of a CMOS gate. They divided the gate current waveform into three regions and the characteristic in each region is approximated by an exponential function. These three exponential functions are specified by four parameters which are functions of load capacitances, aspect ratios of MOS transistors, and the slopes of input signals. The authors stored these parameters in a database from which the corresponding current waveform of a gate can be constructed. More accurate results are obtained with this method as compare to [23] at the expense of a more complex database.

Rouatbi *et al.* [25] combined the timing analysis results of a CMOS circuit with the estimation of supply current waveforms for each logic gate to obtain the total supply current waveform of the circuit. The supply current waveform of a gate is found as a combination of the capacitive and short-circuit current waveforms which are restricted to certain basic forms. The basic capacitive and short-circuit current waveforms of a particular CMOS gate are characterized by a set of parameters. To compute these parameters, the authors collapsed the logic gate, and derived an analytical model from the collapsed gate. Then, they used a symbolic software package(Maple [26]) to solve the involved model equation for the waveform parameters.

In this approach, the individual capacitance contribution from the switched nodes are assumed independent of their position in the topology. The authors claimed that they have achieved 3-4 orders of magnitude speed-up with respect to SPICE with maximum 10% deviation in the obtained current waveforms.

In [27], Benini *et al* estimated the power dissipation in CMOS circuit using an approach derived from the *current-limited switch-level timing simulation* technique presented in [28]. It is assumed that the current drawn by a CMOS gate can have two

possible values which are controlled by the gate-to-source voltage. Furthermore, the reactive effects are modeled as capacitors connected between circuit nodes and ground. Because the choice of the model, no integration is needed during the computation of the time domain responses. Modifications have been made to the approach described in [28] in order to allow the simulation of non-fully-complementary CMOS circuits with increased accuracy and robustness. The event-driven simulator built by the authors is capable of handling different types of CMOS digital circuits including pass-transistors and sequential logic with positive feedback loops. The authors claimed that the speed of their method is 2 orders of magnitude faster than SPICE with an error lower than 10%.

1.3 Motivation and overview of thesis

In [29], an inverter-base model was presented to compute the maximum current and delay of a CMOS gate without integration. The speed of this model is three orders of magnitude faster than SPICE, and it is more accurate than other models that have been reported. The model takes into account the sizes of the devices, output load, input transition time, input transition positions, and short-circuit current in computing the current waveforms. A relatively small database is needed. From the test results presented in [29], this model has an excellent performance in predicting the maximum current and its time of occurrence. It is also capable of producing an accurate current waveform of a gate. Therefore, its application to large VLSI circuits is promising.

In order to test the speed and accuracy of this model on large circuits with a large number of test vectors, an analysis program was needed. In this thesis, the approach presented in [29] is implemented in an event-driven simulation program CUREST which is based on the timing analysis tool TAMIA [31]. The information on the circuit topology as well as the input patterns are provided by TAMIA while the

current waveforms of the circuit are computed by CUREST.

The outline of this thesis is presented as follows. In Chapter 2, the current model proposed in [29] is reviewed, which is followed by the current estimation algorithm in Chapter 3. The accuracy and speed of CUREST are analysed in Chapter 4. Finally, conclusions are made and future work is discussed in Chapter 5.

Chapter 2

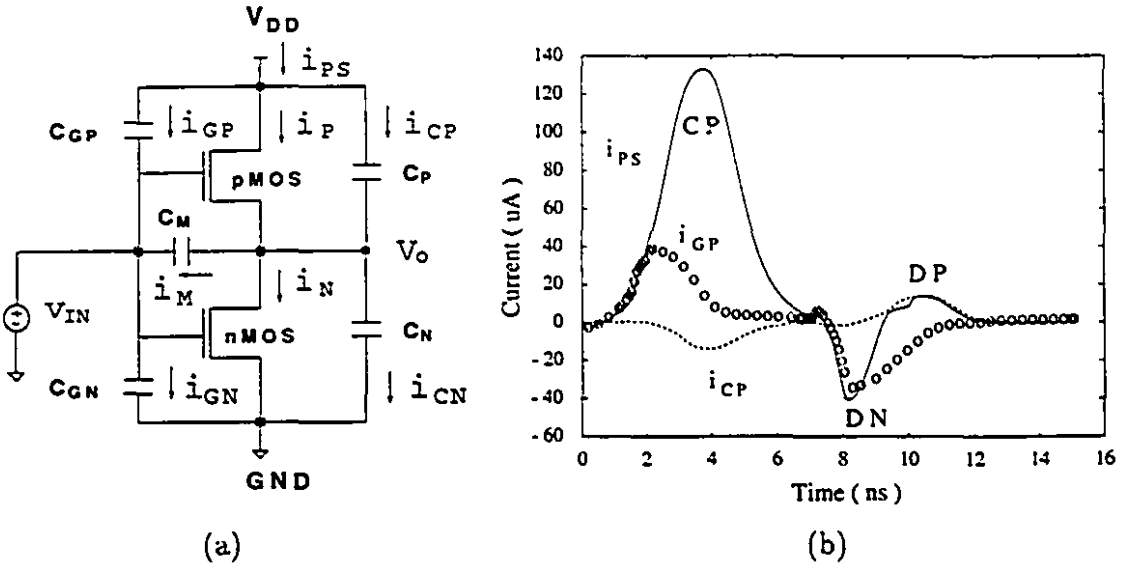
Delay and current models

In this chapter, we review the models that were developed by Nabavi-Lishi [29] to evaluate the supply current and the delay in a CMOS gate, and which we used in the design of CUREST. It is important to understand how these models work in order to appreciate how CUREST is constructed.

2.1 Inverter Model

A transistor-level model of an inverter is shown in Figure 2.1, where the parasitic capacitances are shown explicitly. $C_{GP}(C_{GN})$ represented the gate-to-source and the gate-to-bulk capacitances of the pMOS(nMOS) transistor while $C_P(C_N)$ represented the sum of drain-to-bulk capacitance of the pMOS(nMOS) transistor and the $C_{GP}(C_{GN})$ of the loading pMOS(nMOS) transistors. C_M includes the gate-to-drain capacitances of both the pMOS and the nMOS transistors. These parasitic capacitances are replaced by equivalent constant values which will be discussed in Section 2.4.

The supply current i_{PS} is computed from the three branch currents i_{GP}, i_{CP} and i_P. From HSPICE simulation and experiment, it is observed that there is only one current



(a) (b)

Figure 2.1: (a) Transistor-level model of a CMOS inverter. The effect of the loading gate(s) is included in C_N and C_P. (b) Charging and discharging currents of the inverter shown in (a) when driven by another inverter. i_{PS}(—), i_{GP}(o), and i_{CP}(- -) [29].

peak associated with a charging output node. The current model used to evaluate this peak is called the *CP model.* In order to compute the charging current of an inverter accurately, a *CW* model is developed. When the output is discharging, as illustrated in Figure 2.1(b), the supply current shows both a current maximum, DP, and a current minimum, DN. The models used to compute these two peaks are called the *DP model* and *DN model* respectively. As will be explained in section 2.5, the *DN* peak is rarely computed explicitly and hence, the *DN* model will not be presented. All these models are based on the assumption of a ramp voltage input.

2.2 Inverter with falling input transition: CP and CW Models

In this section, the models that are used to compute the maximum current and delay of a charging inverter i.e. an inverter with a falling input transition are presented. To compute the charging current either CP or CW model can be used. The CP model is presented first.

2.2.1 CP model

When the input falls from V_{DD} to $V_{DD} - \mid V_{tp} \mid$, where V_{tp} is the threshold voltage of the pMOS transistor, the pMOS transistor enters the saturation region. At this moment, the supply current i_{PS} is dominated by i_P which carries both the short-circuit and load currents. For a charging output, i_P is much greater than i_N and hence i_{CP} tracks i_P. Now i_P will reach its maximum before the pMOS transistor enters the triode region, but not later than when the input transition is complete. For this reason, and also because i_{GP} becomes zero when the input transition is completed, it follows that the maximum in i_{PS} must occur at:

$$t_m = min(t_{st}, T_i) \tag{2.1}$$

where t_{st} is the time when pMOS leaves the saturation region and T_i is the input transition time. In order to obtain an explicit expression for t_{st}, the output voltage $v_o(t_{st})$ at time t_{st} is required. To find $v_o(t_{st})$, a simple ramp approximation of the output voltage is used for which the transition time T_o is given by:

$$T_o = aT_i + b\frac{V_{DD}(C_N + C_P)}{I_P} \tag{2.2}$$

where I_P is the pMOS drain current corresponding to $v_{SG} = v_{SD} = V_{DD}$, and a, b are empirical parameters obtained by fitting the delay and maximum current of a min-sized symmetric inverter computed by the CP model to that obtained from HSPICE

simulation. The output voltage can be approximated in any manner of our choice as long as it produces an accurate value of the output voltage at the time t_{st}. A ramp is used because of its simplicity.

At time t_{st}, the source-to-drain voltage of the pMOS transistor is equal to V_{DSATP}, the pMOS transistor's drain saturation voltage. In order to obtain a closed-form solution for the time t_{st}, the long-channel approximation has to be used for V_{DSATP}. This would possibly lead to the resulting HSPICE MOS-3 model overestimating the drain current. However, it has been shown in [29] that, by setting the static feedback parameter ETA to zero, good agreement is obtained with respect to the HSPICE MOS-3 model for operation near pinch-off. Therefore, by combining Eq.(2.2) with the long-channel approximation for V_{DSATP}, t_{st} is given by:

$$t_{st} = \frac{T_i(1 + FB_p + \frac{V_{tp0}}{V_{DD}})}{1 + \frac{T_i}{T_o}(1 + FB_p)} \tag{2.3}$$

where FB_p is a SPICE technology constant for the pMOS transistor, and V_{tp0} represents its threshold voltage. Once t_{st} is known, t_m can be easily determined from Eq.(2.1).

Hence, $v_o(t_m)$ and $v_{in}(t_m)$ are obtained from:

$$v_o(t_m) = V_{DD} \cdot \frac{t_m}{T_o}$$

$$v_{in}(t_m) = V_{DD}(1 - \frac{t_m}{T_{IN}})$$

This permits the determination of $i_P(t_m)$ and $i_N(t_m)$. Moreover, the first derivative of the output voltage at t_m, which is also taken as the effective slope of the input to the loading gate, can be computed from:

$$\alpha_m \equiv \frac{dv_O}{dt}|_{t=t_m} = \frac{i_P(t_m) - i_N(t_m) - C_M\frac{V_{DD}}{T_i}}{C_N + C_P + C_M} \tag{2.4}$$

Now, the maximum supply current $i_{PS}(t_m)$ can be obtained as the sum of three branch currents i_P, i_{CP} and i_{GP}. Since $i_P(t_m)$ is already known, only two branch

currents have to be computed:

$$i_{CP}(t_m) = C_P \cdot \alpha_m$$

$$i_{GP}(t_m) = C_{GP} \cdot \frac{V_{DD}}{T_i}$$

To find the delay, which is defined as the difference between the time when the input and output voltages reach $V_{DD}/2$, the fact that $v_O(t_m)$ lies within the approximately linear region of the output voltage is utilized. This means that a straight line with slope α_m can be used to extrapolate from $v_O(t_m)$ to $v_o = V_{DD}/2$ and the delay can be computed as:

$$\tau = t_m + \frac{\frac{V_{DD}}{2} - v_O(t_m)}{\alpha_m} - \frac{T_i}{2}$$

In general, the accuracy of the models presented above depends on both the range of T_i and the β-ratio in the inverter. For a symmetric inverter($\beta_p = \beta_n$), with $a = 0.86$ and $b = 3.18$, $i_{PS}(t_m)$ is determined with a 10% error for a range of T_i from 1 to 100ns, for the 0.8,1.2 and 3.0 *microns* technologies. For asymmetric inverters, a look-up table or empirical equations are used to adjust the coefficients a and b to meet the requirements on the range of T_i and the β-ratio as illustrated in Table 2.1.

2.2.2 CW model

The CW model can be used to compute the charging current of an inverter. This is obtained in two steps. First the supply current is computed for the time interval $[0,t_m]$, where t_m is the time when the supply current is maximum, and then the remaining part of the supply current is evaluated for $t > t_m$.

For $t \in [0, t_m]$, the pMOS transistor operates mostly in the saturation region and its drain saturation current is relatively independent of the output voltage. Therefore, i_P can be computed without having to estimate accurately the output voltage. It follows

range of W_p/W_n	a	b
12.5–15.5	0.50	7.00
9.5–12.5	0.58	4.00
7.5–9.5	0.62	4.00
6.5–7.5	0.65	4.00
5.5–6.5	0.67	4.00
4.0–5.5	0.72	3.80
3.2–4.0	0.79	3.20
2.9–3.2	0.81	3.30
2.5–2.9	0.86	3.18
1.3–2.5	0.99	2.90
0.7–1.3	1.10	3.00
0.3–0.7	1.30	2.80

Table 2.1: Appropriate values of a and b obtained for a 1.2 micron technology [29].

that the supply current i_{PS} is obtained by evaluating i_P using the preceding method, and then by adding i_P to i_{GP} and i_{CP}, obtained as follows:

$$i_{GP} = C_{GP} \times \frac{V_{DD}}{T_i}$$

$$i_{CP} \approx -C_P \times \frac{V_{DD}}{T_0},$$

where T_i is the input transition time and T_O is defined by Eq.(2.2).

For $t > t_m$, the supply current is calculated using an integration method because accurate output voltage is needed to evaluate i_P in the triode region. $v_o(t_m)$, $i_P(t_m)$

and $i_N(t_m)$ obtained with the CP model are used as initial conditions to compute the supply current i_{PS} using the first order Euler integration method. The derivative of the output voltage at each time point is given as:

$$\frac{dv_o}{dt} = \frac{i_P(t) - i_N(t) - C_M \frac{V_{DD}}{T_i}}{C_N + C_P + C_M}$$

In order to avoid instability during the integration process, it is required that the following conditions should be satisfied.

$$\frac{dv_o}{dt} \geq 0,$$

$$v_o(t + \delta t) \geq v_o(t),$$

$$v_o \leq V_{DD}$$

The integration stops when v_o reaches V_{DD}. It is shown by experimental results that accuracy considerations in computing i_{PS} always limit the step size to a value where instability cannot occur. A typical integration time step is found to be:

$$\delta t = 0.02 \times T_i$$

2.3 Inverter with rising input: DP model

The supply current associated with a falling output voltage is characterized by a negative "peak" followed by a positive one. The DP model is used to compute the latter, while the DN model is design for the former.

For a discharging inverter, the supply current i_{PS} is dominated by the short circuit current i_P, which reaches it maximum at the pinch-off point of the pMOS transistor. To determine the time t_{mp} at which maximum i_P occurs, a similar approach is used to that for the CP model. Thus t_{mp} is computed from Eq.(2.3) where the coefficients

a and b, used to determine T_o from Eq.(2.2), are replaced by a different set, c and d, respectively. The maximum current calculations proceeds as in the CP model.

Since the output voltage at time t_{mp} has not yet reached the linear region, we cannot use $v_o(t_{mp})$ and t_{mp} to compute the delay. However, at time t_{stn} when the nMOS leaves the saturation region, the output voltage can be taken as linear and hence it can be used to calculate the delay. In this case, another ramp approximation is needed to predict $v_o(t_{stn})$ at t_{stn} accurately. Thus,

$$t_{stn} = \frac{T_i(1 + FB_n + \frac{V_{tnO}}{V_{DD}})}{1 + \frac{T_i}{T_{O2}}(1 + FB_n)}$$

where

$$T_{O2} = eT_i + f\frac{V_{DD}(C_N + C_P)}{I_N} \tag{2.5}$$

V_{tnO} is the threshold voltage for nMOS transistor corresponding to modified HSPICE model, FB_n is a HSPICE level-3 technology constant for the nMOS transistor, e and f are empirical constants, and I_N is the drain current for the nMOS transistor at v_{GS}=v_{DS}=V_{DD}.

Now, the delay can be computed as:

$$\tau = t_{stn} + \frac{\frac{V_{DD}}{2} - v_{On}}{\alpha_{stn}} - \frac{T_i}{2}$$

where

$$v_{On} = V_{DD}(1 - \frac{t_{stn}}{T_{O2}})$$

and

$$\alpha_{stn} \equiv \frac{dv_O}{dt}\mid_{t=t_{stn}} = -\frac{i_P(t_{stn}) - i_N(t_{stn}) + C_M\frac{V_{DD}}{T_i}}{C_N + C_P + C_M}$$

Since the charging current contributes the major part of the total supply current, the accuracy in computing the total current in a combinational circuit depends mainly on accurate calculation of the charging supply current. Therefore, we do not have to develop a "DW" model to approximate the discharging current waveform accurately. The DP model is used in all discharging cases, and the waveform is approximated by a triangle(Section 2.5).

2.4 Computing parasitic capacitances in an inverter

In this section, we review how the parasitic capacitances associated with an inverter are evaluated and replaced with constant values. A pMOS transistor is chosen as an example to demonstrate the evaluation processes which are equally valid for nMOS transistor.

Computing C_{GP}:

The capacitance C_{GP} of a pMOS transistor comprises the effects of gate-to-bulk, gate-to-source and gate-over-channel capacitances. Since a pMOS transistor with falling input is in saturation for most of the transition time, C_{GP} is computed as:

$$C_{GP} = CF5 \cdot Cap_p + CGSOp \cdot W_p + CGBOp \cdot L_p \quad for\ falling\ inputs \tag{2.6}$$

By the same token, C_{GN} is given as:

$$C_{GN} = CF5 \cdot Cap_n + CGSOn \cdot W_n + CGBOn \cdot L_n \quad for\ rising\ inputs \tag{2.7}$$

where

$$Cap_p = \frac{\epsilon_{OX}}{t_{OX}} \cdot W_p \cdot L_p,$$

$$Cap_n = \frac{\epsilon_{OX}}{t_{OX}} \cdot W_n \cdot L_n,$$

$L_p(L_n)$ and $W_p(W_n)$ are respectively, the channel length and device width of the pMOS(nMOS) transistor, t_{OX} is the gate oxide thickness, ϵ_{OX} is the oxide dielectric constant, $CF5$ is a user-definable constant in HSPICE with a default value of 2/3, $CGSOp(CGSOn)$ and $CGBOp(CGBOn)$ are the gate-to-source and gate-to-bulk overlap capacitances per meter of channel width for the p-type(n-type) transistor. When transistors operate in triode region, C_{GP} and C_{GN} are computed as:

$$C_{GP} = Cap_p \cdot CF5 \cdot K_{vp} + CGDOp \cdot W_p + CGBOp \cdot L_p \quad for\ rising\ inputs$$

$$C_{GN} = Cap_n \cdot CF5 \cdot K_{vn} + CGDOn \cdot W_n + CGBOn \cdot L_n \quad for\ falling\ inputs$$

where K_{vp} and K_{vn} are voltage-dependent coefficients used in HSPICE capacitance equations. From HSPICE simulations, K_{vp} equals to K_{vn} which has a average value 3/4 provided $CF5 = 2/3$.

Computing C_P:

The equivalent capacitance C_P(C_N for nMOS) includes the drain-to-bulk capacitance C_{DB} and the input capacitances of the loading gate(s). The drain-to-bulk capacitance of a transistor can be evaluated as follow:

$$C_{DB} = K_c(C_j \cdot AD + C_{jsw} \cdot PD) \tag{2.8}$$

where C_j, C_{jsw}, AD, PD are HSPICE parameters, and K_c is a constant determined by using HSPICE simulation. For 1.2 micron technology, $K_c = 0.615$ for pMOS transistor and $K_c = 0.127$ for nMOS transistor. The total input capacitance C_{IN} is equal to:

$$C_{IN} = C_{GP} + C_{GN} + \overline{C_M}$$

where C_{GP}, C_{GN} and C_M are equivalent capacitances of the loading gates, and $\overline{C_M} \approx 1.5 \cdot C_M$ for 1.2 *micron* technology. To account for the overlap currents between the driving and loading gates which will be discussed in Section 2.5, the capacitances C_{GP} and $C_{GN} + \overline{C_M}$ of the loading gate are added, respectively, to C_P and C_N of the driving gate.

Computing C_{MP}:

When a pMOS transistor is in triode region, the gate-to-drain capacitance is given by:

$$C_{GDP} = Cap_p \cdot CF5 \cdot K_{vp} + CGDO_p \cdot W_p$$

In saturation, the drain side of the channel is pinched off, and hence only the second term which corresponds to overlap capacitance contributes. From simulation, the average value of K_{vp} and K_{vn} for 1.2 *micron* technology are respectively,

$$\overline{K_{vp}} = 0.43/CF5, \text{ and } \overline{K_{vn}} = 0 \text{ for a rising input}$$

$\overline{K_{vn}} = 0.43/CF5$, *and* $\overline{K_{vp}} = 0$ for a falling input.

Therefore,

$$C_{MP} = \begin{cases} 0.43 \cdot Cap_p + CGDOp \cdot W_p & \text{for a rising input;} \\ CGDOp \cdot W_p & \text{for a falling input.} \end{cases}$$

and

$$C_{MN} = \begin{cases} CGDOn \cdot W_n & \text{for a rising input;} \\ 0.43 \cdot Cap_n + CGDOn \cdot W_n & \text{for a falling input.} \end{cases}$$

2.5 Current calculation in an inverter chain

In an inverter chain, the capacitive current i_{GP} of the loading inverter overlaps with the supply current of the driving inverter. In order to account for this overlap in computing the total supply current of a chain, all the input capacitances C_{GP} and C_{GN} of the loading inverter are added, respectively, to C_P and C_N of the preceding inverter. This is especially advantageous, since the negative current peak DN associated with the discharging inverter is absorbed in the charging current of the preceding inverter and hence, it is unnecessary to compute DN explicitly. Now, the discharging current comprises only one current peak DP which can be approximated by a single triangle as explained below.

Suppose T_j and L_j represent the input transition time and its start time of the $j-th$ inverter in an inverter chain. Using the CP and DP models, the output transition time T_{j+1}, the delay τ_j, and the supply current maximum I_{maxj} and its time of occurrence t_{mj} for the $j-th$ inverter can be computed. Moreover, the time U_j at which the output ramp T_{j+1} settles can be computed as:

$$U_j = L_j + \tau_j + 0.5 \cdot (T_j + T_{j+1})$$

Since the supply current of a CMOS inverter only flows during the inverter's input and output switching time intervals, L_j and U_j can approximate the lower and upper

bounds of the time interval in which the current of the $j-th$ inverter occurs. Therefore, the supply current of the $j-th$ inverter can be approximated by a triangle which is zero at L_j and U_j, and with a peak equals I_{maxj} at t_{mj}. The total supply current waveform of the inverter chain can be obtained by summing up the triangle current of every inverter in the chain.

2.6 Collapsing of a general CMOS gate into an inverter

The maximum current and delay evaluation of a general CMOS gate can be achieved by collapsing it into an equivalent inverter and then applying the CP and DP models. The collapsing involves finding the equivalent input transition time T_{IN}, the effective width of series/parallel transistors and finally the equivalent parasitic capacitances of the transistor group. The effective input transition time T_{IN} is determine from the group that carries both short circuit and dynamic currents, because this group dominates in determining the maximum current and the delay time. For simplicity, all the lengths of the transistors are assumed identical here. There are two sets of collapsing techniques that are used, respectively, with series-connected transistor group and parallel-connected group. These two techniques can be used recursively on a complex transistor group until a single transistor is obtained.

2.7 Collapsing a series-connected group of transistors

Suppose that the series-connected transistor group that is turning on comprises n transistors. Let T_j and t_j be, respectively, the input transition time and input start

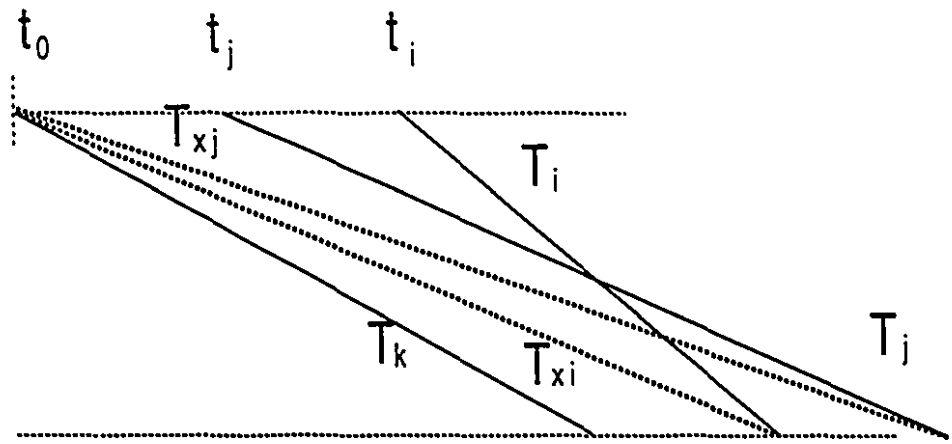


Figure 2.2: A general case for the input signals in a series chain when $q = 3$. Here i, j, and $k \in \{1, 2, ..., q\}$. The signal T_{xj} is used as the effective input for the equivalent transistor which replaces the group[29].

time of the $j-th$ transistor. Since the transistors are series-connected, this group would not turn on until the latest input, starting at t_L, turns on its associate transistor. In practice, there may be a number of switching inputs overlapping with t_L as illustrates in Figure 2.2. The earliest start time of these overlapping inputs, t_0, is chosen as the equivalent input start time and the equivalent input transition time, T_{IN}, is computed by using the following heuristic. Suppose there are $q-1$ inputs overlapping with t_L, and let $T_{xj} \equiv T_j + t_j - t_0$, then

$$T_{IN} = max(T_{x1}, T_{x2}, ..., T_{xq})$$

2.7.1 Width collapsing

In general, the effective width of the series-connected group depends on the output load, the sizes of the transistors, the input transition times, relative delays of the inputs as well as the positions of the inputs in the series chain. In this section, we use the pMOS transistor group to illustrate how to find its effective width W_{pc}. Similar procedures

are applied to nMOS transistor group.

The dependency of W_{pc} on the input signals and the sizes of the transistor is accounted for by expressing it as a function of T_{IN} and W_{pg} where

$$\frac{1}{W_{pg}} = \frac{1}{W_1} + \frac{1}{W_2} + ... + \frac{1}{W_n}$$

For very fast or slow T_{IN}, W_{pc} does not depend on the output load. Because, for fast T_{IN}, the output voltage is still negligible at the time when the inputs have completed switching, and for slow T_{IN} only negligible current flow through the load. Furthermore, for fast T_{IN}, W_{pc} does not depend on input positions, because when T_{IN} completes switching, the gate voltages of all transistors become identical. So, W_{pc} for fast and slow T_{IN} can be expressed by:

$$W_{fc} = c_{fc} \cdot W_{pg}$$

$$W_{sc} = c_{sc} \cdot W_{pg}$$

where W_{fc} and W_{cs} are defined as, respectively, the effective width corresponds to fast and slow T_{IN}, whilc c_{fc} and c_{sc} are constants depending on technology and the number of transistor in the series chain. In particular, c_{sc} also depends on the input signal position in the series chain and is determined from a look-up table such as the one in Table 2.2.

As T_{IN} increases, W_{pc} remains equal to W_{fc} until a "break point" T_B is reached. At this point, the input is so slow that the output voltage change is not negligible before the input transition is completed. This input break point T_B is a function of the output capacitance and the transistor sizes. For $T_{IN} > T_B$, W_{pc} is modeled as:

$$W_{pc} = W_{fc} + (W_{sc} - W_{fc}) \cdot (1 - exp[-B \cdot (T_{IN} - T_B)]) \tag{2.9}$$

where

$$T_B = c\frac{C_N + C_P}{W_{fc}}$$

Both c and B are empirical constants and C_N and C_P are the equivalent capacitances described in Figure 2.1.

Active input(s)	c_{sn}					
	# of pMOS transistors			# of nMOS transistors		
	2	3	4	2	3	4
1	1.82	2.58	3.04	1.78	2.50	3.06
2	1.20	1.75	2.26	1.48	2.00	2.50
3		1.27	1.72		1.63	1.20
4			1.34			1.60
1, 2	1.02	1.43	1.68	1.05	1.50	1.80
1, 2, 3		0.97	1.24		1.10	1.06
1, 2, 3, 4			1.08			0.88

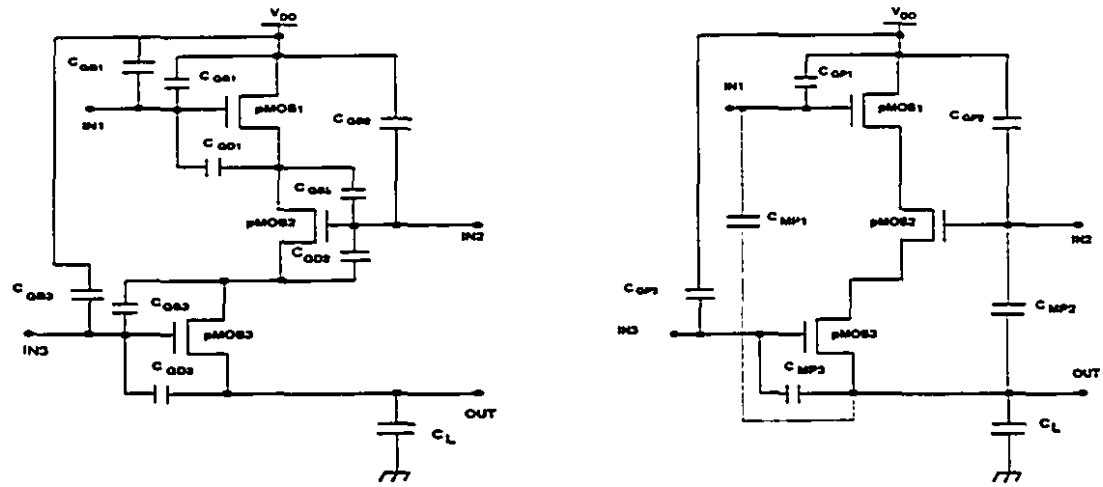
Table 2.2: Coefficients to obtain W_{sc} from W_{pug} (W_{pdg}) for a 1.2 *micron* technology. The input node number increases from the one nearest V_{DD} (GND)[29].

2.7.2 Capacitances collapsing

The techniques used to reduce the parasitic capacitances in series-connected transistors to those in the inverter model in Figure 2.1 are reviewed in this section. As shown in Figure 2.3(b), there are two equivalent capacitances, namely C_{GP} and C_{MP}, associate with the input of each transistor in a string of three pMOS transistors. In general, the total capacitive contribution of each transistor in the string consists of the above mentioned equivalent capacitances as well as its drain-to-bulk and source-to-bulk capacitances. First of all, the techniques used to compute $C_{GP}(C_{GN})$ and $C_{MP}(C_{MP})$ for pMOS(nMOS) transistor is presented.

Computing C_{GP}:

The C_{GP} of each transistor includes the effects of the respective gate-to-bulk, gate-to-



(a) (b)

Figure 2.3: Parasitic capacitances reduction in three series transistors. (a) Parasitic capacitances in three series pMOS transistors. Drain(source)-to-bulk capacitances are not shown. (b) Equivalent capacitances after capacitance reduction in (a)[29].

source, and gate-to-drain capacitances of the individual transistors. In particular, the gate-to-bulk capacitance can be obtained similarly as that in an inverter. However, the gate-to-source and gate-to-drain capacitances vary with the number of transistors connected between V_{DD} and its source. Therefore, simulation is used to find $C'_{GP} = C_{GS} + C_{GD}$ of each transistor in a string. Moreover, a look-up table is constructed to store C'_{GP} of each transistor for various string lengths. Thus, given a transistor with arbitrary width W_p, its C'_{GP} can be obtained by weighting the reference C'_{GP} from the look-up table with W_p/W_{po}, where W_{po} is the reference transistor width.

Computing C_{MP}:

A similar approach is used to compute $C_{MP}(C_{MN})$ of each transistor in a string. Once each individual $C_{MP}(C_{MN}$ for nMOS) is computed, the C_{MP} of the equivalent transistor

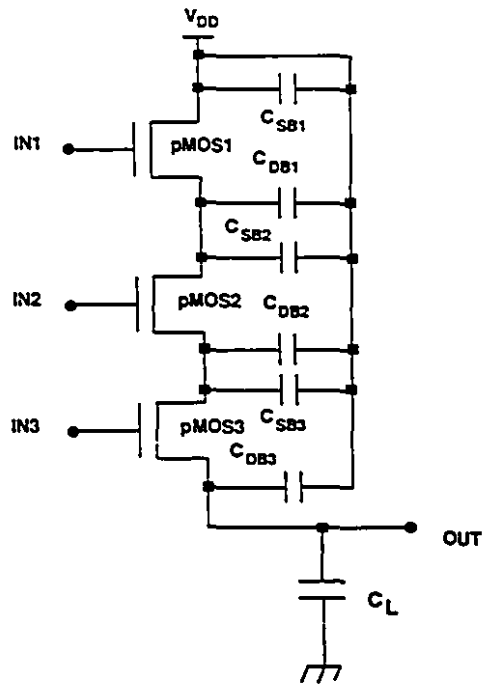


Figure 2.4: Drain and source-to-bulk capacitances in three series pMOS transistors[29].

is calculated from:

$$C_{MP} = \frac{T_{IN}}{V_{DD}} \sum_{j=1}^{q} i_{CMPj} \tag{2.10}$$

where the set $\{1, 2, ..., q\}$ $(q \leq n)$ represents the active inputs within the switching time of T_{IN}, and

$$i_{CMPj} = \begin{cases} C_{MPj} \times \frac{V_{DD}}{T_j} & \text{when } t_j < t \leq t_j + T_j; \\ 0, & \text{otherwise.} \end{cases} \tag{2.11}$$

Computing C_P:

The drain and source-to-bulk capacitances of all transistors in the series chain have to be combined to give the capacitance C_P(C_N for nMOS transistors) in the equivalent inverter. Suppose we have a string of three pMOS transistors as illustrates in Figure 2.4 and we define $C_1 \equiv C_{DB1} + C_{SB2}$, $C_2 \equiv C_{DB2} + C_{SB3}$, $C_3 \equiv C_{DB3}$. It is clear that the

contribution of C_j of the $j-th$ transistor to the effective combined capacitance $C_P(C_N$ for nMOS) depends on its relative position in the string. As a result, C_P is computed as follows:

$$C_P = \sum K_j C_j \tag{2.12}$$

where K_j is a constant determined from HSPICE simulation. Since the initial voltages at the internal nodes for falling inputs is different from that for rising inputs, the $K_j's$ for falling inputs assume different values from that for rising inputs. Besides, the switching activities at the input nodes determine which C_j contributes to C_P. For instance, if IN2 switches from low to high but IN3=0, regardless the state of IN1, C_P is obtained as the sum of C_2K_2 and C_3K_3.

2.8 Collapsing a parallel-connected group of transistors

Suppose we have a group of n transistors in parallel. Let T_j and t_j be the input transition time and start time of the $j-th$ transistor. If this transistor group carries both dynamic and short circuit current, then the effective input is determined by this group. Since the supply current flows as soon as the first input arrives, the effective start time, t_0, is equal to the start time of the earliest input. Due to the fact that all transistors that are turning on can charge(discharge for nMOS) its output node, the effective input transition time T_{IN} is computed as follows:

$$T_{IN} = min(T_{x1}, T_{x2}, ..., T_{xq}) \tag{2.13}$$

where the subscripts $1, 2, ..., q$ $(q \leq n)$ represent the transistors that are switching on. See Section 2.7 for the definition of T_{xj}.

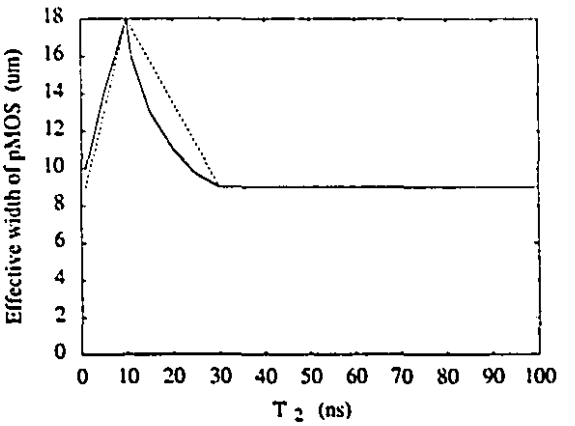


Figure 2.5: Effective width for parallel transistors in a 2-input NAND gate ($W_{p1} = W_{p2} = 9.0\mu m$, $L_p = L_n = 3.0\mu m$). $T_1 = 10ns$, HSPICE (—), and approximation (- - -). Both signals starts at t=0[29].

2.8.1 Width collapsing

The effective width W_{peff} of the parallel transistors depends on the sizes of the devices, their input transitions and relative delay times. This is illustrated in Figure 2.5 which shows the effect of the relative input transition times in a 2-input NAND gate, on the effective width of the equivalent pMOS transistor. This was obtained from simulation by adjusting the inverter size to give approximately the same the current waveform as in the NAND gate.

Let W_{p1} and W_{p2} be the widths of the pull-up transistors in the 2-input NAND gate, and the corresponding input transition time be, respectively, T_1 and T_2. As illustrates in Figure 2.5, the equivalent width is a nonlinear function of T_1 and T_2. However, this function can be approximated by a triangle with a peak value of $W_{p1} + W_{p2}$ as shown with dotted line in Figure 2.5. For a NAND gate with a rising output, the effective input signals $T_{xj} \geq T_{IN}$, because T_{IN} is the fastest input. On the other hand, $T_{xj} \leq T_{IN}$ for a NAND gate with a falling output. Therefore, in each case only one

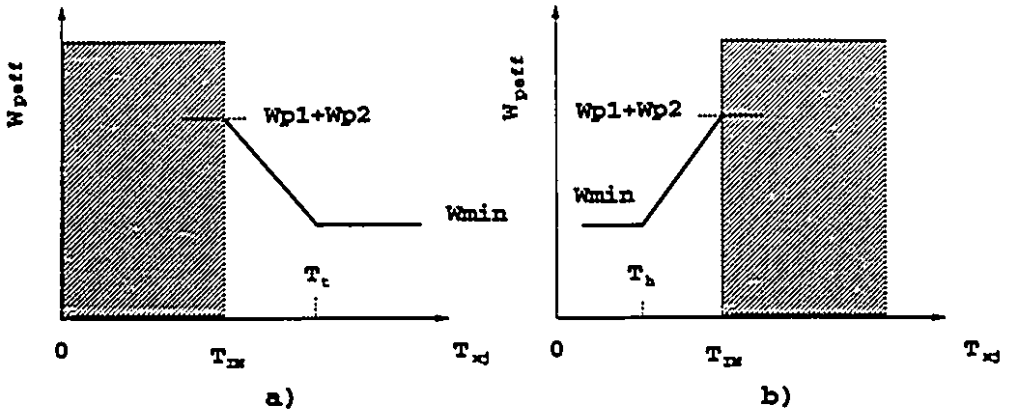


Figure 2.6: The effective width for the two parallel transistors in a 2-input NAND gate, versus T_{xj} for: (a) rising output, (b) falling output. In each case one transistor is driven by T_{IN}, while the other one is driven by T_{xj} which never crosses into the shaded area. Moreover, W_{min} is the width of the transistor driven by T_{IN}[29].

side of a triangle is needed as illustrate in Figure 2.6. Thus, for a 2-input NAND gate, the effective width W_{peff} can be formulated as follows:

$$
\begin{aligned}
&If\ \ T_{IN} = T_1\ \ and\ \ T_2 \geq T_1 : \\
&\qquad (i)\ \ If\ \ T_2 \geq T_t,\ \ W_{peff} = W_{p1} \\
&\qquad (ii)\ \ If\ \ T_{IN} \leq T_2 < T_t,\ \ W_{peff} = W_{p1} + \frac{(T_t - T_2)}{(T_t - T_{IN})} W_{p2}
\end{aligned}
\tag{2.14}
$$

$$
\begin{aligned}
&If\ \ T_{IN} = T_2\ \ and\ \ T_2 \geq T_1 : \\
&\qquad (i)\ \ If\ \ T_1 \geq T_h,\ \ W_{peff} = W_{p2} \\
&\qquad (ii)\ \ If\ \ T_h < T_1 \leq T_{IN},\ \ W_{peff} = W_{p2} + \frac{(T_1 - T_h)}{(T_{IN} - T_h)} W_{p1}
\end{aligned}
\tag{2.15}
$$

where T_h and T_t are found from experiment with values of $T_{IN}/2$ and $3 \cdot T_{IN}$ respectively.

By extending the results for 2-input NAND gate to a n-input gate, the following

heuristic is obtained.

$$W_{peff} = \sum_{1}^{q} S_j W_j \tag{2.16}$$

where W_j is the width of the transistor corresponding to the $j-th$ input, subscripts $1, 2, ..., q$ represent the input signals that overlap with T_{IN}, and S_j is determined as follows:

$$S_j = \begin{cases} 2T_{xj}/T_{IN} - 1, & \text{if } (T_h < T_{xj} \leq T_{IN}); \\ \frac{3}{2} - \frac{1}{2}T_{xj}/T_{IN}, & \text{if } (T_{IN} \leq T_{xj} \leq T_t); \\ 0, & \text{otherwise.} \end{cases} \tag{2.17}$$

2.8.2 Capacitances collapsing

In Figure 2.7(a), the parasitic capacitances associated with a parallel-connected transistor group are identified. The goal of this section is to show how these parasitic capacitances are reduced to the equivalent capacitances as shown in figure 2.7(b).

Computing C_{GP}:

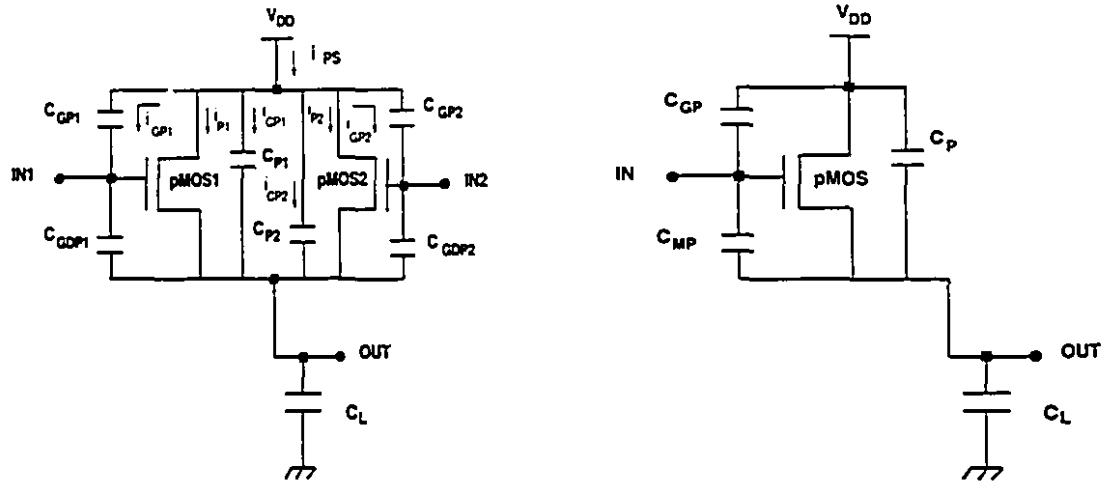
The capacitance C_{GPj} of each transistor comprises the effects of gate-to-bulk, gate-to-source and gate-over-channel capacitances. Since the bulk and source of each transistor in figure 2.7(a) are connected to V_{DD}, C_{GPj} is computed exactly the same as that for an inverter. See Section 2.4 for details.

Computing C_P:

The equivalent capacitance C_P(C_N for nMOS) includes the total capacitance between the output node and V_{DD}(GND). This implies $C_P(C_N)$ contains the drain-to-bulk capacitance C_{DB} of every pMOS(nMOS) transistor in the parallel transistor group as well as the total input capacitance C_{IN} of the loading gate(s). The drain-to-bulk capacitance of a transistor is evaluated exactly the same as that for an inverter. See Section 2.4 for the details.

Computing C_{MP}:

The gate-to-drain capacitance C_{GDPj} of each transistor in figure 2.7(a) are combined



(a) (b)

Figure 2.7: Parasitic capacitances reduction in two parallel transistors. (a) Parasitic capacitances in two parallel pMOS transistors. (b)Equivalent capacitances after capacitance reduction in (a)[29].

to give C_{MP} in figure 2.7(b). C_{GDPj} is computed exactly the same as that for an inverter and the combined C_{MP}(C_{MN} for nMOS transistors) of every transistor is found similarly as the series-connected transistor group.

Chapter 3

CURrent ESTimation Algorithm: CUREST

The current estimation algorithm, CUREST, is built to compute the current waveform of a circuit automatically, provided the circuit topology and all the primary input transition information are given[1]. This information is described in a format similar to that in HSPICE and is parsed by a timing analysis tool called TAMIA. The parsed information, which is stored in TAMIA data structures, is extremely useful for automating the current and delay evaluation. Thus, we build our automation tool within TAMIA and utilize the circuit decomposition information generated by TAMIA to carry out maximum current and delay analysis. In this chapter, a brief review of TAMIA is given, which is followed by the overview and detail of the architectural feature of CUREST.

[1]The input format for CUREST is described in Appendix B

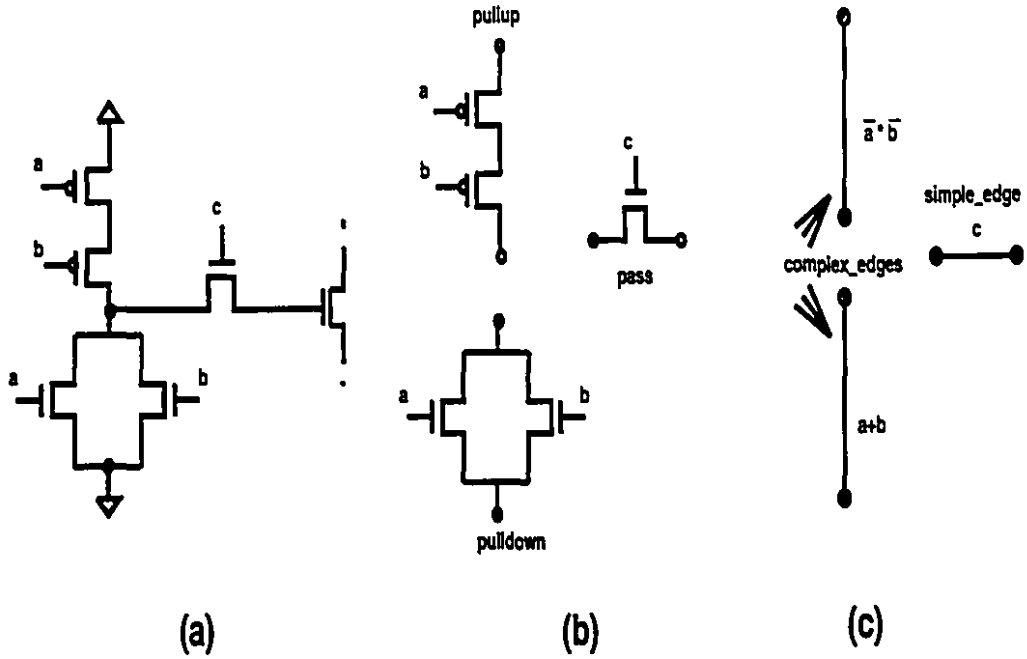


Figure 3.1: The circuit decomposition:(a) a transistor group, (b) transistor subgroups (c) virtual edges

3.1 The timing analyser TAMIA

TAMIA is a circuit-level timing analysis tool for the design of VLSI digital MOS circuits[31]. In order to increase the efficiency of the timing analysis, the dependencies among the nodes in the circuit need to be determined, and hence the decomposition of the circuit into a directed circuit graph is necessary. A circuit graph is composed of transistor groups which are composed of "pullup", "pulldown" and "pass transistor" subgroups. The pullup subgroups are between V_{DD} and output node, the pulldown subgroups are between GND and the output node, and the pass transistor subgroup is in between the input node and output node as shown in figure 3.1.

3.1.1 Circuit Decomposition

A depth-first search is used to carry out the circuit decomposition. Starting at the primary outputs of the circuit, all the transistors connected together through their channels or through interconnection resistors are grouped together. Once a group is identified, the search continues from the inputs of this group until primary inputs are reached.

A transistor group is decomposed by a recursive decomposition algorithm which splits a transistor group into virtual edges. Each virtual edge can be a combination of series or parallel transistors, or just simply one transistor. If an edge consists of more than one transistors it is called a *complex edge*, otherwise it is called a *simple edge*. The circuit decomposition continues until all the complex edges have been recursively decomposed to simple edges. After the circuit decomposition is done, all the transistor groups are identified and stored in a list called a *group_list*. Each entry in the group list represents a transistor group from which the virtual edges, including both the simple and complex edges, can be obtained. Once a virtual edge is obtained, the transistor size and input information associated with its components can be accessed easily by extracting the proper information stored in the TAMIA data structures. Other useful information generated from TAMIA includes the *primary input list* which contains all the primary input node names/numbers and their associated input transition information. The information as described above is used as input to CUREST.

3.2 The overview of CUREST

The objective of CUREST is to find the supply current waveform of a circuit based on its primary input pattern. This is achieved by carrying out the following procedures sequentially. First, the circuit topology description is parsed by TAMIA[2]. Then, the

[2]The parsing is done by the subroutines called *parse-circuit* and *recursive-parse-circuit*.

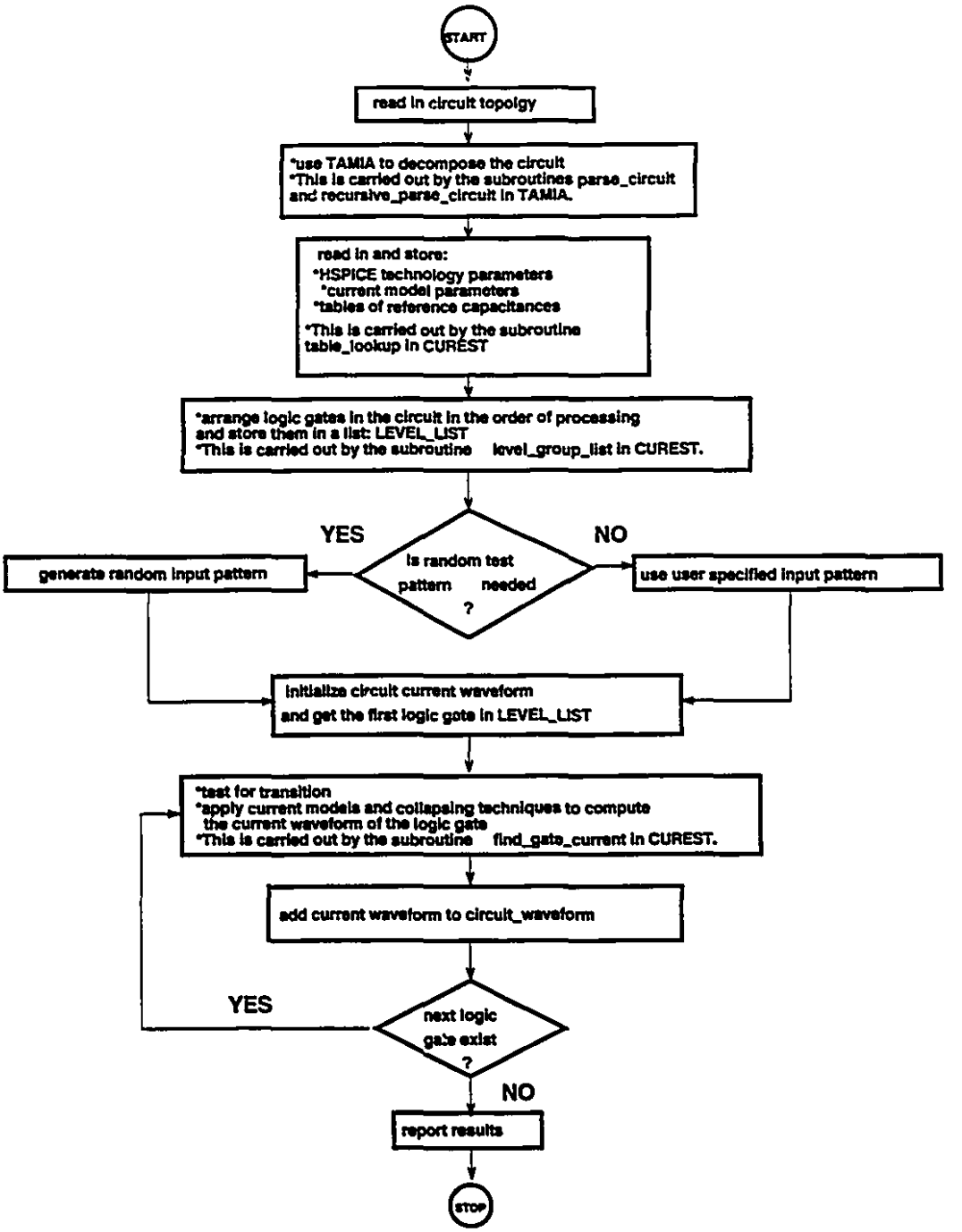


Figure 3.2: The flow diagram of the current estimation algorithm CUREST.

technology parameters, tables of reference capacitances and the current model param-

eters are read and stored. These parameters are needed later for the computation of individual gate current waveform. Since the transistor groups in the *group_list* is not in the order of processing, it is necessary to re-order the transistor groups following the procedures presented in Section3.4. After this is done, we can examine one gate at a time to see if it switches or not. If it does switch, we use the current models and collapsing techniques described in Chapter 2 to compute its current waveform. Otherwise, the next gate is examined. The computed current waveform of each gate is summed. This process repeats until all the gates in the circuit are evaluated.

Once all the gates are analysed, the circuit current waveform is obtained which is the sum of individual gate current waveform. From the total circuit current waveform, the maximum and average current can be readily determined. In Figure 3.2, the flow of the automation algorithm is presented, which is expanded in the following sections. At present, CUREST is capable of handling pure combinational circuits based on NAND/NOR static gates with no feedback.

3.3 Technology dependent parameters for current and delay estimation

CUREST is built within TAMIA. Therefore, the information gathered by TAMIA's circuit parser and circuit decomposition procedures can be shared with CUREST. Beside the necessary information provided by TAMIA, there are plenty of constants and tables of constants that are critical in carrying out the gate collapsing and current estimation operations. Since these constants are technology dependent, it is more efficient to store them in files. As a result, the program has the freedom to process circuits of different technology by just selecting the proper technology files. At present, the $1.2\mu m$ technology files are implemented and the collapsing parameters are applicable to gates with up to 4 inputs. These parameters are read and stored in arrays right

after the circuit decomposition is done by TAMIA.

3.4 Order of processing of logic gates in the circuit

During the circuit decomposition in TAMIA, a *group_list* is generated to store all the transistor groups in the circuit. The order of these transistor groups in the group list is determined by the depth-first search in TAMIA. Since the search starts at primary outputs and ends at primary inputs, it turns out that the loading gates appear before the driving gates. This is not the order in which the gates should be processed, because we cannot process a gate until all its inputs have already been generated by its driving gates. Thus, we have to rearrange the transistor groups to satisfy the order of processing. The procedures used to achieve the re-ordering is explained as follows.

In CUREST, we assign a number which is called a *level_value* to every logic gate output in the circuit. This number is used to distinguish the relative position of a logic gate in the circuit with respect to primary inputs. By default, a logic gate output initially assumes *level_value* of 0 which indicates that the inputs of the logic gate have not been processed yet. First we set the *level_value* of all primary inputs to 1. Then, we determine whether a logic gate is ready to be processed or not by examining the *level_values* of its inputs. An input is considered ready if its *level_value* is greater than 0. Therefore, if all the inputs of a logic gate assume *level_values* greater than 0, then we can conclude that the logic gate is ready to be processed. In this case, the *level_value* of the logic gate output equals to:

$$max(i_1, i_2, ..., i_n) + 1$$

where $(i_1, i_2, ..., i_n)$ is the set of input *level_values*. Once such a logic gate is found, it is removed from the *group_list* and stored in another list called *level_list*. The search continues until the *group_list* is empty. In Figure 3.3, the procedures involved in obtaining the proper processing order is summarized.

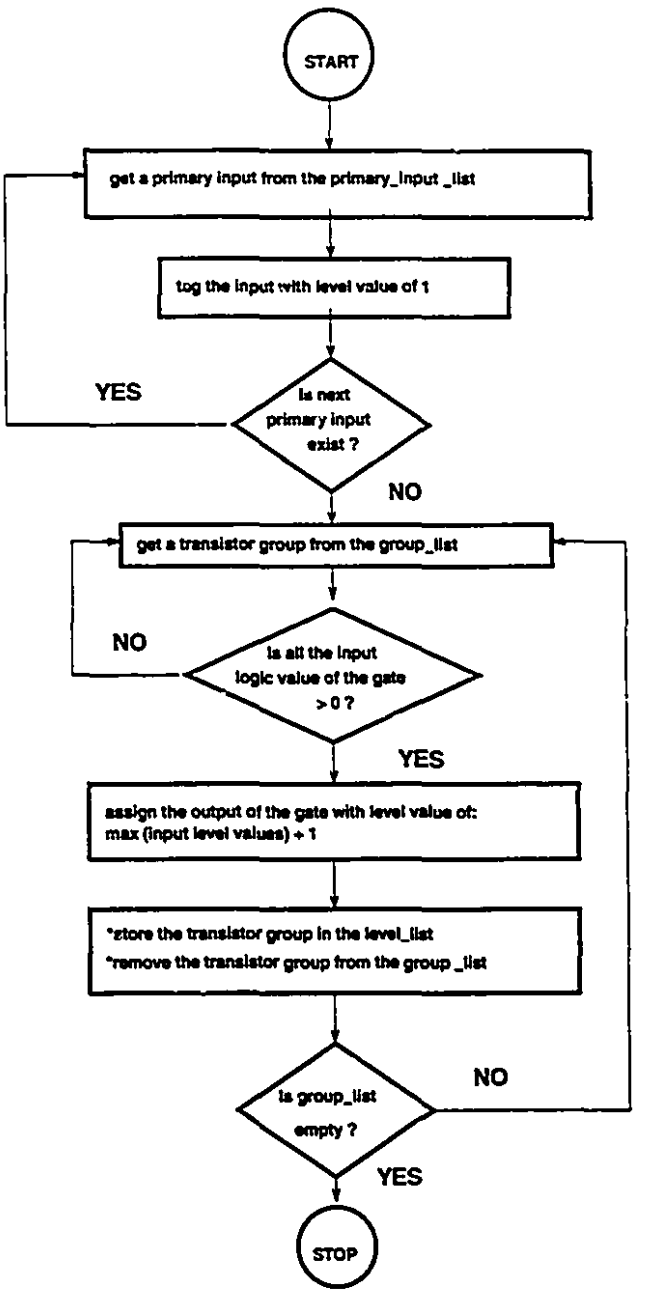


Figure 3.3: The re-ordering of the transistor groups in the group_list.

3.5 Random input generation

The inputs to the circuit can be either specified by the user in the input file or randomly generated within the program. There is an integer called *specify_input* in the input file

which is used to differentiate these two choices. If the user decides to use the randomly generated inputs, he has to set *specify_input=0*. Otherwise, *specify_input* assumes value of 1 which implies that primary inputs should be provided by the user.

If random inputs are needed, the user has the freedom of varying the input start time, input transition time and input transition types[3]. The user has to select from the eight possible combinations by assigning a predefined integer to a variable called *random_type*. The range of this variable varies from 0 to 7 which corresponds to the eight available combinations. By default, random_type is set to 4 which corresponds to vary the input transition types randomly, but keeping the input start time and input transition time unchanged. The user can also control the probability of occurrence of each transition type by setting the corresponding probability parameters to desired values[4]. There are four probability parameters, namely *pr-rise, pr-fall, pr-low* and *pr-high* which represents respectively, the transition probabilities of rising, falling, always-low and always-high. Furthermore, the user can control the number of randomly generated input vectors through a variable called *random_set*. For example, suppose that ten test patterns are required and we would like to vary the input start time, input transition time, and the input transition type, the latter being limited to either rising or falling. Then, it is necessary to assign the above variables with values: *specify_input=0, random_set=10, random_type=7, pr-rise=0.5, pr-fall=0.5, pr-low=0, pr-high=0.*

In particular, if the user decides to fix the input transition time and start time, and vary only the input transition type, (s)he has the choice of carrying out exhaustive analysis on the circuit provided the number of primary inputs is not more than 6. The user can make this choice by setting the variable *exhaustive* equals 1 in the input file. By default, *exhaustive* equals 0 which de-activates the exhaustive analysis procedure.

[3]There are four kinds of transition: always-low, always-high, low-to-high and high-to-low.

[4]The sum of these four probabilities must be equal to 1.

The user can control the range of the start time and transition time associated with the randomly generated input vectors. There are two variables, namely *transition_time* and *start_time* in the input file, that can be used to set the ranges as mentioned above. For example, if the user sets the *transition_time=5ns*, *start_time=2ns*, and *specify_input=0*, then the input vector generated will have input start times varying between 0 and 2ns, and transition times varying between Δt and 5ns. By default, $\Delta t = 1ns$.

3.6 Output logic evaluation of a CMOS logic gate

The current models are only applied to gates that are active. In CUREST, a gate is considered active if its output logic value, which can either assume value of 1 or 0, changes from its original value due to its active input(s). For a rising input transition, the logical values before and after the transition are respectively, 0 and 1, and vice-versa for falling inputs. By examining the logical value at the gate of a CMOS transistor, we can determine whether this transistor is on or off. The gate output logic value can then be determined by examining the on/off conditions of its components. For a series-connected transistor group, it is required that all transistors should be on in order to make this transistor group conducting. On the other hand, it is required that all transistors in a parallel-connected transistor group should be off to make this transistor group off. The algorithm *evaluate(subgroup)* is developed to test the on/off condition of a transistor subgroup using the strategy as mentioned above. Therefore, by examining the on/off conditions of the pull-up transistor subgroup, the output logic value of the logic gate is determined. This examination is carried out by the algorithm *logic_evaluation (logic_gate)* where the *logic_gate* is not limited to standard NAND/NOR logics. It could be a complex gate. The pseudo-code of these two algorithms are summarized as shown below.

```
logic_evaluation (logic_gate)
  { evaluate (pullup_subgroup);   /*check if pullup is ON*/
    if (pullup_subgroup is ON)
      { evaluate (pulldown_subgroup); /*check if pulldown is ON*/
        if (pulldown_subgroup is ON)
          { report design error;
            /*at static condition, pull-up and pull-down subgroups of
              complementary CMOS logics cannot be ON at the same time*/
            exit;
          }
        else
          return (logic_gate has logicvalue of 1);
      }
    else
      { evaluate (pulldown_subgroup); /*check if pulldown is OFF*/
        if (pulldown_subgroup is OFF)
          { report design error;
            /*at static condition pull-up and pull-down subgroups of
              complementary CMOS logics cannot be OFF at the same time*/
            exit;
          }
        else
           return (logic_gate has logicvalue of 0);
      }
  }

evaluate (subgroup)
  { if (subgroup is a simple transistor)
      { if (subgroup is ON)
           return (subgroup is ON);
        else return (subgroup is OFF);
      }
    else
      { /*assume there are n subedges in the subgroup*/
        for (i=1 to n)
           { if ((subgroup==SERIES) and (evaluate(subedge_i)==OFF))
                return (subgroup is OFF);
             if ((subgroup==PARALLEL) and (evaluate(subedge_i)==ON))
                return (subgroup is ON);
           }
        /*now, for series-connected subgroup all its subedges is ON and
          for parallel-connected subgroup all its subedge is OFF*/
        if (subgroup == SERIES)
           return (subgroup is ON);
        if (subgroup == PARALLEL)
           return (subgroup is OFF);
      }
  }
```

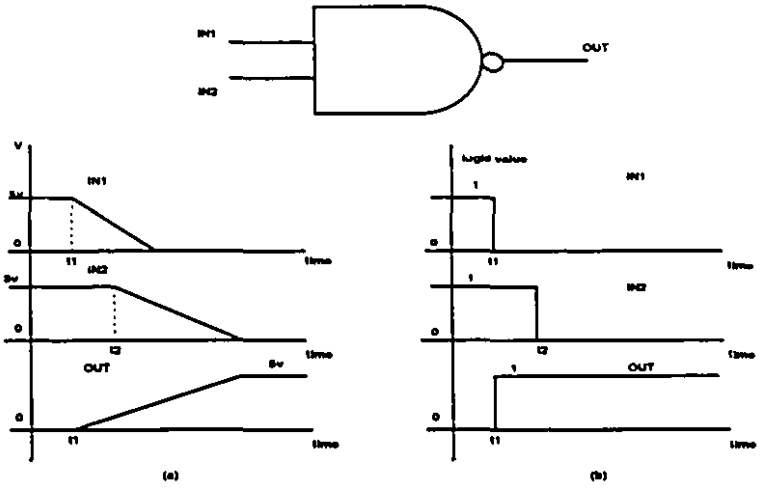


Figure 3.4: Logic evaluation of 2-input NAND gate with respect to its inputs. (a) The input and output transition changes. (b) The input and output logic changes.

3.7 The total circuit current waveform evaluation

Due to circuit delay, it may happen that a logic gate switches more than once. It is necessary to know the output transition type as well as the input(s) that are causing the switching before a proper current model can be selected to compute the associated current waveform. Suppose the input to a logic gate is represented by a pair (t_i, T_i) where t_i and T_i are, respectively, the input start time and input transition time. In CUREST, the correlation of the input and output transition is found by carrying out the following procedures. First of all, the gate output logic value is determined by *logic_evaluation* algorithm before any active input switches. This is done by assuming a falling input logic value is 1 and a rising input is 0 before they switch. Then, the earliest input pair (t_s, T_s) that may cause the output to switch is identified. This is achieved by arranging the active input pairs in ascending order of t_i and the pair (t_s, T_s) is found as the earliest falling(rising) input that may turn on the pull-up(pull-down) subgroup if the output logic value was at 0(1). Since all inputs that overlap (t_s, T_s) may charge or discharge the output node, it is necessary to collect all the overlapping

signals and evaluates the output logic value with respect to the overlap signals.

For example, in Figure 3.4, there are two overlapping signals at the 2-input NAND gate. In this case, the logic value of the output node is 0 before IN1 and IN2 switch and hence, the earliest input pair is IN1. It is assumed that the new logic value of the input is equal to its stabilized logic value right at the transition start time. Therefore, IN1 assumes logic value of 0 at time t_1 and IN2 assumes logic value of 0 at time t_2. As illustrate in Figure 3.4(b), the output logic value changes are obtained by carrying out logic_evaluation with respect to the active inputs one at a time starting with the earliest input pair (t_1, T_1). At time t_1, IN1 switches to low and IN2 is still high, and hence the output logic value switches to high. The next switching activity occurs at time t_2 where IN2 switches to low. Since both IN1 and IN2 are low, the output stays high.

From this analysis, the output transition type can be found and hence, a proper current model can be selected to compute its current waveform. If a charging output is detected, the user can either use the CW model or the CP to compute the corresponding current waveform. CW model is selected if the variable *integration* in the input file is assigned to 1. Otherwise, CP model is used because by default *integration* is assigned to 0. For discharging case, DP model is used. If the output switches more than once, we conclude a glitch occurs. The glitch current calculation will be discussed in Section 3.8.

Since the circuit is purely combinational with no feedback, only a single-pass over the logics in the circuit is needed to compute the total current waveform. The procedures involved in evaluating the current waveform of a logic gate as well as the whole circuit are summarized in Figure 3.5. Given a logic gate, the first thing we should do is to check if any of its input(s) is switching or not. If there is no switching input, we skip the gate and go to process the next one from the *group_list*. If switching inputs exist, we would arrange these inputs in ascending order respect to their input start time. Next, the output logic value of the gate before any of these input switches is determined. Once this is done, the earliest active input (t_s, T_s) that may induce the output to switch can be found. We collect all the active inputs that overlaps with (t_s, T_s) and store them in *overlap_list*. Then, logic evaluation is carried out with respect to inputs

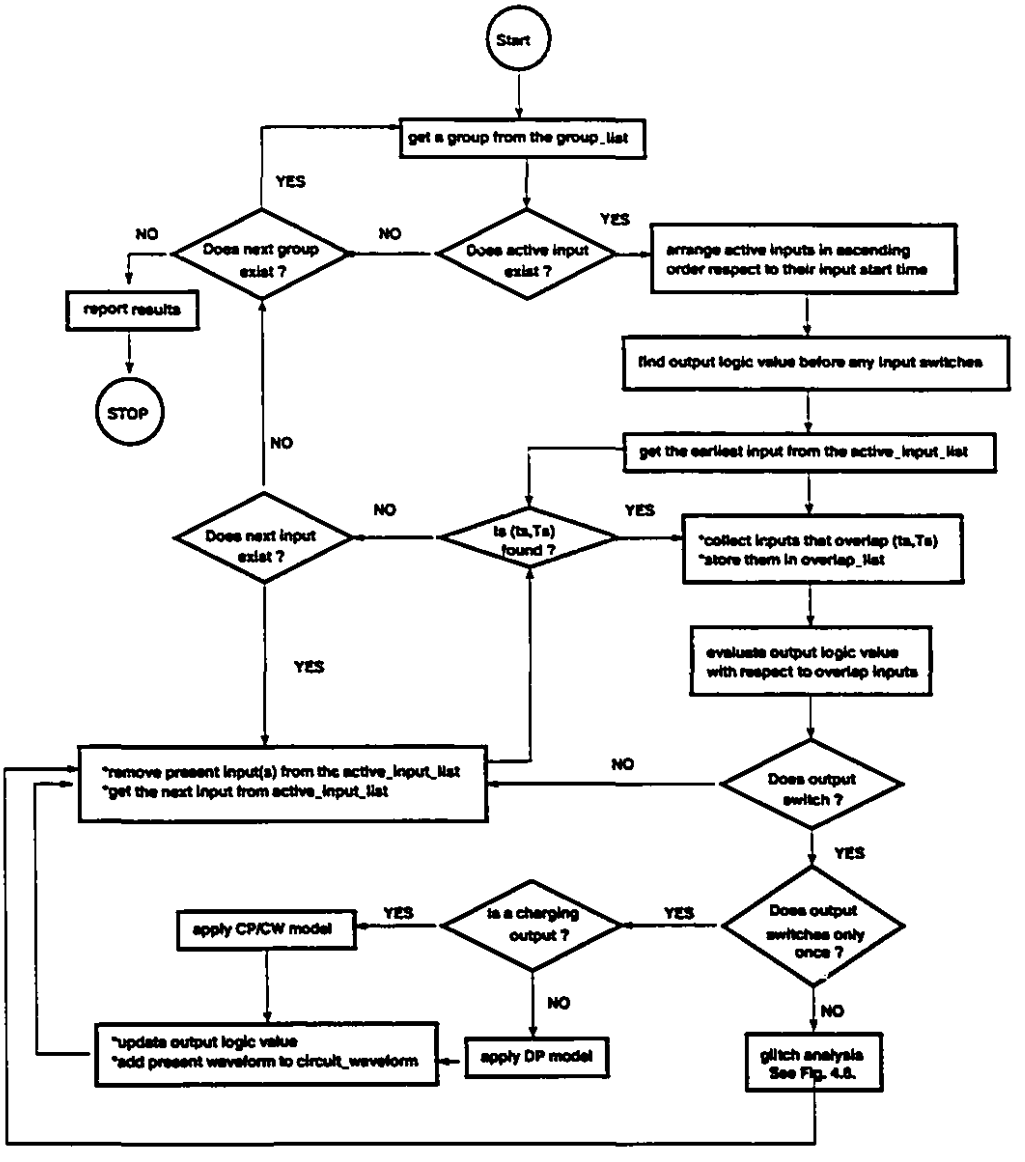


Figure 3.5: The flow diagram of circuit waveform evaluation.

in the *overlap_list*. If the output does switch, a proper current model is selected to compute the corresponding current waveform. The above procedures are repeated for the next logic gate until all the gates in the circuit are processed. The total current waveform of the circuit is obtained as the sum of all individual gate contributions.

3.8 Glitch current estimation of a logic gate

A logic gate with overlapping rising and falling inputs may produce a glitch at its output. We can detect a glitch by carrying out logic evaluation at a gate and count the number of transitions induced by the overlapping inputs. If the output switches more than once due to only one set of overlapping input signals, we conclude that a glitch occurs. In CUREST, we assume that the glitch output does not propagate because in practice glitches are quickly filtered out. As shown in Figure 3.6, the logic evaluation cannot detect glitch condition such as those in case (b) for which, at present, we do not have a good model to predict the current waveform. From HSPICE simulations shown in Table 3.1, glitch currents associated with overlapping inputs that cannot be detected by CUREST are relative small and hence we can ignore them without acquiring much error. For glitches that can be detected, such as those in Figure 3.6(a), we compute their current waveform by using the CP or CW model which is a fair approximation as

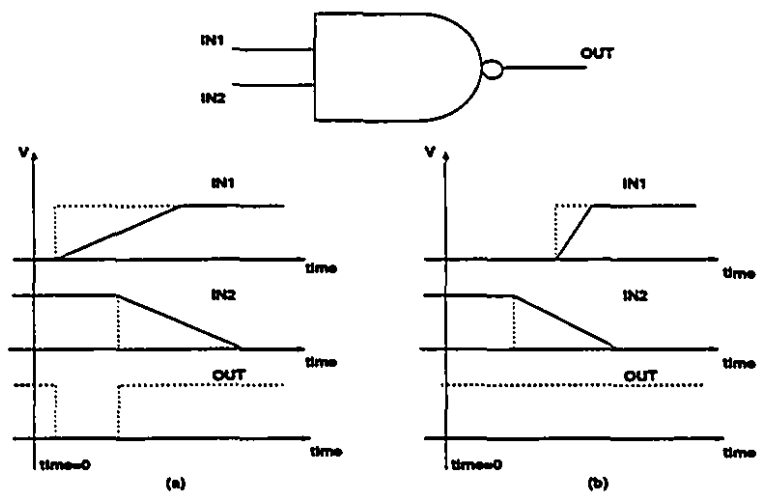


Figure 3.6: Logic evaluation of 2-input NAND gate with overlapping inputs. The dotted lines indicate the logic values of the inputs and outputs. (a) The output switches twice: high-low-high. (b) The output does not switch.

NAND-2				NOR-2			
IN1 lagging IN2		IN2 lagging IN1		IN1 lagging IN2		IN2 lagging IN1	
lag time(ns)	Imax (mA)	lag time(ns)	Imax (mA)	lag time(ns)	Imax (mA)	lag time(ns)	Imax (mA)
4.0	1.44	4.0	0.05	4.0	0.11	4.0	1.21
3.0	1.44	3.0	0.05	3.0	0.10	3.0	1.14
2.0	1.37	2.0	0.13	2.0	0.17	2.0	0.92
1.0	0.91	1.0	0.38	1.0	0.28	1.0	0.69

Table 3.1: HSPICE:Maximum glitch current drawn by 2-input NAND and NOR gates. IN1 falls with 5ns and IN2 rises with 5ns.

long as the overlapping signals are not extremely close or far apart. For example, by using the CP model, the maximum glitch current drawn of a 2-input NAND and NOR gates are 1.26 mA and 1.04 mA respectively. These values are compared to HSPICE results in Table 3.1.

To correlate the glitch output and its inputs, the following procedures are used. Suppose all the overlapping signals are stored in a list called *overlap_list*. First the output logic value V before any overlap input switches is determined. Then, the earliest signal T_1 that switches the output logic value is located and recorded in a list called *list_1* which records all inputs that are associated with the first switching activity of the output node. The inputs that precede T_1 are removed from the *overlap_list*. If the next signal that follows T_1 has the same transition type as T_1, it is recorded in *list_1* as well, because it also contributes to the first switching of the output node. Otherwise, logic evaluation is carried out with respect to this particular input. If this input induces the output to switch, it is recorded in a list called *list_2* which is the list used to store inputs that are associated with the second switching activity of the output node. However, if this input does not induce any output change, it is removed

from the *overlap_list*. The search goes on until an active input that induces the output changes for the third time is encountered. Since only CP or CW is used to compute the output current waveform, we have to select the list with inputs that charge the output node. The list is selected based on the output logic value V. For example, if $V = 0$, then *list_1* should be selected because it contains inputs that induce the first switching activity of the output node i.e. from 0 to 1 which is a charging activity. By the same token, if $V = 1$, *list_2* is selected.

Once the charging current waveform is computed, the two lists are cleared so that they can be used again for the next two output transitions. Besides, all inputs in the two lists are removed from the *overlap_list* as they are being stored in *list_1* and *list_2*. As the output node has made two complete transitions from its original logic value V, it is clear that the output logic value is still V before the next two transitions occur. Therefore, the inputs that are causing the next two transitions can be found by repeating the above procedures. This process continues until the *overlap_list* is empty. The flow diagram for glitch current evaluation is summarized in Figure 3.7. This algorithm is capable of handling complex gates.

The flow of this algorithm is illustrated by an example given as follow. Suppose we have a 2-input NAND gate with inputs IN1 and IN2 as shown in Figure 3.6(a). It is clear that the *overlap_list* will start with these two inputs and IN1 is the first input in the list. In this case, the output logic value is 1 before any of these two inputs switch. Since the charging has not occurred yet, the *charge_flag* is still equal to 0. Because IN1 is rising, it is stored in *list_1* before it is removed from the *overlap_list*. Now, IN2 becomes the first as well as the only input left in the *overlap_list* and output logic value is still 1. Since IN2 is a falling input, logic evaluation is applied to the pull-up transistor subgroup and it turns out charging occurs. Therefore, the *charge_flag* is set to 1 and IN2 is stored in *list_2* before it is removed from the *overlap_list*. At this point, the *overlap_list* is empty and hence, we can use the input in *list_1* together with the CP model to compute the glitch current waveform.

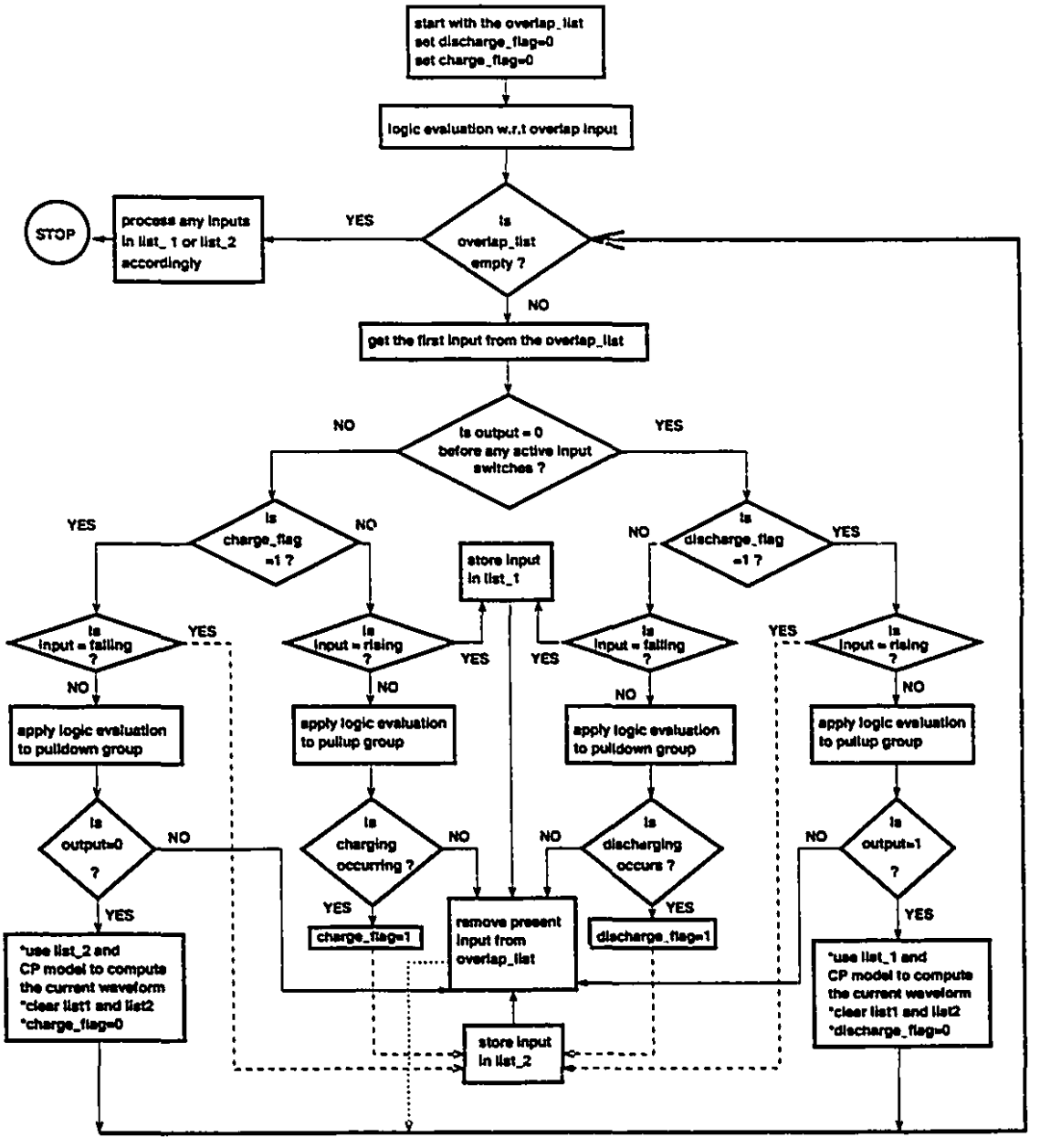


Figure 3.7: The flow diagram of glitch current evaluation.

3.9 Obtaining the total current waveform

The total current waveform is made up of a set of equally space current-time points. The number of current-time points is equal to the upper limit of the current waveform

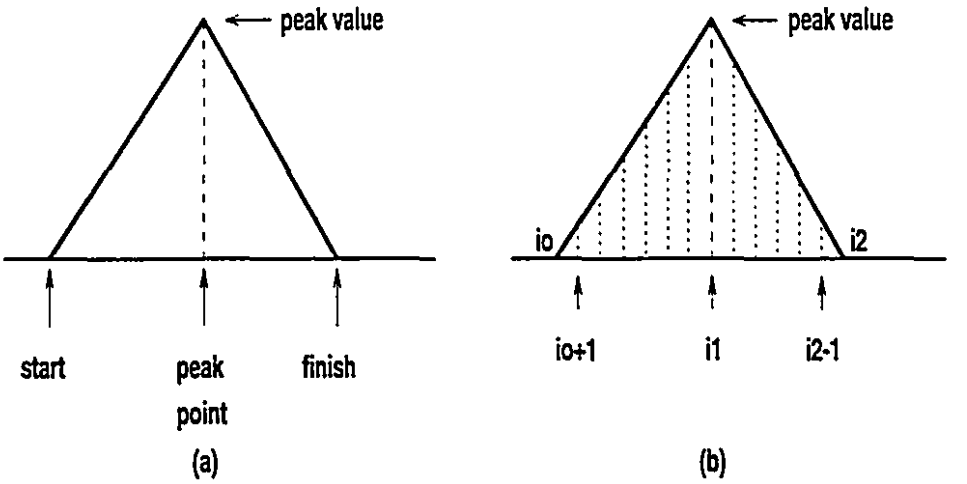


Figure 3.8: Current triangle obtained from CP or DP model.
(a)The four points required to uniquely define a current triangle.
(b) The array indices of the current triangle.

divided by the spacing between the current-time points. The user can change upper bound of the current waveform by setting the variable *stop_time* in the input file to a value to his desire. But, the lower bound of the current waveform is fixed to 0. The spacing between the current-time point is controlled by the variable *def_step* in the input file. The current-time points are stored in an array where the index of an array entry represents the number of time points away from the lower bound of the current waveform.

Suppose we have a current triangle obtained from the CP or DP model. This triangle can be uniquely defined by four points: the start point, finish point, peak point and the peak value as shown in Figure 3.8(a). To add this triangle to the total current waveform, first we have to determine the corresponding array indices of the start, finish and the peak points. The array indices shown in Figure 3.8(b) are found by rounding off the quotients of the the start, finish and peak time points with *def_step* to their nearest integers. Next, we add the triangle to the current array as follow:

$$array[i] = array[i] + \frac{peak\_value}{i_1 - i_0} \cdot (i - i_0) \; for \; i_0 < i < i_1$$

$$array[i] = array[i] + \frac{peak\_value}{i_2 - i_1} \cdot (i_2 - i) \; for \; i_1 < i < i_2$$

where $array[i]$ is the current array.

For those current waveforms obtained by the CW model, if the current array index does not match with the integration time point of the CW model, interpolation is used to find the corresponding current values which are then added to the current array.

3.10 The output of CUREST

Once the total current waveform of the circuit is found, the maximum current peak I_{MAX} and its time of occurrence T_{MAX} are determined by just locating the array element with the largest magnitude. Suppose the dimension of the total current waveform is n. Then,

$$I_{MAX} = max(array[0], array[1], ..., array[n]) = array[M]$$

$$T_{MAX} = time\_step \cdot M$$

where $array$ is the current waveform array and M is the array index of the largest array element.

The average current I_{AVG} can be obtained as a bonus during the searching for the current peak since we can sum up the array values as we scan the array from the beginning to the end. I_{AVG} is calculated as follow:

$$I_{AVG} = \frac{\sum_i current\_array[i]}{stop\_time} \cdot time\_step$$

Furthermore, the input test vector pattern as well as the current triangle associated with each individual logic gate in the circuit are available. At present, CUREST is designed to report the highest five current peaks in a current waveform, the average

current I_{AVG}, and its input test vector pattern. If random input generator is used, only the highest $I_{MAX}[0]$ and its four other peaks, the highest average current, and their associated input test vectors are reported. These limits can be changed by the user if necessary[5].

[5]The parameters that are used to control these limits are defined in the TAMIA data structure file.

Chapter 4

Results

Four test circuits were used to study the speed and accuracy of CUREST. Each circuit is characterized in the table below: The circuit diagrams for the Decoder, 4-bit adder

Circuit Name	Circuit Function	Total Gates	Primary Inputs	Primary Outputs
Decoder	decoder	18	6	8
4-bit adder	adder	40	9	5
ALU 181	ALU	101	14	4
ISCAS85 c880	ALU & control	555	60	26

Table 4.1: Test circuits used for speed and accuracy analysis of CUREST.

and the ALU 181 are presented in [29]. The ISCAS85 c880 is described in [30]. All the circuits are implemented with NAND, NOR and NOT gates only. In particular, the XOR gates in ALU 181 were replaced with NAND gates. Since the present model parameters are applicable to 4 inputs or less, all gates with more than 4 inputs were reconfigured using combinations of gates with less than or equal to 4 inputs. The size and implementation of each cell-type that is used in the above test circuits are

described in Appendix A.

4.1 Speed comparisons between CUREST and HSPICE

A large number of random test vectors were applied to the above four test circuits and the analysis time that is devoted to various steps in CUREST is shown in Table 4.2. CP and DP models were used for approximating, respectively, charging and discharging current waveforms of each logic gate in the test circuit and the time interval between every current array point is set to 0.05 ns. On average it takes about 0.5ms to 1.3ms to

Circuit Name	Number of trials	CPU time				% of average switching activities
		TAMIA pre-processing	CUREST processing	Total time	Process time per gate	
Decoder	1000	2.45s	20.82s	23.27s	1.29ms	87.5%
	4000	1.90s	75.45s	77.35s	1.07ms	87.5%
4-bit adder	1000	1.97s	41.17s	43.14s	1.08ms	84.7%
	4000	1.93s	161.2s	163.1s	1.02ms	84.7%
ALU 181	1000	2.44s	83.18s	85.62s	0.85ms	65.8%
	4000	2.41s	325.0s	327.5s	0.81ms	65.8%
c880	1000	6.50s	300.6s	307.1s	0.55ms	49.8%

Table 4.2: The amount of CPU time required for CUREST to analyse the four test circuits on SUN4/490 with 32 Mb main memory and running with SUN OS 4.103 operating system.

evaluate a logic gate in CUREST. As the number of trials increases, the process time per gate decreases slightly since the time spent at pre-processing becomes less significant as compared to the total process time. Moreover, it happens that the process time per gate is significantly less for large circuits because of relatively less switching activities.

Circuit Name	Number of trial	Total CPU time	Process time per gate
Decoder	1	7.72s	0.43s
4-bit adder	1	21.1s	0.53s
ALU 181	1	73.4s	0.73s
c880	1	652.82s	1.18s

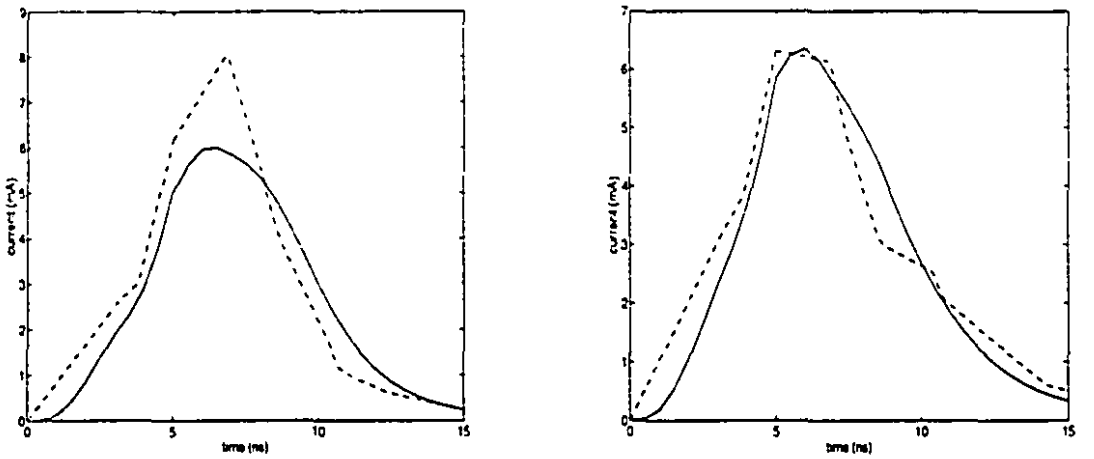
Table 4.3: CPU time required for HSPICE to simulate the four test circuits on SUN4/490 running with SUN OS 4.103 operating system. The maximum integration step equals to 0.5ns.

This is probably due to our neglecting some types of glitches and the propagation of glitches. The accumulation of these two effects may lead to many non-switching gates especially in large circuits where glitches are likely to occur frequently. The HSPICE simulation time required for the the four test circuits is shown in Table 4.3. It takes about 0.5s to 1s to process a gate in HSPICE, and hence the speed-up of CUREST is about three orders of magnitude faster than HSPICE. In HPSICE, 0.5ns time step was used in transient analysis to capture the maximum peak within 10% error. [1]

4.2 Accuracy of CUREST with respect to HSPICE

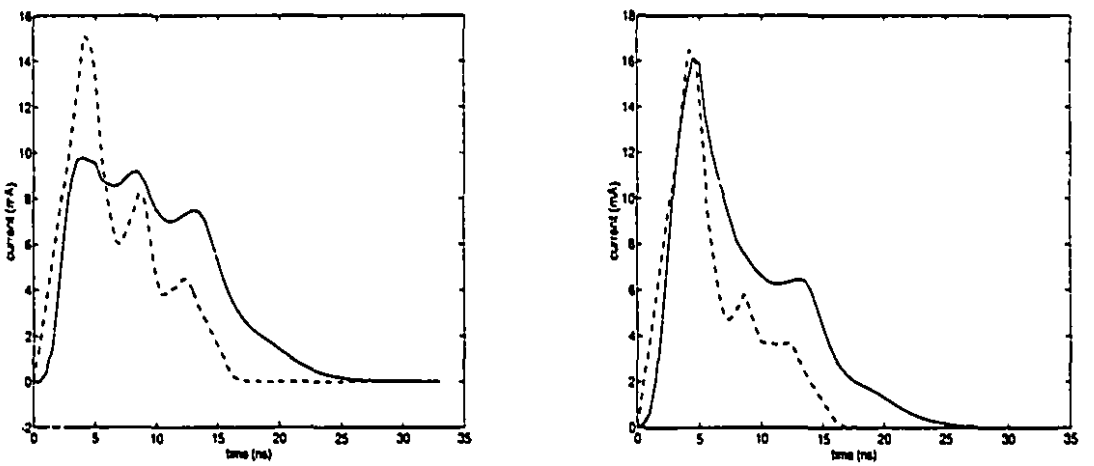
Each of the test circuits was subjected to 10 randomly generated test vectors, and the corresponding current waveforms were plotted with respect to HSPICE. For each

[1]The HSPICE time step is determined as follow. First, a simple logic gate is simulated with a time step of 0.01ns, and its associated current peak $Imax$ is recorded. Then, we gradually increase the time step until 10% error occurs with respect to $Imax$.



(a) (b)

Figure 4.1: The current waveform of 139 Decoder(dash) with respect to HSPICE(solid). (a)Worst case out of 10. (b)Best case out of 10.

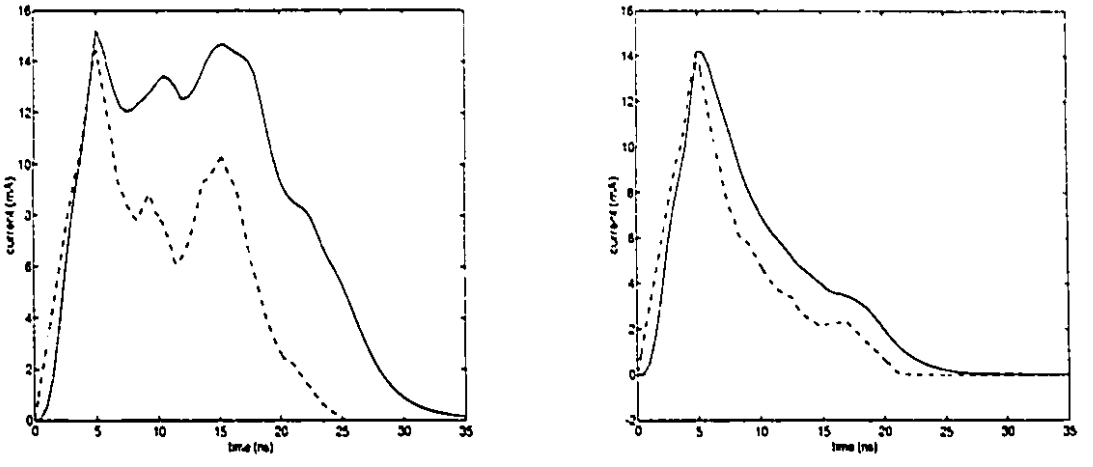


(a) (b)

Figure 4.2: The current waveform of 4-bit adder(dash) with respect to HSPICE(solid). (a)Worst case out of 10. (b)Best case out of 10.

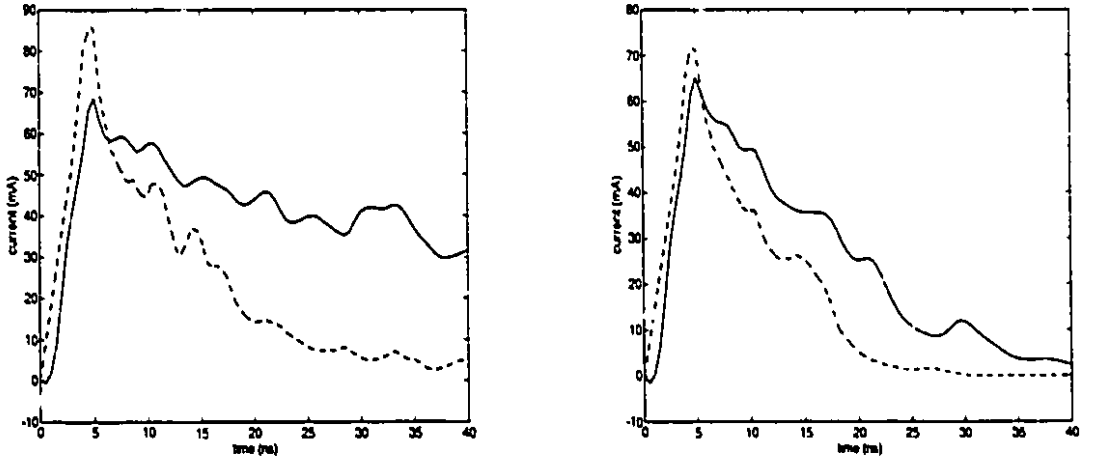
test circuit, the best and the worst current waveforms as compared with HSPICE are shown in Figures 4.1 to 4.4. The HSPICE time step was set to 0.1ns.

From these current waveforms, we note that CUREST is good at determining the



(a) (b)

Figure 4.3: The current waveform of ALU 181(dash) with respect to HSPICE(solid). (a)Worst case out of 10. (b)Best case out of 10.



(a) (b)

Figure 4.4: The current waveform of ISCAS85 benchmark circuit c880(dash) with respect to HSPICE(solid). (a)Worst case out of 10. (b)Best case out of 10.

rise time and peak of the current waveform, but underestimates the fall time. After carrying out a detail analysis of the 4-bit adder, we find that there are two serious problems associated with glitch negligence. First, we note that some glitches do prop-

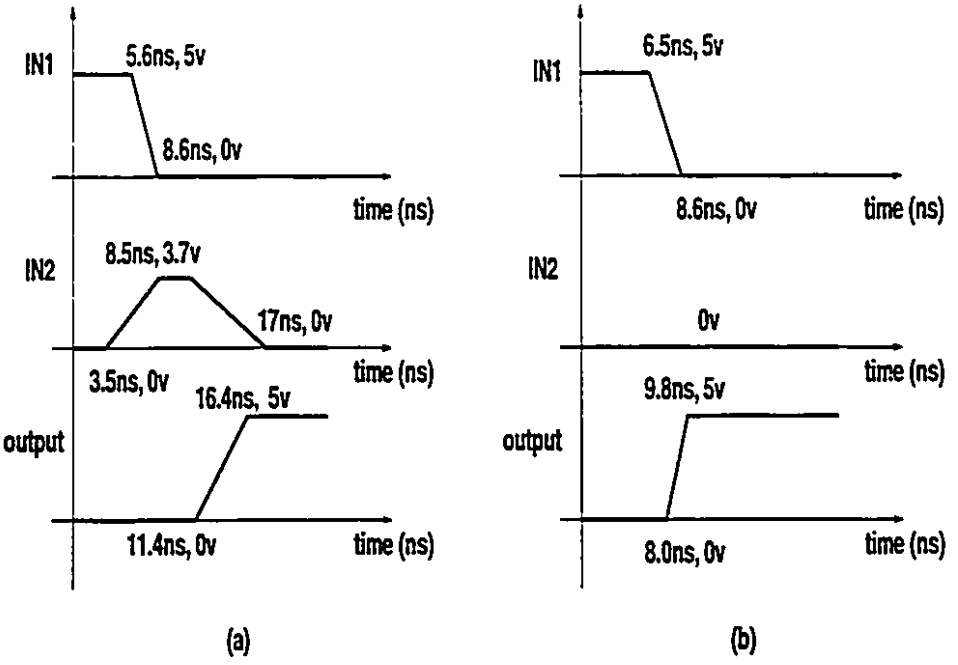


Figure 4.5: The input and output transitions of an internal 2-input NOR gate in the 4-bit adder. IN2 is a glitch. (a) The transition information associated with the inputs and output computed by HSPICE. (b) The transition information associated with the inputs and output computed by CUREST.

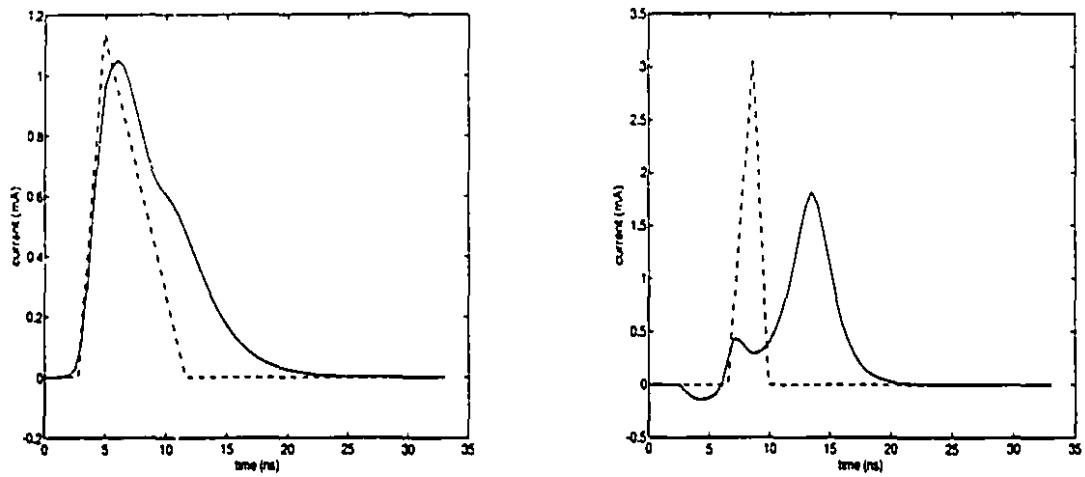
agate. For instance, if we feed a glitch in the shape of a pulse with a peak value of 4.5v to an inverter, the output of the inverter will show a full falling transition followed by a full rising transition. Since we assume glitch does not propagate, the output transition may be computed incorrectly because we treat a glitch as a stable signal with voltage either at V_{DD} or 0. Figure 4.5 is used to illustrate the effect of this error, where the effective input should be the falling edge of the glitch. But, due to our assumption, IN1 is taken as the effective input. As a result, the output transition time as well as the output transition start time is wrong. A similar error would occur in a NAND gate if IN1 is rising and IN2 is a dip. The above error will propagate and accumulate as

output node number	HSPICE mA	CUREST mA
16	0.91	1.04
28	0.53	0.33
31	0.64	0.22
17	0.24	0.26
32	0.51	0.11
39	0.99	1.00
42	0.34	0.00
12	0.33	0.00
46	1.70	0.00
50	1.00	0.00
Total:	7.19	2.96

Table 4.4: Individual and total gate current drawn at 13ns. A gate is identified by its unique output node number.

the signal travels towards the primary output nodes. As a result, the overall transient period is shrunk.

The other serious problem we noted is that the output transition time predicted by the current models is always shorter than the corresponding HSPICE value. For example, in Table 4.4 the current drawn by individual gates at 13ns in Figure 4.2(a) is compared with HSPICE. In particular, gate 12 in Table 4.4 does not contribute any current at 13ns because its upper triangular current bound is less than 13ns as show in Figure 4.6(a). The current waveform of gate 46 in Figure 4.6(b) is shifted towards left because error occurs during the computation of the effect input transition. This error is induced by a glitch as we have already explained above. The error in computing



(a) (b)

Figure 4.6: The current waveforms of individual gates in the 4-bit adder. The triangular waveform computed by CP or DP model is represented by (- - -) line and the smooth current waveform obtained from HSPICE is drawn in solid line. (a) The current waveforms of gate 12. (b) The current waveforms of gate 46.

the current peak is also due to the glitch, because it induces a wrong effective input transition time. Finally, gate 50 and gate 42 are considered inactive because we ignore the propagation of glitches.

Chapter 5

Discussion and conclusions

The current and delay estimation techniques presented in [29] have been automated by utilizing the circuit information provided by the timing analysis tool TAMIA. A C program CUREST is incorporated into TAMIA to carry out the current and delay estimation of every gate in a circuit and obtain the total current waveform of the circuit by summing the individual gate current waveforms. The CUREST input file is similar to HSPICE where the circuit topology as well as the primary input pattern being applied are specified. In CUREST, the user also has the option to apply randomly generated input patterns to a circuit. Furthermore, if primary inputs are less than or equal to 6, the user can elect to carry out an exhaustive analysis on the circuit which is almost impossible to do without the package. The user has the access to the individual current waveform associated with each logic gate, its input/output transition information and its delay. As the size of the circuit and the number of test vectors increase, it is impossible to record all the current waveforms associated with each test vector. As a result, only a couple worst case current waveforms and the worst case average current are saved and output.

To evaluate the speed and accuracy of CUREST, four test circuits of increasing size and complexity were used. The speed of CUREST with respect to HSPICE varies from two order of magnitudes, for small circuit, to three order of magnitudes, for large

circuit. The overhead associated with circuit preprocessing by TAMIA and CUREST is higher for large and complex circuit, but as the number of input test vectors that are applied to the circuit increase, the percentage of total time that is spent as overhead is diminished. Therefore, CUREST is more efficient in computing the current waveforms of large VLSI circuits fed with large number of input patterns.

From the accuracy analysis, we noted that the current models have an excellent performance in predicting the delay, current peaks and the time at which these peaks occur for individual logic gates. But, in some cases, there are significant discrepancies between the total current waveforms obtained from HSPICE and CUREST. The error is especially serious for large circuits. From our investigation, we discovered that these errors are due to inadequate treatment of glitches. This problem is partially solved by approximating the glitch current drawn using the CP model which is acceptable if the input signals are neither too close nor too far apart. Further improvement in glitching current calculation can be achieved by introducing threshold in the overlapping signals and further research is needed to explore the potential of using the existing current models to compute the glitch current.

In conclusion, the current model in [29] is fast and accurate especially in predicting the current peak and its time of occurrence. Its potential application on large VLSI circuit could be definitely impressive once the glitch problem is solve.

Bibliography

[1] F. Najm, R. Burch, P. Ying and I. Hajj, "CREST- a current estimator for CMOS circuits", *IEEE Conf. on Computer-Aided Design*, pp. 204-207, November 1988.

[2] R. Burch, F. Najm, P. Ying and D. Hocevar, "Pattern- independent current estimation for reliability analysis of CMOS circuits", *ACM/IEEE 25th Design Automation Conference*, pp. 294–299, June 12–15, 1988.

[3] F. Najm, R. Burch, P. Ying and I. Hajj, "Probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 9, No. 4, pp. 439–450, April 1990.

[4] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. on Computers*, pp. 668–670, June 1975.

[5] F. Najm, I. N. Hajj and P. Yang, "An extension of probabilistic simulation for reliability analysis of CMOS VLSI circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 10, No. 11, pp. 1372–1381, November 1991.

[6] H. Kriplani, F. Najm, P. Ying and I. Hajj, "Resolving signal correlations for estimating maximum currents in CMOS combinational circuits", *ACM/IEEE 30th Design Automation Conference*, pp. 384–388, 1993.

[7] H. Kriplani, F. Najm, and I. Hajj, "Maximum current estimation in CMOS circuits", *ACM/IEEE 29th Design Automation Conference*, pp. 2–7, 1992.

[8] R. Burch, F. Najm, P. Ying and T. N. Trick, "A Monte Carlo approach for power estimation", *IEEE Trans. on VLSI systems*, Vol. 1, No. 1, pp. 63–71, March 1993.

[9] T. H. Krodel, "PowerPlay - fast dynamic power estimation based on logic simulation", *IEEE Conf. on Computer Design*, pp. 96–100, October 1991.

[10] F. Dresig, Ph. Lanches, O. Rettig and U. G. Baitinger, "Simulation and reduction of CMOS power dissipation at logic level", *European Design and Test Conference*, pp. 341–346, 1993.

[11] S. Devadas, K. Keutzer and J. White, "Estimation of power dissipation in CMOS combinational circuits using Boolean function manipulation", *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 3, pp. 373–383, March 1993.

[12] P. Vanoostende, P. Six, H. J. De Man, "PRITI: Estimation of maximal currents and current derivatives in complex CMOS circuits using activity waveforms", *European Design and Test Conference*, pp. 347–353, 1993.

[13] P. E. Landman and J. M. Rabaey, "Power estimation for high level synthesis", *European Design and Test Conference*, pp. 361–366, 1993.

[14] *HSPICE Users' Manual, H9001*, Meta-Software, Inc., 1990.

[15] R. P. LLopis and H. G. Kerkhoff, "A fast and accurate characterization method for full-CMOS circuits", *IEEE Proceedings EURO-DAC 92*, pp. 410–415, September, 1992.

[16] A. Tyagi, "Hercules: A power analyser for MOS VLSI circuits", *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1987, pp. 530–533.

[17] D. Stark and M. Horowitz, "Analysing CMOS power supply networks using Ariel", *ACM/IEEE 25th Design Automation Conference*, pp. 460–464, June 1988.

[18] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits", *IEEE J. Solid-State Circuits*, Vol. SC-19, No. 4, pp. 468–473, August 1984.

[19] J. E. Hall, Dale E. Hocevar, Ping Yang, and Michael J. McGraw, "SPIDER – a CAD system for modeling VLSI metallization patterns", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 6, pp. 1023–1031, November 1987.

[20] S. Chowdhury and J. S. Barkatullah, "Estimation of maximum currents in MOS IC logic circuits", *IEEE Trans. on Computer-Aided Design*, Vol. 9, No. 6, pp. 642-654, June 1990.

[21] U. Jagau, "SIMCURRENT – an efficient program for the estimation of the current flow of complex CMOS circuits", *IEEE Conf. on Computer-Aided Design*, pp. 396–399, November 1990.

[22] D. A. Haeussler and K. F. Poole, "CURRANT: a current prediction software tool using a switch-level simulator", *Southeaston 89 Proceedings*, Vol. 3, 89, pp. 946–948, 1988.

[23] A. Deng, Y. Shiau, and K. Loh, "Time domain current waveform simulation of CMOS circuits", *Proc. IEEE Conf. on Computer-Aided Design*, pp. 208–211, November 1988.

[24] J. H. Wang, J. T. Fan, and W. S. Feng, "A novel current model for CMOS gates", *Proc. IEEE Symp. on Circuits and Systems*, pp. 2132–2135, May 1992.

[25] F. Rouatbi, B. Haroun and A. J. Al-Khalili, "Power estimation tool for sub-micron CMOS VLSI circuits", *ACM/IEEE Int. Conf. on Computer-Aided Design*, pp. 204–209, Nov. 8–12, 1992.

[26] Maple, version 4.2, Watcom, University of Waterloo.

[27] L. Benini, M. Favalli, P. Olivo and B. Ricco, "A novel approach to cost-effective estimate of power dissipation in CMOS ICs", *European Design and Test Conference*, pp. 354–360, 1993.

[28] G. Ruan, J. Vlach and J. Barby, "Current-limited switch-level timing simulator for MOS logic networks", *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 6, pp. 659–667, 1988.

[29] A. Nabavi-Lishi, "Delay and current evaluation in CMOS circuits", *Ph.D. Thesis*, McGill University, 1993.

[30] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," *IEEE Int. Symp. on Circuits and Systems*, pp. 695–698, June 1985.

[31] M. R. Dagenais, "Timing analysis for MOSFETs an integrated approach",*Ph.D. Thesis*, McGill University, 1987.

Appendix A

The cell-types in the test circuits

The device sizes of the cell-types are selected based on CMC_EDGE CMOS4s standard cells. Modifications are made such that the ratio Wp/Wn of the equivalent inverter is within the range that is specified in the Table 2.1. All the cell-types are implemented based on the NAND, NOR and NOT logics.

```
******INVERTER*****
.subckt not1 in out
mp1 out in vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn1 out in 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
.ends not1
.subckt inv in out
mp1 out in vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn1 out in 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
.ends inv

*******BUFF1********
.subckt buff1 in out
mp1 n1  in  vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn1 n1  in  0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
mp1 out n1 vdd vdd  pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn1 out n1 0   0    nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c1  n1  0 1pf
.ends buff1

******NOR2**************
.subckt nor2 in0 in1 out
mp1 n1  in0 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp2 out in1 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn1 out in0 0   0   nmos l=1.2u w=7.04u ad=14.0p as=21.9p pd=11.0u ps=20.3u
mn2 out in1 0   0   nmos l=1.2u w=7.04u ad=21.9p as=14.0p pd=20.3u ps=11.0u
.ends nor2

**********NOR3**************
.subckt nor3 in0 in1 in2 out
mp5 n1  in0 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp6 n2  in1 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
```

```
mp7 out in2 n2  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn5 out in0 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
mn6 out in1 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
mn7 out in2 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
.ends nor3

**************NOR4**************
.subckt nor4 in1 in2 in3 in4 out
mp1 n1 in1 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp2 n2 in2 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp3 n3 in3 n2  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp4 out in4 n3 vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn1 out in1 0  0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn2 out in2 0  0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn3 out in3 0  0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn4 out in4 0  0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
.ends nor4

*********NAND2**********
.subckt nand2 in0 in1 out
mp1 out in0 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp2 out in1 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn1 out in0  n1 0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn2 n1  in1   0 0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
.ends nand2

**********NAND3*************
.subckt nand3 in0 in1 in2 out
mp1 out in0 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp2 out in1 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp3 out in2 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mn1 out in0  n1 0   nmos l=1.2u w=18.7u ad=14.9p as=14.9p pd=20.3u ps=20.3u
mn2 n1  in1  n2 0   nmos l=1.2u w=18.7u ad=14.9p as=58.4p pd=20.3u ps=43.6u
mn3 n2  in2  0  0   nmos l=1.2u w=18.7u ad=67.3p as=14.9p pd=44.6u ps=20.3u
.ends nand3

*************NAND4**************
.subckt nand4 in1 in2 in3 in4 out
mp1 out in1 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp2 out in2 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp3 out in3 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp4 out in4 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mn1 out in1 n1  0   nmos l=1.2u w=21.6u ad=17.2p as=67.3p pd=23.2u ps=49.4u
mn2 n1  in2 n2  0   nmos l=1.2u w=21.6u ad=17.2p as=17.2p pd=23.2u ps=23.2u
mn3 n2  in3 n3  0   nmos l=1.2u w=21.6u ad=17.2p as=17.2p pd=23.2u ps=23.2u
mn4 n3  in4 0   0   nmos l=1.2u w=21.6u ad=77.7p as=17.2p pd=50.4u ps=23.2u
.ends nand4

*********NAND5: nand3(in0 in1 in2)=n4, nand2(in3 in4)=n5
**********nor2(n4,n5)=n6,not(n6)=out**********
.subckt nand5 in0 in1 in2 in3 in4 out
mp1 n4 in0 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp2 n4 in1 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp3 n4 in2 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mn1 n4 in0  n1 0   nmos l=1.2u w=18.7u ad=14.9p as=14.9p pd=20.3u ps=20.3u
mn2 n1 in1  n2 0   nmos l=1.2u w=18.7u ad=14.9p as=58.4p pd=20.3u ps=43.6u
mn3 n2 in2  0  0   nmos l=1.2u w=18.7u ad=67.3p as=14.9p pd=44.6u ps=20.3u
mp4 n5 in3 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp5 n5 in4 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn4 n5 in3 n11 0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
```

```
mn5 n11 in4  0 0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
mp6 n12 n4 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp7 n6 n5  n12 vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn6 n6 n4  0   0   nmos l=1.2u w=7.04u ad=14.0p as=21.9p pd=11.0u ps=20.3u
mn7 n6 n5  0   0   nmos l=1.2u w=7.04u ad=21.9p as=14.0p pd=20.3u ps=11.0u
mp8 out n6 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn8 out n6 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c4  n4 0 1pf
c5  n5 0 1pf
c6  n6 0 1pf
.ends nand5

***************AND5*****************
.subckt and5 in0 in1 in2 in3 in4 out
mp1 o1 in0 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp2 o1 in1 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp3 o1 in2 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mn1 o1 in0  n1 0   nmos l=1.2u w=18.7u ad=14.9p as=14.9p pd=20.3u ps=20.3u
mn2 n1 in1  n2 0   nmos l=1.2u w=18.7u ad=14.9p as=58.4p pd=20.3u ps=43.6u
mn3 n2 in2  0  0   nmos l=1.2u w=18.7u ad=67.3p as=14.9p pd=44.6u ps=20.3u
mp4 o2 in3 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp5 o2 in4 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn4 o2 in3  n1 0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn5 n1 in4   0 0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
mp6 n1 o1  vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp7 out o2  n1 vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn6 out o1  0  0   nmos l=1.2u w=7.04u ad=14.0p as=21.9p pd=11.0u ps=20.3u
mn7 out o2  0  0   nmos l=1.2u w=7.04u ad=21.9p as=14.0p pd=20.3u ps=11.0u
c1 o1 0 1pf
c2 o2 0 1pf
.ends and5

************AND4****************
.subckt and4 in1 in2 in3 in4 out
mp1 n4 in1 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp2 n4 in2 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp3 n4 in3 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mp4 n4 in4 vdd vdd pmos l=1.2u w=4u ad=10p as=10p pd=13u ps=13u
mn1 n4 in1 n1  0   nmos l=1.2u w=21.6u ad=17.2p as=67.3p pd=23.2u ps=49.4u
mn2 n1 in2 n2  0   nmos l=1.2u w=21.6u ad=17.2p as=17.2p pd=23.2u ps=23.2u
mn3 n2 in3 n3  0   nmos l=1.2u w=21.6u ad=17.2p as=17.2p pd=23.2u ps=23.2u
mn4 n3 in4 0   0   nmos l=1.2u w=21.6u ad=77.7p as=17.2p pd=50.4u ps=23.2u
mp5 out n4 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn5 out n4 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c4  n4 0 1pf
.ends and4

***********AND3*************
.subckt and3 in0 in1 in2 out
mp1 n3 in0 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp2 n3 in1 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mp3 n3 in2 vdd vdd pmos l=1.2u w=5u ad=12.5p as=12.5p pd=15u ps=15u
mn1 n3 in0  n1 0   nmos l=1.2u w=18.7u ad=14.9p as=14.9p pd=20.3u ps=20.3u
mn2 n1 in1  n2 0   nmos l=1.2u w=18.7u ad=14.9p as=58.4p pd=20.3u ps=43.6u
mn3 n2 in2  0  0   nmos l=1.2u w=18.7u ad=67.3p as=14.9p pd=44.6u ps=20.3u
mp4 out n3 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn4 out n3 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c3 n3 0 1pf
.ends and3
```

```
********AND2************
.subckt and2 in1 in2 out
mp1 n2 in1 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp2 n2 in2 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn1 n2 in1  n1 0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn2 n1 in2   0 0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
mp3 out n2 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn3 out n2 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c2 n2 0 1pf
.ends and2

*************OR4***************
.subckt or4 in1 in2 in3 in4 out
mp1 n1 in1 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp2 n2 in2 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp3 n3 in3 n2  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp4 n4 in4 n3  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn1 n4 in1 0   0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn2 n4 in2 0   0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn3 n4 in3 0   0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mn4 n4 in4 0   0   nmos l=1.2u w=2.00u ad=5.0p as=5.0p pd=9.0u   ps=9.0u
mp5 out n4 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn5 out n4 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c4 n4 0 1pf
.ends or4

***********OR3*************
.subckt or3 in0 in1 in2 out
mp5 n1 in0 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp6 n2 in1 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp7 n3 in2 n2  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn5 n3 in0 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
mn6 n3 in1 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
mn7 n3 in2 0   0   nmos l=1.2u w=3.04u ad=7.7p  as=7.7p  pd=11.08u ps=11.08u
mp8 out n3 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn8 out n3 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c3 n3 0 1pf
.ends or3

********OR2************
.subckt or2 in0 in1 out
mp1 n1 in0 vdd vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mp2 n2 in1 n1  vdd pmos l=1.2u w=60.8u ad=152p as=152p pd=126.6u ps=126.6u
mn1 n2 in0 0   0   nmos l=1.2u w=7.04u ad=14.0p as=21.9p pd=11.0u ps=20.3u
mn2 n2 in1 0   0   nmos l=1.2u w=7.04u ad=21.9p as=14.0p pd=20.3u ps=11.0u
mp8 out n2 vdd vdd pmos l=1.2u w=10.72u ad=27.4p as=27.4p pd=26.3u ps=26.3u
mn8 out n2 0   0   nmos l=1.2u w=5.36u  ad=16.7p as=19.3p pd=16.9u ps=17.9u
c2 n2 0 1pf
.ends or2

********XOR2: nand(nand(~in1,in2), nand(in1,~in2))***********
.subckt xor in1 in2 not1 not2 out
****nand(not1,in2)=n3
mp3 n3 not1 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp4 n3 in2  vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn3 n3 not1 n6  0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn4 n6 in2  0   0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
****nand(in1,not2)=n4
mp5 n4 in1  vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp6 n4 not2 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
```

```
mn5 n4 in1  n7  0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn6 n7 not2 0   0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
****nand(n3,n4)=out
mp1 out n3 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mp2 out n4 vdd vdd pmos l=1.2u w=10.0u ad=25p as=25p pd=25u ps=25u
mn1 out n3 n8  0   nmos l=1.2u w=19.0u ad=15.2p as=59.4p pd=20.6u ps=44.3u
mn2 n8  n4 0   0   nmos l=1.2u w=19.4u ad=68.5p as=15.2p pd=45.2u ps=20.6u
c3  n3 0 1pf
c4  n4 0 1pf
.ends xor
```

Appendix B

The input format for CUREST

The input format for CUREST is exactly the same as that for TAMIA except there is a slight modification to include the primary input patterns. Besides, a few options that are exclusive to CUREST are added. A decoder circuit is used for illustration.

```
****comment line  starts with ''*''****
.include cell_mf.spi
.interval int1

****circuit topolgy is described below****
x1 4 20  inv
x2 3 19  inv
x3 6 18  inv
x4 2 17  inv
x5 1 16  inv
x6 5 15  inv
x7 20 21 inv
x8 19 22 inv
x9 17 23 inv
x10 16 24 inv
x11 21 22 18 14  nand3
x12 21 19 18 13  nand3
x13 22 20 18 12  nand3
x14 19 20 18 11  nand3
x15 23 24 15 10  nand3
x16 23 16 15 9   nand3
x17 24 17 15 8   nand3
x18 16 17 15 7   nand3

****primary inputs used in TAMIA****
vin1 1 0 symbolic(int1 symbolic(fast_high(0 5 4ns 0)
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))
vin2 2 0 symbolic(int1 symbolic(fast_high(0 5 3ns 0)
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))
vin3 3 0 symbolic(int1 symbolic(fast_high(0 0 5ns 5)
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))
vin4 4 0 symbolic(int1 symbolic(fast_high(0 0 7ns 5)
```

```
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))
vin5 5 0 symbolic(int1 symbolic(fast_high(6n 5 8ns 0)
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))
vin6 6 0 symbolic(int1 symbolic(fast_high(6n 5 9ns 0)
+ slow_high(0 0 0 0) fast_low(0 0 0  0) slow_low(0 0 0 0)))

****external loading capacitances****
cl7 7 0 1pf
cl8 8 0 1pf
cl9 9 0 1pf
cl10 10 0 1pf
cl11 11 0 1pf
cl12 12 0 1pf
cl13 13 0 1pf
cl14 14 0 1pf
c15  15 0 1pf
c16  16 0 1pf
c17  17 0 1pf
c18  18 0 1pf
c19  19 0 1pf
c20  20 0 1pf
c21  21 0 1pf
c22  22 0 1pf
c23  23 0 1pf
c24  24 0 1pf

****primary input patterns used in CUREST:
     input_name, input_node_number, start_time, transition_time,
     transition_type****
i1 1 0.0 5.0e-9 4
i2 2 0.0 5.0e-9 4
i3 3 0.0 5.0e-9 4
i4 4 0.0 5.0e-9 4
i5 5 0.0 5.0e-9 4
i6 6 0.0 5.0e-9 4

****primary output node numbers****
.output 7 8 9 10 11 12 13 14
.display 7 8 9 10 11 12 13 14

****Options that are applied to CUREST****
*def_step: determine the spacing between current time points
*stop_time: determine the stop time of the current waveform
*hazard: 1 included glitch current
*          0 ignored glitch current
*integration: 1  CW model is selected
*             0  CP model is selected
*specify_input: 0 randomly generated inputs are needed
*                1 inputs are specified by the user
*random_type:
* 0 transition start time, transition time and transition type are specified
* 1 randomly generate transition start time
* 2 randomly generate transition time
* 3 randomly generate transition start time and transition time
* 4 randomly generate transition type
* 5 randomly generate transition start time and transition type
* 6 randomly generate transition time and transition type
* 7 randomly generate transition start time, transition time
*   and transition type
```

```
*random_set: determine the number of randomly generated input vectors
*pr_rise: probability of rising input
*pr_fall: probability of falling input
*pr_high: probability of stable input staying at Vdd
*pr_low:  probability of stable input staying at Gnd
*time_t0: start time of primary inputs
*tran_time: transition time of primary inputs
*tran_type: transition type of primary inputs
*exhaustive: 0 exhaustive analysis is selected
*            1 exhaustive analysis is not selected

.options min_step=1e-12 max_step=1e-6 def_step=5e-11 min_dv=0.02
+  max_dv=0.4 ac_error=0.3 max_high=5.0 min_high=4.9
+  max_low=0.1 high_treshold=4.0 low_treshold=1.0
+  target_error=0.2 ac_iter=50
+  stop_time=15.0e-9 hazard=1 integration=0 specify_input=0
+  random_type=4 random_set=4000 pr_rise=0.5 pr_fall=0.5 pr_high=0.0
+  pr_low=0.0 time_t0=0.0e-9 tran_time=5.0e-9 tran_type=4 exhaustive=0
* Min_step used to be 1e-11
* Max step used to be 1e-6
* Min_dv used to be 0.05
* Target_error used to be 0.2
.end
```