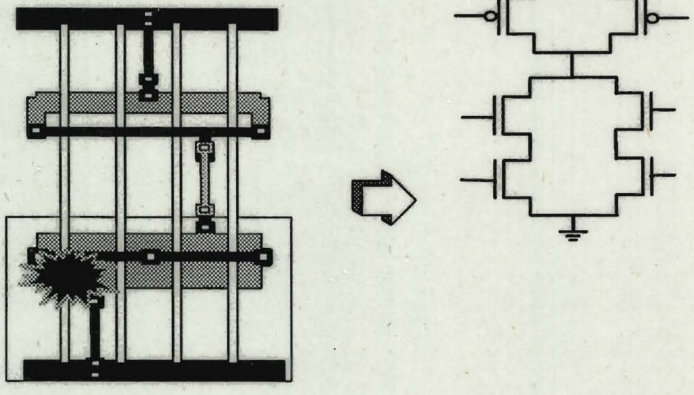
Modeling and simulation of defect induced faults in CMOS IC's

Modeling and Simulation of Defect Induced Faults in CMOS IC's

$$F = \overline{a \cdot b + c \cdot d}$$
$$F_{bri} = F + \overline{a \cdot b} \cdot c \cdot d$$



Chennian Di

Modeling and Simulation of Defect Induced Faults in CMOS IC's

Modeling and Simulation of Defect Induced Faults in CMOS IC's

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op vrijdag 31 maart 1995 om 16.00 uur

door

Chennian Di

geboren te Xi'an, P.R. China

Dit proefschrift is goedgekeurd door de promotoren

prof. Dr. –Ing. J.A.G. Jess
en
prof. ir. M.T.M. Segers

Druk: Dissertatiedrukkerij Wibro, Helmond

CIP–DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Di, Chennian

Modeling and simulation of defect induced faults in CMOS IC's / Chennian Di. – Eindhoven: Eindhoven University of Technology. – Fig., tab.
Thesis Technische Universiteit Eindhoven. With ref.
ISBN 90–386–0040–2
NUGI 832
Subject headings: integrated circuits; CAD / integrated circuit testing.

Acknowledgements

I would like to thank all who helped me going through this valuable and memorable period. Particularly I would like to thank Professor Jochen Jess who took all the trouble to grant me the chance of conducting and finishing this thesis work. In his group, I enjoyed the true academic freedom and yet got all the supports. His vision, enthusiasm and encouragement have been always a stimulus to me.

I would like to thank Geertleon Janssen for letting me use his BDD package and helping me come up with the algorithm documented in the Appendix of this thesis.

From the bottom of my heart, I am very grateful to my wet–grandparents and my parents. It is the early life with them that gives me the strength to carry on in the last years without fear.

Summary

The quality of testing Integrated Circuits (IC) highly depends on the manufacturing process and on a specific design. This is especially true for CMOS digital IC's since the generally used single stuck–at fault model cannot fully describe the behavior of defects induced during the manufacturing process. This thesis outlines a technology–driven testing flow to study the behavior of defect–induced faults with the ultimate goal of generating a reliable and economic test for CMOS digital IC's.

The thesis starts with the introduction of a layout–circuit fault extractor system to study what are the possible occurring faults for a design. This system takes the circuit layout, defect mechanisms and statistics of a process line as inputs and computes all the possible occurring faults and their probabilities. The central topic of the thesis is the modeling and simulation of the two major faults: bridging and open faults.

The main issue addressed in this thesis is how the behavior of each defect–induced bridge or open fault can be accurately modeled and yet a fast fault simulation procedure can be obtained for a large circuit. This thesis employs a simple "divide and conquer" approach. Following this approach, the whole task is completed in two steps. In the first step, the circuit extracted from the layout of a design is further abstracted at logic–level and simultaneously each defect–induced fault is modeled at logic–level as Boolean expressions. Such modeling is realized either by approximate computations or circuit–level simulations. In the second step, the fault simulation is conducted at logic–level just by manipulating these modeled Boolean expressions. Consequently, both accuracy and efficiency can be obtained. The thesis details several systems with a different degree of accuracy and efficiency.

The first system uses an approximate transistor model to model each bridge fault. This results in a very fast modeling and simulation system but with the disadvantage that not every undefined state caused by a bridge can be resolved. With the introduction of two new concepts, the "generic–bridge–table" and the "generic–cell–table", the second system models each bridging fault with a circuit–level simulator. This results in a

reasonable modeling and simulation speed but with the advantage that almost every undefined state caused by a bridging fault can be resolved. The system developed for open faults can model both the hazard and charge–sharing effects of each open fault and yet can perform the fault simulation for opens almost as fast as for single stuck–at faults.

All the systems are verified by experiments with well established benchmark circuits. The results are encouraging.

Contents

1

General Introduction

1.1 Background

Though today's technology, being capable of integrating a few million transistors on a single chip, provides tremendous functionalities with high performance, it provides very poor accessibility to the external world because of the limited pin count. It is very hard for test engineers to check the correctness of a manufactured Integrated Circuit (IC). Testing of IC's is becoming a bottleneck for the whole design and manufacturing cycle. This problem is even more protruding for the dominating CMOS technologies. This is because the manufacturing defects may cause many more complex faults than the practically used single stuck–at fault models at logic–level [24,53]. For example, even with very careful process control and the elimination of all possible causes, a random spot defect, as one of the major manufacturing defects, may still occur in the final manufactured IC's. For a CMOS circuit, various faults that cannot be described by single stuck–at faults may result. Figure 1.1 illustrates a piece of layout of two CMOS cells in a design and their corresponding transistor schematic. If spot defects (d_1 through d_6) occur in the positions as shown in the layout, some network nodes are erroneously connected or a node is broken into two parts as indicated in the schematic. The first type of fault is called a **bridging fault** and the second type is called an **open fault**. In general, any fault that is caused by a spot defect is referred to as a **defect–induced fault.** Clearly, except d_6 which shows the direct stuck–at 0 of the output of one cell, rest of the defects cannot be mapped into the single stuck–at faults that are assumed at the inputs and the outputs of the cells. They have more complex behavior than stuck–at faults. In general, a defect–induced fault cannot be always mapped into a stuck–at fault. The missing link between the heuristic fault model assumed at logic level and defect–induced faults was first made public in the late 70s

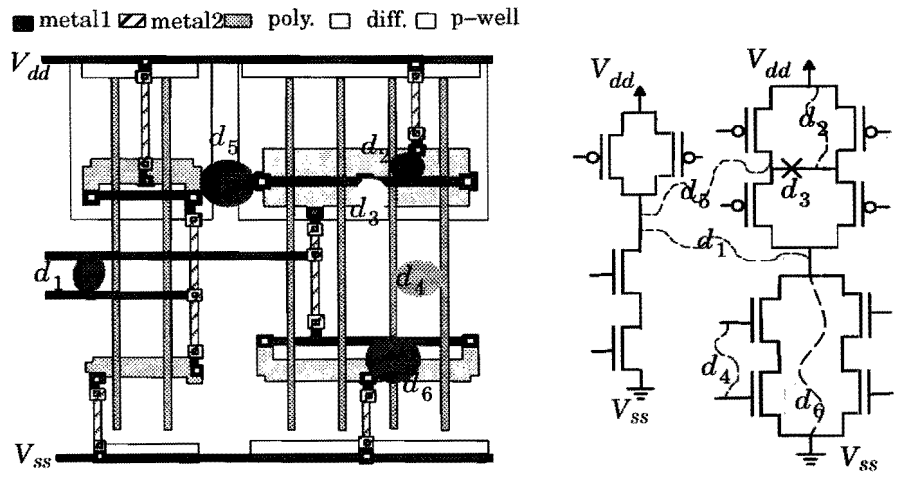


Figure 1.1 Illustration of defects and their induced faults.
(d_1, d_2, d_5, d_6: extra metal1 d_4: extra poly. d_3: missing metal1)

[24,53]. However little attention has been paid until the IC's manufacturing feature size was sufficiently scaled down and the demand of high quality and high performance IC's was increased.

As for testing, the only way to find those defects would be by microscopic inspection. However this procedure is much too expensive for testing mass production. From the example shown in figure1.1, it is obvious that the actual occurrence of a fault during manufacturing depends on the actual defects, the technology, the fabrication process and the actual layout of the circuit. To study the impact of defects on a design and on the existing testing methods, it is essential to know:

1) What are the defects and their characteristics ? How can these data be obtained from a specific process line?

2) With the available defect information, then for a specific design, what kind of circuit faults can possibly occur and what are their probabilities?

After the above issues to be settled, then the next questions are:

3) How do these defect–induced faults manifest themselves in a design? What is the electrical behavior? How serious is it that the tests targeted at single stuck–at faults cannot detect these defect–induced faults?

4) If the single stuck–at fault model is not adequate, what kind of fault models should be used and how can test patterns be generated for them?

To answer these questions, a bottom–up test approach has to be established. This thesis refers to such an approach as a technology–driven test

methodology. The thesis intents to identify and formulate the problem and further investigate possible solutions.

1.2 Schematic of a "technology–driven" test philosophy

1.2.1 Inductive fault analysis (IFA)

As mentioned in section 1.1, defect mechanisms in IC's are too complex to be captured by the standard "stuck–at" fault model. Consequently the reliability of the test coverage prediction has to be questioned. In order to obtain more reliable test coverage we obviously have to study the defect mechanisms within the fabrication technology and the way that they translate into faulty circuit behavior. Such a method is the so–called "inductive fault analysis". Figure 1.2 is supposed to illustrate this method and the way it leads to an adequate characterizations of circuit faults, to reliable fault coverage computations and eventually to improved test vector sets. The following sections will elaborate each step briefly.

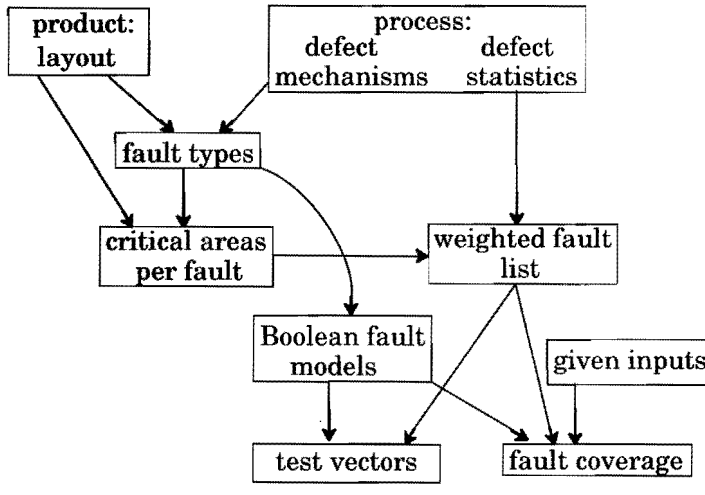


Figure 1.2 A "technology–driven" test flow.

1.2.2 Input to IFA

The analysis starts with two separate sets of input data, namely:

- the product in terms of its chip layout, which is in essence a set of rectangles and their coordinates;

– data characterizing the fabrication process.

The latter needs some explanation. The fabrication process has a long sequence of lithographical steps interleaved by physical/chemical steps. Many things can go wrong. However for IFA it is assumed that any systematic or repetitive defect patterns are eliminated while the process goes through the set up phase or is in maintenance. We are only interested in those defects showing up during the stable processing of valid products. Then the defects are random in the first place. In the second place, they amount to local disturbances in the form of extra or missing spots of material. Those defects are called "random spot defects".

In the sequel, for IFA, a fabrication process can be characterized by

- the layers of the chip structure characterizing the defect mechanisms.
- the geometrical shape of the defect.
- the stochastic size distribution of the geometrical shape parameters (such as diameter or edge length).
- the stochastic distribution of the frequency of the occurrence of the defect.

Depending on the kind of fabrication process it may be arbitrarily difficult to characterize it in the above way. IFA therefore often makes simplifying assumptions [21,34]. For instance it is assumed that defects appears only in a single layer [21], which is obviously not true in some cases. Some other example: defects may be assumed to be of circular or square shape (the latter assumption allows for a particularly effective computation)[21,25]. Size distributions come in all kinds of forms [20]. The only fact that seems to be reasonably safe is that very small and very large defects are very rare. As to the frequency of occurrence the assumption of an equal spread of occurrence of defects seems in general to lead to pessimistic analysis. Therefore most defect frequency models account for the clustering of defects in certain locations of some wafer [48].

The more thoroughly the fabrication process is characterized the higher the reliability of IFA.

1.2.3 Relation between defects, faults and critical areas

Assuming that most defects can be characterized by random spots of extra or missing material the associated circuit faults most likely appear as net bridges or opens in the interconnect structure of the chip under study. A way to characterize the set of faults actually occurring as, for example, the consequence of a spot of metal in a metal layer, can be pictured as follows: we choose a spot of random size d and let it travel over all the locations in the metal layers. If the spot is centered in a certain location such that it

short–circuits two nets say n_1 and n_2, together causing a bridge, we attache a name to this circuit fault and we find all points where the spot causes the same bridge. The set of all these points establishes the critical areas for this particular fault, where the size d is a parameter. Obviously the critical area is a nondecreasing function of d. Combining the critical area analysis with the statistical information about defect density and size yields a probability measure for the respective circuit fault to occur[16,21].

The computational work involved with doing this for all possible faults is considerable. The results of using the system described in [57] indicate about the cost involved and they are hopeful.

1.2.4 Adequate fault modeling for test vector generation

Of course bridges and opens are physical characterizations of the effect of fabrication defects. It would be very expensive to find those defects by microscopic inspection. Therefore for economic reasons testing at the end of mass production must happen by automatic electrical measurements using programmable instrumentation. Moreover the most economic way is to apply digital test signals at the signal ports and observe the output signals. There is a whole industry supporting instrumentation optimized for this purpose. It is important for industry to be able to stay using this equipment because it represents usually a large investment loan if one considers the total investment into the line. This leads to the central topic of this thesis, namely the characterization of defect induced circuit faults by Boolean expressions. The thesis discusses a number of ways to capture the fault behavior of bridges and opens by Boolean expressions. In addition it presents results on the computational work involved for finding the logic models. Furthermore efficient fault simulation techniques of using those logic models are developed and eventually the reliable test coverage can be predicted. The results are encouraging in terms of accuracy and efficiency.

One issue remains unsolved in this thesis, namely the question how to generate economic test vector sets for the new models. Of course having an efficient fault simulation technique may be considered as a partial solution to the problem. Results of further study can be expected in the future.

1.3 Outline of the thesis

In general, it is not an easy task to capture the Boolean behavior of defect–induced faults accurately such that fast fault simulations and improved test vector sets can be obtained. This is especially true for non–regular CMOS logic circuits. This thesis focuses on the accurate

modeling and efficient simulation of defect–induced faults for static CMOS combinational circuits. The thesis is organized as follows.

In chapter 2, after the defects and circuit faults are described in detail, the well developed concept of "critical area" together with a system to extract critical areas is introduced. To obtain the possible faults for a design, a probabilistic model of combining critical areas with defect statistics is developed. The results of extracting the faults by this system for a set of benchmark circuits are presented. The results are analyzed and a suggestion for fault modeling is given.

Chapter 3 formulates the problem for one type of the important faults, the bridging faults. An approximate modeling and simulation method is developed based on the results of some experimental study. The developed method tries to improve the modeling accuracy as much as possible while the modeling and simulation efficiency can be maintained. The method uses a simple and yet explicit transistor model to analyze each bridging fault. As the result of the analysis, each bridging fault is modeled at the logic level in terms of Boolean functions, called faulty Boolean functions. The fault simulation can be performed at logic level by just using the faulty Boolean functions. This method is effective for many bridges and out–performs switch–level approaches.

In chapter 4, the problem of bridging faults is further studied in order to achieve the circuit–level accuracy without sacrificing the fault simulation efficiency. With the exploitation of some design features, two new concepts are introduced in this chapter. The first one, the "generic–bridge–table", is applied to characterize the behavior of each bridging fault. The second one, the "generic–cell–table", is used to characterize how each cell interprets an input. These two sets of tables are derived dynamically for a design by SPICE simulations. It is demonstrated that they can be easily used by any logic fault simulator to determine whether a bridge is detected. Thus both circuit–level accuracy and logic–level simulation efficiency are obtained.

In chapter 5, a method of modeling and simulating open faults is proposed. This method follows the same philosophy as for bridging faults. For any open fault, this method performs a local analysis by taking both the hazard and charge–sharing effects of the open into account. Afterwards, the open is modeled in terms of a detecting condition at logic–level. Then, the fault simulation can be performed at logic level by just manipulating the detecting conditions. This is efficient and also accurate.

Chapter 6 reviews the whole thesis and evaluates the methods developed in this thesis. At the end, possible future work is suggested.

2 Defects and CMOS Circuit Faults

2.1 Spot defects and critical areas

In a mature manufacturing process the essential causes of malfunctions of IC's are the so–called random spot defects. Those defects are local contaminations of the layer structures establishing electrical elements. They are mainly induced by dust particles during photolithographic processing. Typical examples are spots of metal or polycrystalline silicon and pin holes in the silicon–oxide insulation layers. Figure 2.1 shows two photos taken from a process line indicating the existence of such spot defects.

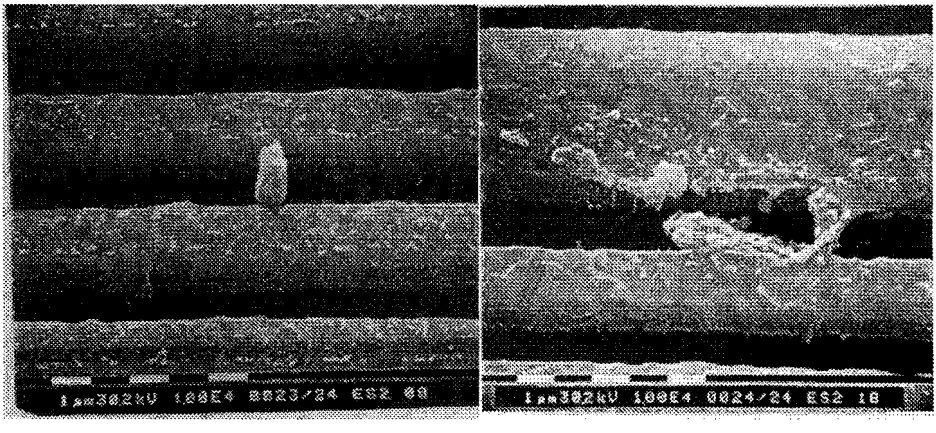


Figure 2.1 Examples of spot defects.

Spot defects can be conceptualized as missing or extra material with a random size. For a specific process line, usually spot defects can be characterized by a defect size distribution and a defect density, namely the probability of occurrence of each different defect size and the number of

defects per unit area. Such information can be captured by process monitors [11,35]. These process monitors are usually regularly structured patterns implemented in some layers. They can be placed on the wafer between dies. After processing, the defect data, namely the defect size distribution and defect density, can be obtained by electrically measuring the monitors.

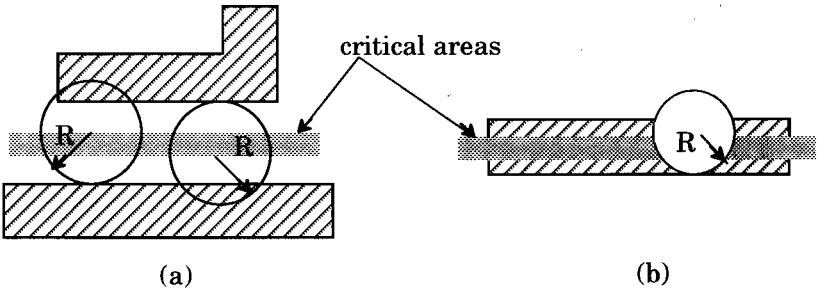


Figure 2.2 Illustration of critical areas.

The combination of layers of an IC, named "structure", corresponds to certain electrical elements, like a transistor or a via. If a defect is present on some layer of a structure it may cause a fault affecting the entire structure. Typically two or more conducting patterns are unintentionally connected or some conducting patterns are broken. At the circuit level, the defect may cause bridging faults among network nodes or the splitting of some network nodes. One way of studying the impact of defects on a layout design is by means of extracting the critical areas [48]. Roughly speaking, the set of center points of all defects causing a fault of a defined type relative to some layout structure defines the critical area for this layout structure. Figure 2.2(a) illustrates critical areas bridging two patterns for a specific defect size. Figure 2.2(b) shows the critical areas of a defect breaking a pattern. Clearly the critical area is a function of the defect size. It is possible that there are as many critical areas for any structure as there are defect mechanisms affecting each layer. The initial application of critical areas is for yield predictions [25,48,55]. Among various systems developed to extract critical areas, one of the efficient methods [25] uses a geometrical computation. Figure 2.3 illustrates the extraction procedure. It first scans the layout to identify the potential parts of the layout where a defect may induce a fault (illustrated in figure 2.3(a)). Then the potential parts are expanded or shrunk for a given defect size (figure 2.3(b)). Finally the contour of a set of rectangular regions is computed and the union of all critical areas is obtained (figure 2.3(c)). The complete concept and the detailed algorithms are described in [25].

The impact of defects on fault modeling and simulation has also been noticed. Unfortunately, for a long time, there were no accurate and efficient tools to

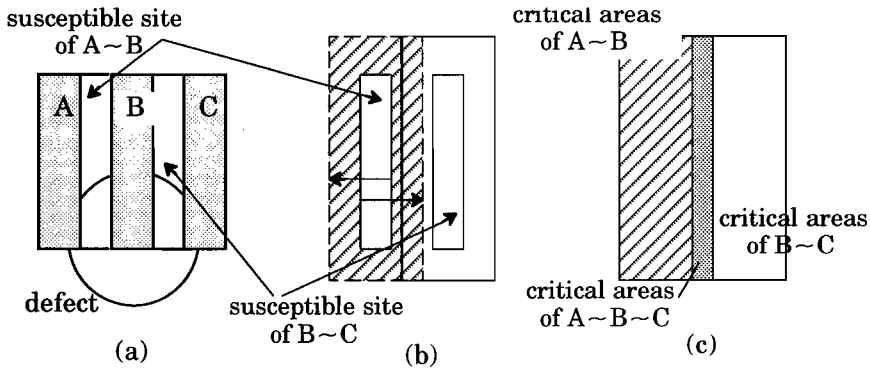


Figure 2.3 Illustration of critical areas extraction. (a) susceptible site extraction. (b) expansion of susceptible site. (c) critical areas computation.

model and simulate the large amount of defect–induced faults for a relatively large circuit. Instead, most people intend to use the single transistor stuck–on(off) as a supplement fault model to the stuck–at fault model. The arbitrary and heuristic nature of this model caused it to find hardly any applications. Only a few years ago, attention was drawn towards the fact that the occurrence of a circuit fault largely depends on the defect conditions and the circuit layout [21,34]. Such occurrence is technology and design dependent. The accurate and realistic faults can only be obtained from the physical layout of a design by detailed analysis. Under defect conditions not only the possible faults but also their probability of occurrence should be obtained. This procedure as described in chapter 1 is known as IFA [21,55]. Since the appearance of paper [21] many systems capable of modeling defects as node bridging and line open faults have been developed [25,55]. The previously mentioned system of extracting critical areas can be applied to perform inductive fault analysis as well. That is, instead of computing total

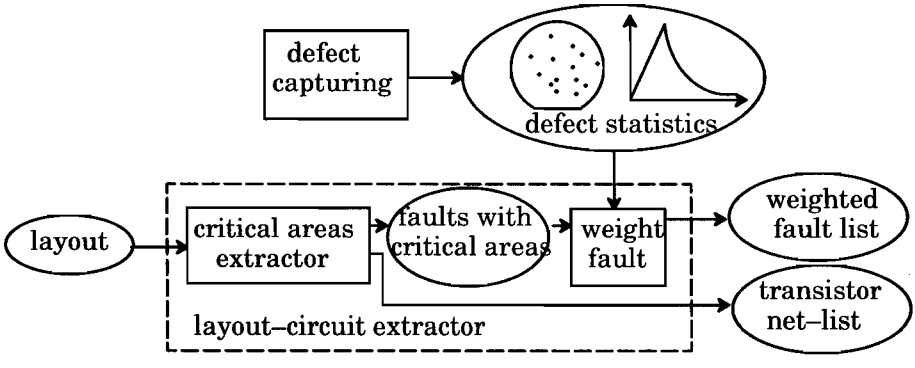


Figure 2.4 The overview of the analysis system.

critical area, the intersection of critical areas related to different faults is also computed as shown in figure 2.3(c). Very recently the improvement of the method and its application to inductive fault analysis is described in [57]. The essential difference of these two approaches [25,57] from others is that it first computes the critical areas for each particular fault. Then, the final probability of the occurrence of a fault can be obtained by taking into account the defect statistics. Thus obtaining the probability of a fault is independent of the critical area extraction. With such a modular feature, further analysis, for example, to verify a design for different defect statistics, can be done without repeating the whole extraction procedure. Consequently, this strategy is much faster than the full simulation method employed in [21,55]. The whole system of performing the inductive fault analysis is illustrated in figure 2.4.

The following section presents how the probability of the occurrence of a fault is derived from the extracted critical areas by combining defect statistics.

2.2 Likelihood of the occurrence of a fault

For every defect mechanism, the critical areas can be extracted as it is illustrated in [25]. Usually, more than one different defect mechanism can induce the same fault, or vice versa only one defect mechanism may induce more than one fault. The final probability result should take these situations into account.

First some notation is introduced. Let $M = \{m_1, m_2, \ldots, m_I\}$ be the set describing a total of I possible independent defect mechanisms, such as "extra metal" and "missing polysilicon". Assume that the defect mechanisms are mutually stochastically independent processes as in [48]. As defects from every defect mechanism occur with a random size and the number of defects is random as well, let $D_m(x)$ represent the defect size distribution and γ_m the defect density, where x denotes the defect size, confined from min to max, and $m \in M$, the defect mechanism. Let $F = \{f_1, f_2, \ldots, f_J\}$ be the set describing a total of J possible distinct fault types, such as, bridge, line open and transistor stuck–on. Let $N = \{n_1, n_2, \ldots, n_K\}$ be the set defined by the K electrical nodes of a design. Since one defect mechanism can induce more than one fault affecting one or more nodes, a fault can be represented as a pair $< f, n >$ where $f \subseteq F$ and $n \subseteq N$.

The sensitivity of a particular fault $< f, n >$ due to a defect of size x from a defect mechanism m, or the probability that such a fault occurs, is related to its critical area by

$$A_m^{<f,n>}(x) = S_m^{<f,n>}(x) \; A_{layout} \tag{2.1}$$

or
$$S_m^{<f,n>}(x) = \frac{A_m^{<f,n>}(x)}{A_{layout}} \tag{2.2}$$

where $A_m^{<f,n>}$ is the critical area, and $S_m^{<f,n>}$ the sensitivity, due to a defect mechanism m, both as a function of a defect size x. A_{layout} is the total layout area.

This sensitivity (eq.(2.2)) is in fact a measure of the design's vulnerability to different defect mechanisms and to each different defect size. However, in a manufacturing environment the probability of occurrence of each different defect size is not the same. Therefore, the average probability of occurrence of a fault $< f, n >$ due to a defect from a defect mechanism m is computed as

$$\Phi_m^{<f,n>} = \int_{\min}^{\max} S_m^{<f,n>}(x)\ D_m(x)dx \tag{2.3}$$

where $D_m(x)$ is the defect size distribution that can be obtained from a manufacturing line. Eq.(2.3) represents the likelihood of a fault for all defect sizes induced by one defect mechanism.

Since more than one defect from a defect mechanism m may occur, we obtain the average number of times that $< f, n >$ occurs as

$$\lambda_m^{<f,n>} = \gamma_m\ A_{layout}\ \Phi_m^{<f,n>} \tag{2.4}$$

As mentioned before, more than one defect mechanism can induce the same fault. Therefore, the probability of each fault $< f, n >$ due to defects from all possible defect mechanisms is expressed as

$$W^{<f,n>} = \sum_{m\ \in\ M} \lambda_m^{<f,n>} \tag{2.5}$$

Since the result $W^{<f,n>}$ is not normalized, in the sequel it is referred to as the relative *weight* of the fault. This weight represents the likelihood of occurrence of the fault $< f, n >$ due to all possible defects. After substitution of eq.(2.2), eq.(2.3) and eq.(2.4) into eq.(2.5), the final weight is obtained as

$$W^{<f,n>} = \sum_{m\ \in\ M} \gamma_m \int_{\min}^{\max} A_m^{<f,n>}(x)\ D_m(x)dx \tag{2.6}$$

It is straightforward to obtain the relative weight for each fault. First the critical areas of all the possible defect mechanisms that cause the same fault are grouped together, i.e. if defects of extra metal and extra contact both cause the same bridge fault, then the critical areas for each defect size of both defect mechanisms will be put in one group. This process is repeated for every

different mechanism of each fault $< f, n >$. After such grouping, the total number of faults for every type of fault is reduced to the total number of distinct faults. Then the weight for each fault is computed in the same procedure as executed by deriving eq.(2.2) to eq.(2.5).

2.3 Fault extraction for CMOS circuits

2.3.1 Circuit and fault classification

A full CMOS combinational circuit discussed in this thesis can be viewed as an interconnection of CMOS cells. A CMOS **cell** has a network of serial–parallel PMOS transistors as pull–up (**P**) and, its dual part, the pull–down (**N**) part. For ease of analysis, the network node which is the drain, source or gate of a transistor is classified in terms of the following three types:

1) **input node** : all the primary inputs, power supply $V_{dd}(V_+)$ and ground $V_{ss}(V_-)$;
2) **output node** : the outputs of all the cells including primary outputs and intermediate outputs;
3) **internal node** : all the nodes inside cells (exclusive input and output nodes).

A set of ISCAS85 benchmark circuits [7] is used for analysis. They are implemented in a standard cell design approach with double metal and a single polysilicon for a 2μ CMOS technology (source: Microelectronics Center of North Carolina (MCNC)). The cell library consists of both simple (such as NAND and NOR) and complex (such as And–Or–Invert (AOI) and Or–And–Invert (OAI)) cells.

The bridging and open faults, as two major types of faults, are further classified as:

- **single bridge**: a bridge caused by a defect connects two distinct nodes.
- **multiple bridge**: a bridge caused by a defect connects more than two distinct nodes.
- **single open**: one node is disconnected from the network due to a defect.
- **multiple open**: the network is split into more than two connected subnetworks due to a defect.

Concerning the type of the network node, the single bridging faults can be further classified as follows.

- **input to input bridge**: a bridge caused by a defect connects two input nodes (e.g. d_4 in figure 1.1).

- **output to output bridge**: a bridge caused by a defect connects two output nodes(e.g. d_1 in figure 1.1).
- **internal to internal bridge**: a bridge caused by a defect connects two internal nodes. It may occur either inside one cell or between two different cells. For ease of analysis, the bridges between $V_{dd}(V_{ss})$ to internal nodes are classified to belong to this type as well(e.g. d_2 in figure 1.1).
- **internal to output bridge**: a bridge caused by a defect connects together an internal to an output node. This may also happen inside one cell or between two different cells (e.g. d_5 in figure 1.1).
- **single stuck-at bridge**: a bridge caused by a defect connects either V_{dd} or V_{ss} to an output node (e.g. d_6 in figure 1.1). This type of bridge directly shows typical stuck–at behavior.

Regarding the network topological level, the single bridging fault again can be divided as **feedback bridge** and **non-feedback bridge**. A feedback bridge is a bridge that causes the output of one bridged cell having at least one fanout path to the input of another bridged cell. Otherwise the bridge is called non–feedback bridge. Figure 2.5 illustrates a feedback bridge.

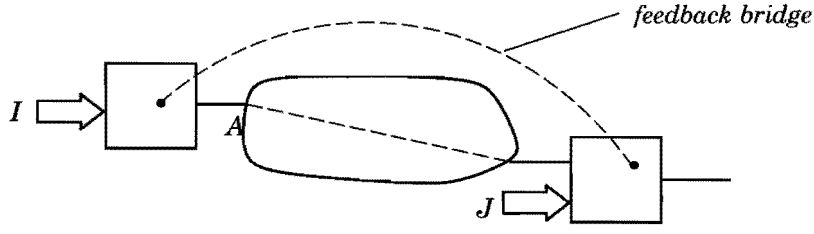


Figure 2.5 Illustration of a feedback bridge.

The above definitions and classifications are used throughout the whole thesis.

2.3.2 Analysis of the results of some extraction experiments

Early results of using the method described in this chapter to analyze NMOS circuits were presented in [16]. Assume all possible defect mechanisms may occur and the size of defects is in a certain range. The analysis shows that the most likely faults are bridging and line open faults. Other peculiar faults, such as new parametric transistors, have very low probability of occurrence. The combination of different types of faults, such as a bridge and an open caused by one defect, are also rare. It is further observed that the probability of occurrence of a single bridge or an open is much higher than that of a multiple one although the number of multiple faults can be half of the total

of the extracted faults. The dependence of the extracted faults to possible variations of the manufacturing line is also considered by taking different defect statistics into account. The results show that the same fault may have a different probability of occurrence. Moreover, the weight increase is not uniform for every fault. The experimental results [50] for some product chips also indicate the influence of the defect statistics. Below the results for CMOS circuits using the analysis system presented in [57] are presented.

For this set of benchmark circuits, we only consider missing or extra *metal1*, *metal2*, *poly* and *thin or thick oxide* layers since these layers usually occupy the most part of a layout. The critical areas are extracted for defect sizes ranging from 0μ to 20μ with an increment step of 5μ (after 2μ). The size distribution, as shown in figure 2.6, is taken as in [20] with 2μ as its peak size.

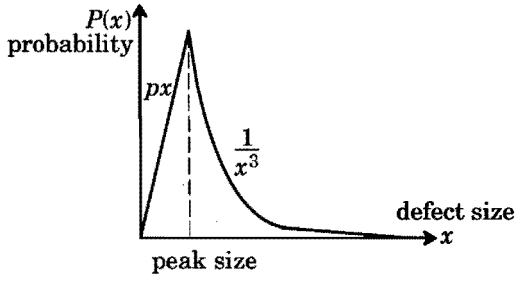


Figure 2.6 A typical defect size distribution.

Table 2.1 Some extraction results

circuit	circuit data				extracted opens			extracted bridges		
	#PI	#PO	#trans.	#cell	#open	#%	W%	#bridge	#%	W%
c432	36	7	728	152	2172	21.5	42.5	7932	78.5	57.5
c499	41	32	1396	284	4805	25.4	44.5	14144	74.6	55.5
c880	60	26	1164	236	4227	22.8	43.8	14335	77.2	56.2
c1355	41	32	1768	366	6858	29.2	46.1	16623	70.8	54.9
c1908	33	25	2058	411	7195	25.1	45.0	21467	74.9	55.0
c2670	157	64	2974	604	8757	17.8	38.7	40481	82.2	61.3
c3540	50	22	4122	791	14718	21.7	46.6	53189	78.3	53.4
c5315	178	123	6734	1288	20743	16.8	39.9	102485	83.2	60.1
c6288	32	32	8464	1848	32687	28.6	42.5	81787	71.4	57.5
c7552	206	107	8854	1795	29962	18.0	44.5	135832	82.0	55.5

#PI: primary inputs. #%: percentage of each type over total extracted faults.
#PO: primary outputs. W%: percentage of relative weight over total weight.

Because of the low probability of occurrence of some peculiar faults, only the

bridging and open fault types are extracted. Table 2.1 shows some statistics of the circuit and the results of the extraction.

For each circuit, both the percentage of each type of the extracted faults and the respective percentages of the relative weight over the total weight are listed. As can be expected, the open faults are much less than the bridging faults. This is because open faults usually involve a single network node and its fanout trees while bridges can be as many as the number of combinations of all network nodes. On average, opens only account for about 22.7% of all extracted faults. However the relative weight of the opens is not necessarily smaller than the ones of bridges. On average, the relative weight of opens is about 43.4%. That is, statistically both bridge and open have the same possibility of occurrence.

Table 2.2 Classification of extracted bridging faults

circuit		single bridge					
	#multi.	#out–out	#ssa	#in–in	#in–out	#other	#feedback
c432	4922	1906	376	236	355	137	1073
c499	8312	3677	650	401	835	269	1456
c880	8727	3797	592	392	567	260	818
c1355	9411	4925	814	555	724	194	1877
c1908	12750	5793	888	636	1004	396	2139
c2670	21235	15030	1522	932	1248	514	2089
c3540	29313	17981	1682	1448	1958	807	4365
c5315	50344	43040	2932	2225	2747	1197	4776
c6288	46652	24495	3760	2520	3563	797	12277
c7552	64834	57792	4002	2827	4839	1538	8260
#%	55.92%	31.85%	3.98%	6.79%		1.45%	8.50%
W%	2.20%	79.77%	14.18%	3.72%		0.134%	21.83%

#multi.: multiple bridge. #out–out: output to output bridge. #ssa: single stuck–at bridge. #in–in: internal to internal bridges. #in–out: internal to output bridges.

The bridging faults can be further distinguished as single and multiple bridges. The total number of them is listed in table 2.2. Concerning the node type, the total number of classified single bridges is also listed in table 2.2. The *input to input* type bridges are not included since it is easy to detect them. Other unclassified bridges are listed under the category of *#other*. They include the bridges between primary inputs and internal nodes or bridges between V_{dd} and an N–type internal node, etc. The bridges between an internal node in the P–part and an internal node in the N–part are also

classified as belonging to this category. Thus actually the internal to internal bridges under the category of #in–in only consist of the bridges either in the P–part or the N–part. The number of feedback bridges is also listed. The percentage of each type of bridge and its relative weight is shown in figure 2.7 and figure 2.8 respectively for each circuit. The last two rows of table 2.2 list the total percentage of each type of bridge and its relative weight for the complete set of circuits. This is also illustrated by figure 2.9.

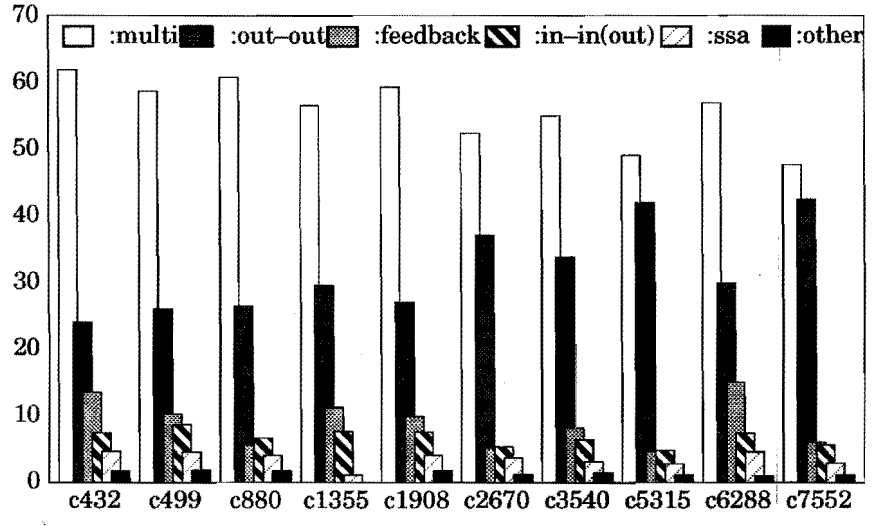


Figure 2.7 Relative number of different types of bridges versus circuits.

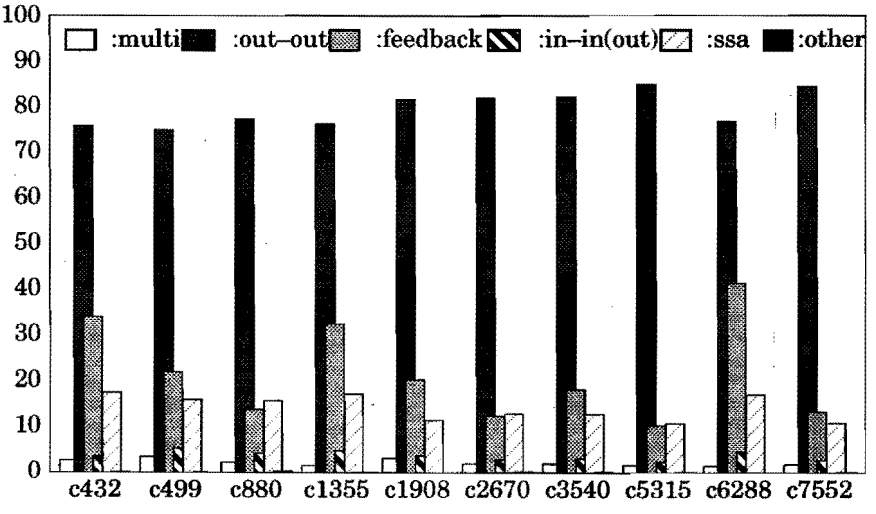


Figure 2.8 Relative weight of different types of bridges versus circuits.

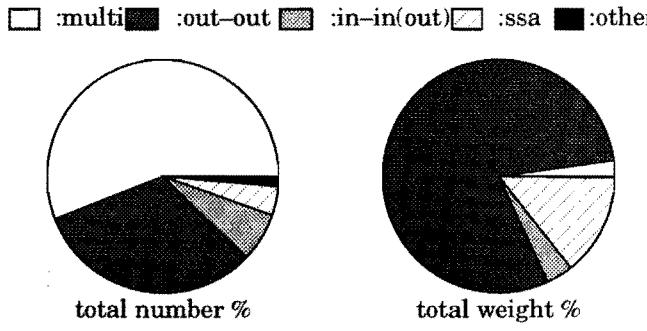


Figure 2.9 Average relative number and weight of different bridges.

From the above results, it can be seen that though there is some variation between different circuits, in general, the multiple bridges are the majority of bridges (55.9%). But their relative weight is very low (only 2.2% !). This is expectable since usually the multiple faults occur only when large defects are present in the layout. Most actually measured defect size distributions show that the probability of the occurrence of large defects is relatively small. This implies that for a normal design, single bridges occur more often than multiple bridges. It is interesting to observe that the single stuck–at bridges only count on average about 3.98% of extracted bridges. The relative weight is not very high either (about 14.18%). The percentage and the relative weight of the internal to internal node bridges are less than 10%. As for other peculiar type of bridges, both the number and their relative weight are very low. Obviously they are insignificant for this set of benchmarks. As one might have already expected the majority of the single bridges are *output to output* bridges (about 31.89%). Their relative weight is very high (79.77% !). This is predictable since in cell–based designs the related wires are much longer than the connecting wires inside a cell or between two adjacent cells. Consequently their critical areas are relatively large.

The feedback bridging faults are also identified. For some circuits, the feedback bridges can be 15% of all extracted bridges. On average, there are about 8.5%. But their relative weight (21.83%) is higher than that of single stuck–at and internal to internal bridges.

To summarize, it can be concluded that for the layouts of this set of benchmark circuits, in terms of both the number and its relative weight of each type of bridge, the output to output node bridges should receive the highest attention. Then next in order are feedback bridges, single stuck–at bridges and internal to internal node bridges. The very last ones are multiple and other peculiar bridges. It can be speculated that for other cell–based design styles, similar statistics regarding the type of bridges can be obtained.

For the same functionality, it is obvious that different implementations may result in completely different scenarios. This is reflected by the circuits c499 and c1355 since functionally they are the same but the extracted faults are different. It will be seen in later chapters that their testability for defect–induced faults is also different.

2.4 Conclusions

With the aid of a flexible analysis tool using the statistical relation developed in this chapter, the analysis of a set of circuits shows that the faults are dependent on the circuit layout and the defect statistics. The conventional single stuck–at faults are only a subset of all possible faults under spot defect conditions. Furthermore single faults have a higher probability of occurrence than multiple faults. The *output to output* node bridges have the highest probability of occurrence. Thus studying the impact of these faults for testing should be given higher priority. This thesis will focus on the single faults only. In the sequel, the term "fault" is implicitly referring to the notion of a "single fault". The reason of choosing single faults is not only based on the results of the statistical study. From the testing point of view, it can be expected that the large defects affecting more network nodes (multiple faults) can be easily screened out in the early phase of processing by conventional testing methods. Only the defects affecting one or two nodes are hard to detect. As for feedback bridges, they are not considered in this thesis since they induce usually unpredictable sequential behavior. They most likely show timing errors rather than some static fault. For bridging faults, the scope of this thesis is confined to the static analysis.

3 Bridging Fault Modeling and Simulation with Approximate Accuracy

3.1 Introduction

The previous chapter viewed some statistics of defect–induced faults. With this information available, this chapter will focus on one particular type of fault, namely the single bridging fault. Its behavior will be examined and a method of modeling and simulating the bridging faults will be investigated. It is difficult to analyze the electrical behavior of a bridging fault accurately. In this thesis only static analysis is performed by simulations. Furthermore, the defects considered are fatal defects. That is, the resistance of the defects is considered to be negligible. Thus bridged nodes are forced to have the same potential.

Brief analysis shows that with very few exceptions, the basic problem of modeling is associated with the conducting circuit from power supply to ground caused by a bridge. To illustrate, table 3.1 shows the SPICE simulation results for the bridges in figure 3.1 (the number next to each transistor indicates the relative size of the transistor and we maintain this convention in the sequel). It can be seen that for inputs activating these bridges, there is a conducting circuit from power supply to ground. It may result in the bridged output having a voltage value ranging from the potential of power supply (V_+) to ground (V_-). The actual output voltage value depends on how the cells are driven. Such an output is different from a normal logic "1" value driven only by the pull–up or a logic "0" driven only by the pull–down part of a cell. In this situation the output voltage value cannot be easily interpreted as a logic value since it depends on how it drives fanout cells. Figure 3.1 also shows a fanout situation. For the applied input *abcdef*=100111 (the quoted 1s(0s) are fault free values), the output bearing a value 2.10V can drive x to 4.20V which can be read as "1" and y to 1.43V which can be read as "0". Usually the output is said to be in "unknown state". Clearly the basic phenomena caused by a bridging fault is that a digital circuit is changed

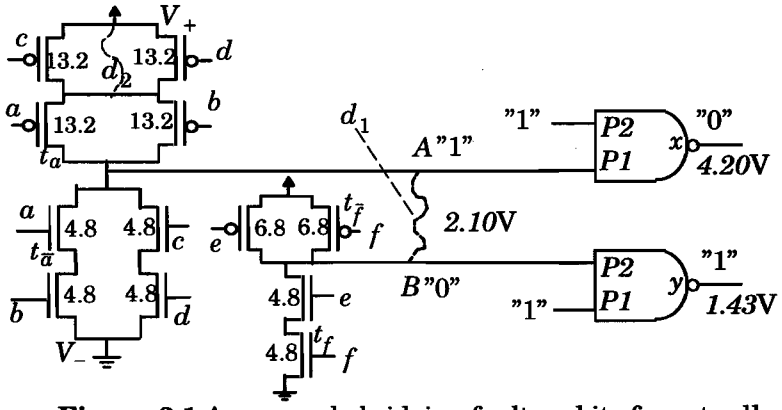


Figure 3.1 An example bridging fault and its fanout cells.

Table 3.1 Bridged output

bridge	inputs		output	bridged output	
	a b c d	*e f*	*A B*	SPICE(V)	switch
d_1	1 0 0 1	1 1	1 0	2.10	x
	0 0 0 1	1 0	1 0	2.87	x
	0 0 0 0	1 1	1 0	3.72	x
	1 1 0 1	0 0	0 1	3.40	x
	1 1 1 1	0 1	0 1	0.96	x
d_2	1 0 1 1		0	3.71	x
	0 0 1 1		0	4.40	x

x: unknown state.

into a circuit with unknown behavior. The exact behavior can only be obtained by simulating the bridging fault along with its fanout cells up to the primary outputs by using a circuit level simulator. In view of the large number of extracted faults, obviously it is very hard to achieve circuit level accuracy for a large circuit within an acceptable amount of time.

At the time that this research was started, a lot of methods [1,4,12,22,27, 31,39,43,44,46,49] have been developed to solve the above problem. Most of them intend to use a switch–level model [8, 9] to model and simulate the bridging faults. Such method models a transistor either as an ideal conductor with a constant conductance (strength) or with zero conductance and only the strongest path is used for the decision. Using such a simulator, the results for the bridge d_1 and d_2 in figure 3.1 are listed at the last column of table 3.1. The unknown state, denoted as "x", is usually obtained at the bridged output and carried through the rest of the simulation. This may give too pessimistic or too optimistic solutions and does not solve the problem. Among the various methods of improving the inadequate switch–level model, most intend to use

a resistive network model [1] to evaluate a bridging fault and interpret the output of the bridged cell as a logic level simply by comparing the conductances of pull–up and pull–down parts of a conducting circuit. However these methods have several limitations. First the model used is not accurate enough to predict the output voltage for a bridge. The second limitation is already illustrated in figure 3.1. For input *abcdef*=100111, the pull–down conducting strength of B is stronger than the pull–up conducting strength of A. The output is predicted as "0". But in fact it can be read as "0" by x and "1" by y as shown in figure 3.1. Furthermore most of them are not fully aware of the results of IFA at the time. As a result, the developed methods usually target just for one particular type of bridging fault.

Since it is very expensive to resolve all the unknown states, based on some experimental observations, this chapter presents a new method which tries to eliminate the unknown states as much as possible at the local cell level while maintaining the modeling and simulation efficiency. This method covers more types of bridges than other methods. It is effective for most of the bridging faults. Part of this chapter was previously published in [17].

3.2 A logic modeling and simulation strategy

It is evident that a bridging fault can be accurately modeled if

1) the output behavior of a bridged cell can be accurately evaluated;
2) an unknown input voltage value can be correctly interpreted as a logic value after it is propagated through subsequent cells.

Assume when a bridging fault is activated, the output voltage of the bridged cell is accurately computed. Now let us examine how the output can be propagated through subsequent cells. In theory, the interpretation process seems not an easy task since, in the worst case, it may require the simulation of the entire circuit in order to distinguish an unknown state. However the simulation results in table 3.1 for two bridges shown in figure 3.1 give the impression that most of the time the outputs of bridged cells have a value either below 2.0V or above 3.0V. For a normal design implemented for a typical 5.0V technology, these values can be locally interpreted as logic "0" or "1" respectively without any propagation along its fanout cells. For a specific technology, a highest logic "0" voltage V^0 and a lowest logic "1" voltage V^1 can be defined. In this chapter, any voltage value higher than V^1 is said to be in the **logic "1" range** and any voltage value lower than V^0 is said to be in the **logic "0" range**. Otherwise it is said to be in the **undefined range**. Inspired by the above observation, it can be expected that if this is the case for most of the bridging faults, then a great amount of computations can be avoided.

To verify such an observation, an intensive analysis using SPICE simulations has been conducted for many designs including the ISCAS85 benchmarks listed in chapter 2. Table 3.2 summarizes the results for each circuit of the ISCAS85 benchmarks. The table shows the percentage of all cases that the output voltage of bridged cells is either in logic "1" or logic "0" ranges. On average, 96.62% of the output values of bridged cells indeed fall into the distinguished logic ranges.

Table 3.2 SPICE simulation results

circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
logic%	96.4	97.1	96.0	96.6	97.1	96.7	97.2	95.9	97.6	95.6

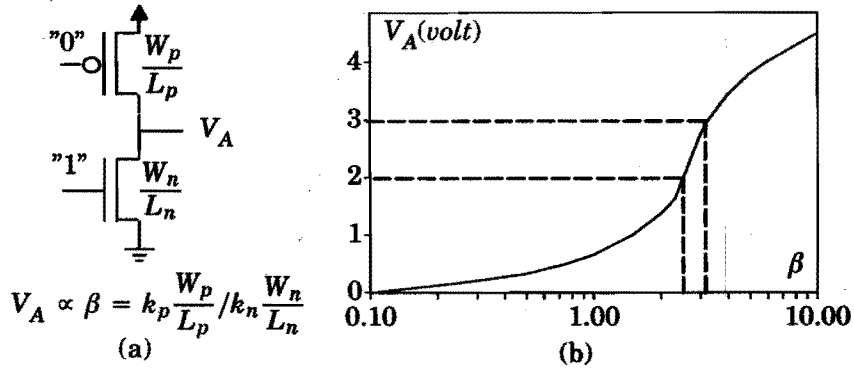


Figure 3.2 (a) A simple conducting circuit. (b) Output voltage versus β.

This probably can be better explained by simulating a simple conducting circuit shown in figure 3.2(a). Figure 3.2(b) shows the output voltage of the simple circuit as a function of pull–up to pull–down beta ratio $\beta = k_p \frac{W_p}{L_p} / k_n \frac{W_n}{L_n}$, where k_p and k_n are process dependent parameters and $\frac{W}{L}$ is the transistor width–length ratio. It can be observed that the output voltage ranging between 2.0V and 3.0V results in a rather narrow range of β between 2.7 and 3.1 for this specific technology. This may imply that the probability of a bridge to cause an equivalent β in the undefined range is small. That is, the probability of having an output voltage in the logic ranges is large. As for the bridges involving internal nodes, such a bridge usually splits some pull–up (pull–down) paths into two parts. In order to detect the bridge, part of the split paths needs to be activated as shown by the input in figure 3.1(b) for d_2. The equivalent β of a partial pull–up path versus a partial or complete pull–down path usually has large chance that the resulting β value falls into the logic ranges.

This experimental observation, in fact, can help to eliminate a lot of unknown states and still allows fast simulation since the fault propagation can simply be realized by using the following principle:

Modeling principle: If the output voltage of a bridged cell is higher than V^1, then a logic "1" is read at the output. If the output voltage of a bridged cell is lower than V^0, then a logic "0" is read. Otherwise it is considered that the fault effect will not appear at the output of the bridged cell.

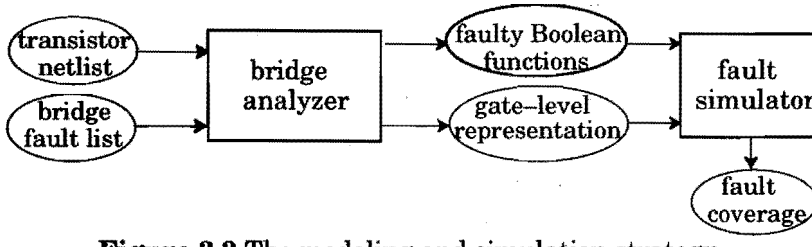


Figure 3.3 The modeling and simulation strategy.

With the above principle, the whole fault modeling and simulation can be done in the way as illustrated in figure 3.3. Assume the transistor netlist and the extracted bridging faults are available. The logic level representation of the circuit is extracted from the transistor netlist. Then, for each extracted bridging fault, a local circuit analysis is performed only for those inputs that would cause a conducting circuit. Each evaluated output voltage of a bridged cell can be read as a logic level by using the above principle. Collecting the analysis results for all the possible inputs, the behavior of the bridged cells can be characterized in terms of a Boolean function. Since such a Boolean function partially (or completely) describes the faulty behavior of the bridged cells caused by this bridge, it is named the **Faulty Boolean Function**. After all the bridges are processed, a set of faulty Boolean functions is obtained. Then the fault simulation can be conducted at the logic level by just manipulating the faulty Boolean functions. Thus any efficient logic fault simulation technique can be used. Since there is no need to perform circuit level computations any more, the fault simulation can be very fast.

3.3 An approximate evaluation method

Now let us examine how to evaluate a bridging fault efficiently. Although the cells are relatively small, the full analysis of using a circuit simulator can still be very time–consuming considering the large number of possible bridging faults extracted from the layout. Thus instead of using a circuit level simulator, it is interesting to know whether there is any other way to evaluate

a bridging fault. The principle used here only tries to eliminate some of the unknown states by confining the voltage range, thus it is sufficient if the computed value is accurate enough within the defined logic voltage ranges. Such a possibility is investigated by using an approximate transistor model.

Using such an approximate transistor model, the dc–characteristic of an NMOS transistor is characterized as (a PMOS is modeled in the same way):

$$\begin{cases} I_{ds} = k_n \frac{W}{L}(V_{gs} - V_{tn} - \frac{1}{2}V_{ds})V_{ds}\ , & V_{ds} < V_{gs}-V_{tn} \\ I_{ds} = 0, & \text{otherwise} \end{cases}$$

where V_{tn} is the zero–bias threshold voltage, k_n the process dependent parameter, and $\frac{W}{L}$ the transistor width–length ratio. In the model, a transistor works in a linear region if it conducts, otherwise it is off. This is because in a conducting circuit the voltage level at any drain (source) cannot be higher than V_{dd} when the gate of the transistor is driven by a logic "1". Thus $V_{ds} < V_{gs}-V_{tn}$ is always true. The model also neglects the body–effect of the MOS transistor. It should be noted here that this model is still a nonlinear model which is different from others, such as the one used in [49]. Thus the model is more accurate than other approximate ones using a resistive network model [1] or a linear transistor model [49].

Table 3.3 SPICE results versus approximate method

circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
actual%	96.4	97.1	96.0	96.6	97.1	96.7	97.2	95.9	97.6	95.6
approx.%	93.8	94.9	92.0	92.2	94.1	93.4	94.4	91.6	95.6	92.1
voltage dif.(v)	0.08	0.14	0.15	0.15	0.14	0.16	0.13	0.16	0.11	0.18

The approximate transistor model above has been used to evaluate the bridging faults for the benchmarks described in chapter 2. The last row of table 3.3 shows the absolute average difference between the computed values and the SPICE results for each circuit. For the whole set of benchmarks, the average difference is about ± 0.14V from the SPICE results. The third row of table 3.3 shows the percentage of all cases where the outputs are correctly predicted within logic ranges using the approximate method. On average, 93.41% of all output values are correctly predicted within the logic ranges for the whole set of benchmarks. The actual percentages computed by SPICE are shown in second row of table 3.3 as well. As can be expected, the actual percentages computed by SPICE are higher than the percentages by using this approximate method.

For the bridges in figure 3.1, the last two columns of table 3.4 show the estimated voltage value and the predicted logic level using this model and the

principle. It should be noted here that according to the modeling principle, if an output is in undefined state, the fault free value is assumed at the output.

Table 3.4 Comparison with SPICE

bridge	inputs		output	bridged output		
	a b c d	*e f*	*A B*	SPICE(V)	approximate(V)	level
d_1	1 0 0 1	1 1	1 0	2.10	2.35	x*
	0 0 0 1	1 0	1 0	2.87	2.99	1
	0 0 0 0	1 1	1 0	3.72	3.53	1
	1 1 0 1	0 0	0 1	3.40	3.10	1
	1 1 1 1	0 1	0 1	0.96	1.14	0
d_2	1 0 1 1		0	3.71	3.10	1
	0 0 1 1		0	4.40	3.10	1

x* : undefined state.

This method still appears a little bit pessimistic compared with SPICE. However the model does have the advantage that any conducting circuit can be transformed into the one shown in figure 3.2(a) with $\frac{W_n}{L_n}$ and $\frac{W_p}{L_p}$ as the equivalent width–length ratios of the respective pull–up and pull–down parts. The equivalence can be established according to following rules:

1) two serial connected transistors with W_1/L_1 and W_2/L_2 can be replaced by a transistor with $W/L = W_1/L_1 + W_2/L_2$.
2) two parallel connected transistors with W_1/L_1 and W_2/L_2 can be replaced by a transistor with $W/L = \frac{W_1/L_1 \times W_2/L_2}{W_1/L_1 + W_2/L_2}$.

These two rules can be derived using the approximate transistor model.

For the equivalent conducting circuit, the output voltage V_A can be derived by solving the following equation,

$$(\beta - 1)V_A^2 + 2(V_+ - V_{tn} - \beta V_{tp})V_A - \beta(V_+ - 2V_{tp})V_+ = 0$$

where $\beta = (k_p\frac{W_p}{L_p})/(k_n\frac{W_n}{L_n})$. It is not difficult to prove that V_A is an increasing function of β. A value of β^1 exists so that $V_A = V^1$ and a value of β^0 exists so that $V_A = V^0$. Therefore

$$\beta > \beta^1 \Rightarrow V_A > V^1 \qquad \text{and} \qquad \beta < \beta^0 \Rightarrow V_A < V^0$$

holds which implies that, for a specific technology, it is not even necessary to solve all equations for the output voltage. Only the equivalent β value is needed. Consequently the evaluation can be very fast. Below this method is used to construct the faulty Boolean functions.

3.4 Specification of faulty Boolean functions

For ease of extraction, each CMOS circuit is represented by a **connection graph**, $G = (V, E)$. Each vertex $v \in V$ represents a network node which can be of the type *input node*, *output node* or *internal node* as defined in chapter 2. An undirected edge $e \in E$ represents a transistor and has an associated Boolean variable (defined by its gate input function) and a weight representing its transistor width–length ratio. As an example, the graph representation of the circuit in figure 3.4(a) is shown in figure 3.4(b).

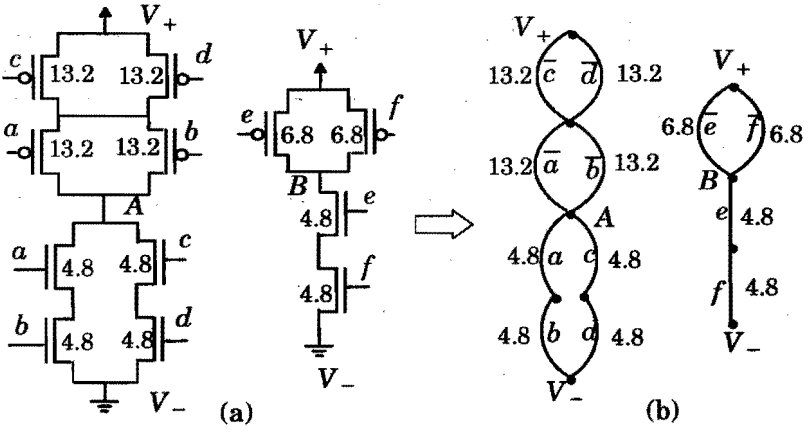


Figure 3.4 Illustration of the connection graph representation.

A **simple path** between a and b is denoted as s_{ab}. P_{ab} is the set of all distinct paths between a and b. A **term** T_s of a path s is the product of all Boolean variables in s. A path s conducts iff $T_s = 1$.

Using the above notations, the fault free Boolean function F_A of a cell (with its output node as A) can be expressed by all its "on" terms or "off" terms as:

$$F_A = \sum_{s \in P_{AV_+}} T_s \tag{3.1}$$

or

$$\overline{F}_A = \sum_{s \in P_{AV_-}} T_s \tag{3.2}$$

Obviously, F_A is the "on" set of A while $\overline{F}_A$ is the "off" set of A.

In case of a bridging fault, the output of a bridged cell is said having a **faulty–on** behavior if the fault free output is "0" but in case of the bridge the output is "1". Vice versa the output of a bridged cell has a **faulty–off** behavior if the fault free output is "1" but in case of the bridge the output is "0". Otherwise the cell is said to be fault free.

Applying the modeling principle here, a faulty–on is caused if the output voltage is higher than V^1 but fault free output is "0". A faulty–off is caused if the output voltage is lower than V^0 but fault free output is "1".

Assume a bridge between the cell with output A and another cell with output B as it is illustrated in figure 3.5(a).

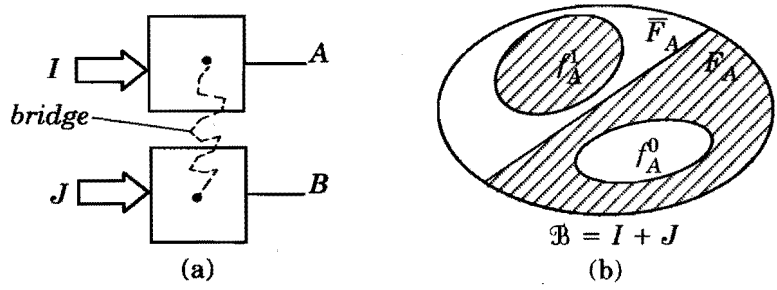


Figure 3.5 (a) Illustration of an arbitrary bridge. (b) Its new input space.

Let F_A be the fault free function of A. Regarding its input space, F_A can be expressed as:

$$F_A = \{x \in I \mid A \text{ is "on"}\}$$

In the presence of the bridge, A and B become functions of both inputs I and J. Let the new input space be denoted as $\mathfrak{B}$, that is, $\mathfrak{B} = I + J$. Below we just specify the faulty Boolean function of A. The faulty Boolean function of B can be derived in the similar way. The faulty Boolean function $\tilde{F}_A$ of A in the presence of the bridging fault is defined as:

$$\tilde{F}_A = \{x \in \mathfrak{B} \mid A \text{ is "on"}\} \tag{3.3}$$

Then the **faulty–on set** and the **faulty–off set** of A are defined as:

$$f_A^1 = \{x \in \mathfrak{B} \mid \overline{F}_A \wedge \tilde{F}_A\} \tag{3.4}$$

$$f_A^0 = \{x \in \mathfrak{B} \mid F_A \wedge \overline{\tilde{F}_A}\} \tag{3.5}$$

The complement of f_A^1 and f_A^0 are obtained as

$$\overline{f_A^1} = \{x \in \mathfrak{B} \mid F_A \vee \overline{\tilde{F}_A}\} \tag{3.6}$$

$$\overline{f_A^0} = \{x \in \mathfrak{B} \mid \overline{F}_A \vee \tilde{F}_A\} \tag{3.7}$$

The set $\mathfrak{B}$ is then split into three parts: faulty–on set f_A^1, faulty–off set f_A^0 and the rest of the inputs. Obviously $\mathfrak{B}$ can also be viewed as the union of F_A and $\overline{F}_A$ respectively considering the inputs in J as "don't cares". Figure 3.5(b) illustrates the relation of f_A^1 and f_A^0 with respect to F_A and $\overline{F}_A$.

With the above definitions, the following theorem holds.

Theorem 3.1: Assume a cell with its fault free function as F_A is affected by a bridge. Let f_A^1 and f_A^0 be the faulty–on set and faulty–off set of the cell. Then

$$\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1 \tag{3.8}$$

Proof: Eq.(3.3) can be partitioned into two parts:

$$\tilde{F}_A = \{x \in \mathfrak{B} \mid F_A \wedge (A \text{ is "on"})\} \cup \{x \in \mathfrak{B} \mid \overline{F}_A \wedge (A \text{ is "on"})\} \tag{3.9}$$

In the first part, the set containing the original "on" set F_A, except the inputs in f_A^0, A is still "on". Thus the first subset in eq.(3.9) should be the original "on" set F_A minus the faulty–off set f_A^0 (the shaded part in figure 3.5(b)). The second subset in eq.(3.9) is exactly the faulty on set f_A^1. Thus

$$\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1.$$

❑

Theorem 3.1 shows that if f^1 and f^0 of a cell are obtained, the behavior of a bridging fault can be characterized.

For the above specified faulty Boolean function, using the modeling principle in 3.2, the following corollary is true.

Corollary 3.1: Assume a bridge affects the outputs A and B of two different cells and their fault free functions are F_A and F_B respectively. Assume that the faulty–on sets and faulty–off sets are obtained as f_A^1 and f_A^0, f_B^1 and f_B^0 respectively. Then their faulty Boolean functions

$$\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1 \tag{3.10}$$

and

$$\tilde{F}_B = F_B \cdot \overline{f_B^0} + f_B^1 \tag{3.11}$$

have the following property:

$$(\tilde{F}_A \oplus F_A) \cdot (\tilde{F}_B \oplus F_B) = 0 \tag{3.12}$$

Proof: The proof is conducted for each type of bridge.

1) For an *output to output* node bridge, the proof is easy. After substitution of eq.(3.10), (3.11) into (3.12), the eq.(3.12) can be expanded and simplified as:

$$(\tilde{F}_A \oplus F_A) \cdot (\tilde{F}_B \oplus F_B) = (F_A \cdot f_A^0 + \overline{F_A} \cdot f_A^1) \cdot (F_B \cdot f_B^0 + \overline{F_B} \cdot f_B^1) \tag{3.13}$$

The first two products of eq.(3.13) $F_A \cdot F_B \cdot f_A^0 \cdot f_B^0$ and $\overline{F_A} \cdot \overline{F_B} \cdot f_A^1 \cdot f_B^1$ obviously cannot be true since F_A and F_B being both a "1" or a "0" imply

that there is no conducting circuit from V_+ to V_-. Thus $f_A^0 \cdot f_B^0 = 0$ and $f_A^1 \cdot f_B^1 = 0$. The other two products $F_A \cdot \overline{F_B} \cdot f_A^0 \cdot f_B^1$ and $\overline{F_A} \cdot F_B \cdot f_A^1 \cdot f_B^0$ cannot be true either since the bridged output voltage cannot be in the logic "1" and "0" ranges simultaneously. That is, $f_A^0 \cdot f_B^1 = 0$ and $f_A^1 \cdot f_B^0 = 0$. Thus the corollary is true.

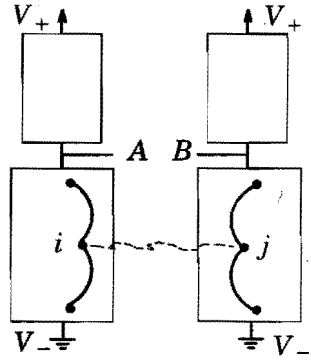


Figure 3.6 Illustration of an arbitrary internal–internal node bridge.

2) For an *internal to internal* node bridge, a bridge between an internal node in the P–part of a cell and an internal node in the N–part of a cell is not considered since such kind of bridge can hardly occur. To be general, assume a bridge occurs in the pull–down parts of two cells as shown by a bridge between i and j in figure 3.6. F_A and F_B are the fault free function of A and B respectively.

The faulty–on and faulty–off sets of A and B can be derived from any two path segments s_{V_+i} and s_{jV_-} or s_{V_+j} and s_{iV_-}. Let us analyze a two–path segment s_{V_+i} and s_{jV_-} first. Path s_{jV_-} can be further classified as the one across node B, denoted as s_{jBV_-}, and the one without across node B, denoted as $s_{j\overline{B}V_-}$.

For any input establishing conducting paths s_{V_+i} and $s_{j\overline{B}V_-}$, obviously both $F_A = 1$ and $F_B = 1$. Since only output A is on the conducting circuit, thus if $V_A < V^0$, only a faulty–off behavior is caused at A.

For any input establishing conducting paths s_{V_+i} and s_{jBV_-}, $F_A = 1$ and $F_B = 0$. Both A and B are on the conducting circuit. Since $V_B < V_A$, thus if $V_A < V^0$, $V_B < V^0$ is also true. In this case only a faulty–off behavior is caused at A while B behaves as if it is fault free. Similarly, if $V_B > V^1$, only a faulty–on behavior is caused at B.

For any paths s_{V_+j} and s_{iV_-}, the analysis is similar. Thus for any possible situation, A and B cannot have a faulty behavior simultaneously. Therefore, $(\tilde{F}_A \oplus F_A) \cdot (\tilde{F}_B \oplus F_B) = 0$.□

The proofs for other types of bridges are similar. This corollary can help to obtain fast fault simulation.

3.5 The details of extracting the Faulty Boolean function

3.5.1 An extraction procedure

First the fault free functions should be extracted. According to eq.(3.1),(3.2) each one can be easily obtained by extracting the "on" paths (P_{AV_+}) or "off" paths (P_{AV_-}) using a depth first search routine.

To extract the faulty Boolean functions for a bridge connecting two nodes i and j, basically it suffices how to obtain the faulty–on set and the faulty–off set for each bridged cell. To obtain the faulty–on and faulty–off sets, all the conducting circuits caused by this bridge have to be analyzed. The conducting circuits can be obtained by tracing the actual transistors connected to the bridged nodes. Regarding the graph representation, a conducting circuit can be established by a path from V_+ to one bridged node and a path from another bridged node to V_-. That is, any two path segments s_{V_+i} and s_{jV_-} or s_{V_+j} and s_{iV_-} establish a conducting circuit. Let $s_{V_+ijV_-}$ and $s_{V_+jiV_-}$ denote such two path segments respectively. Let $P_{V_+ijV_-}$ and $P_{V_+jiV_-}$ be the path sets containing all of those paths respectively. In addition to an individual path, any non–empty subset of $P_{V_+ijV_-}$ or $P_{V_+jiV_-}$ can establish a conducting circuit as well. A set containing all of the conducting circuits is defined below.

$$\Phi=\{\theta \mid (\theta \subseteq P_{V_+ijV_-} \vee \theta \subseteq P_{V_+jiV_-}) \wedge \theta \neq \emptyset\} \tag{3.14}$$

For each $\theta \in \Phi$, its corresponding Boolean expression is defined as:

$$C_\theta = \sum_{s\in\theta} T_s \cdot \overline{\sum_{s\in P_{V_+ijV_-}\setminus\theta} T_s} \tag{3.15}$$

or

$$C_\theta = \sum_{s\in\theta} T_s \cdot \overline{\sum_{s\in P_{V_+jiV_-}\setminus\theta} T_s} \tag{3.16}$$

If C_θ is satisfied, then only the paths in θ conduct but no others. A conducting circuit θ is *valid* if C_θ is satisfiable. Consider the set Φ according to eq.(3.14) to be established, then the equivalent transistor width–length ratios of the

P–part and the N–part of each $\theta \in \Phi$ can be computed. Such a computation can be realized by iteratively applying the rules in section 3.2 on the graph representation of θ. The computation is linear in the number of transistors. In turn the beta ratio of each θ, denoted as β_θ, can be computed. Then the f^1 and f^0 can be obtained by applying Algorithm 3.1.

Algorithm 3.1: extraction of faulty Boolean function

$f^1 \leftarrow 0$; $f^0 \leftarrow 0$;
for each $\theta \in \Phi$ **do**
  construct C_θ;
  if C_θ satisfiable **then**
    compute β_θ;
    if $(\beta_\theta > \beta^1)$ **then**
      $f^1 \leftarrow f^1 + C_\theta$;
    else
      if $(\beta_\theta < \beta^0)$ **then**
        $f^0 \leftarrow f^0 + C_\theta$;

3.5.2 Obtaining conducting circuits

The set Φ in eq.(3.14) containing all the conducting circuits can be obtained by enumerating either the paths or the input space of the bridged cells. But usually such an enumeration process is not efficient. In this thesis, the problem of obtaining all the conducting circuits is formulated as a general graph problem and can be solved more efficiently. All the paths in set $P_{V_+ijV_-}$ or $P_{V_+jiV_-}$ can be viewed as a path–connected graph between V_+ and V_-. Then each element of Φ is viewed as a path–connected subgraph between V_+ and V_-. The detailed definition of path–connected graph and an enumeration algorithm to obtain all the path–connected subgraphs are presented in appendix A.

3.5.3 Boolean function representations issue

The idea of modeling bridging faults as a set of faulty Boolean functions before the fault simulation is simple and straightforward. The key issue is obviously how they can be represented and stored efficiently. If there is no proper way of handling the storage, then the proposed method is impractical since it may require a large amount of memory even for a small circuit in view of the large number of extracted faults. Fortunately the Reduced Ordered Binary Decision Diagram (ROBDD) data structure [6] has the feature of compactly

representing Boolean functions. During the analysis and simulation, the representation of a Boolean function and all its manipulations are based on a ROBDD package [28]. In fact, for a cell affected by a bridge, only its faulty–on and faulty–off sets are needed. Its faulty Boolean function can be easily constructed according to eq.(3.8). The compactness of using ROBDD can be illustrated by the following example. Figure 3.7(a) shows two bridges among three cells. After fault analysis, for bridge *#1*, we have

$$f_B^0(\#1) = \bar{d} \cdot (a \cdot b + c) \quad \text{and} \quad f_R^0(\#1) = d \cdot (\bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{c})$$

For bridge *#2*, we have

$$f_A^0(\#2) = \bar{e} \cdot (a \cdot b + c) \quad \text{and} \quad f_R^0(\#2) = e \cdot (\bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{c})$$

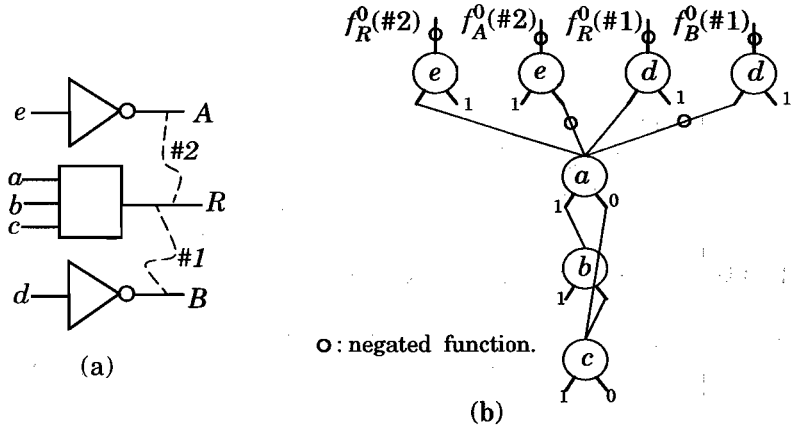


Figure 3.7 Illustration of compact representation.

Their ROBDD representations are shown in figure 3.7(b). It can be seen that for this fixed variable ordering, subexpressions $(a \cdot b + c)$ and $(\bar{a} \cdot \bar{c} + \bar{b} \cdot \bar{c})$ are shared by these faulty–off sets. Any other way of representation would require much more space. The compactness largely depends on the variable ordering and the actual bridges within the specific cells. But the bridges analyzed here only affect the outputs of not more than two cells. The number of their input variables is relatively small. The experimental results shows that this way is feasible.

3.5.4 Reduction of Boolean input space

Since it can be expensive to construct ROBDDs for a Boolean function, the efficiency of extracting a faulty Boolean function is mainly determined by the number of valid conducting circuits, i.e. the valid elements in the set Φ in eq.(3.14).

The fault analysis discussed so far assumes that the inputs of the bridged cells are independent. This assumption simplifies the implementation. The true validity of each conducting circuit can be verified later during the simulation phase. However the inputs usually have some relations among each other in terms of primary inputs. It can be the case that some of the conducting circuits do not exist in terms of primary inputs rendering their analysis to be unnecessary. For instance, analyzing the bridge in figure 3.1, if $f = a$, then any conducting circuit from V_+ to V_- consisting of either both transistors t_a and t_f or both $t_{\bar{a}}$ and $t_{\bar{f}}$ is not a valid one. Obviously it is expensive to check the validity by substituting local variables as functions of the primary inputs in a large circuit. To keep the analysis still at the local cell level, a reduction technique which makes use of certain implication relations is proposed below.

This technique requires a preprocessing step. Before fault analysis, such a preprocessing extracts all the following implications among any set of inputs of a cell:

1) The inverted variables. That is, if $F = \bar{a}$, $F \Rightarrow \bar{a}$ ($a \Rightarrow \bar{F}$) would be extracted ($\Rightarrow$ denotes implication);

2) Implicates of a function. For example, for $F = \overline{a \cdot b}$, $\bar{a} \Rightarrow F$, $\bar{b} \Rightarrow F$, $F \Rightarrow \bar{a}$ and $F \Rightarrow \bar{b}$ would be extracted;

3) Any two input functions to the same cell that satisfy an implication relation. For example, for inputs $F_1 = a \cdot (b+c)$ and $F_2 = a \cdot c$ to a cell, $F_2 \Rightarrow F_1$ and $\bar{F}_1 \Rightarrow \bar{F}_2$ would be extracted.

The extracted relations are stored and repeatedly used to derive valid conducting circuits for each bridging fault. Applying the above implications during the analysis, the input space of the bridged cells is only expanded one level down from the bridged site to the primary inputs. For each input variable f, let $I(f)$ be the product of all the input variables that are implied by f, i.e., $I(f) = \prod_{g \mid f \Rightarrow g} g$. The reduction may be achieved through the following procedure:

Algorithm 3.2: reducing invalid conducting circuits

for each $\theta \in \Phi$ **do**
  $exp \leftarrow$"1";
  for each input variable f in θ **do**
    $exp \leftarrow exp \cdot I(f)$;
  if exp satisfiable **then**
    a valid θ is found;

Though not all of the invalid conducting circuits can be avoided, a large portion of analysis can be bypassed by using this technique. The effectiveness of this technique varies from circuit to circuit.

3.6 A fault simulator for faulty Boolean functions

The set of faulty Boolean functions derived above for all the bridging faults can be easily used by any logic fault simulator. Here the well–known Parallel Pattern and Single Fault Propagation (PPSFP) [54] technique is adapted to show the easy exploitation of parallelism. First the classical fault simulation problem for single stuck–at faults is formulated. Then it is shown that a bridging fault can be simulated within the same framework.

The fault simulation is conducted on the network graph $G_n(V_n, E_n)$. Each node $v \in V_n$ represents a cell. Each directed edge $(v, u) \in E_n$ represents the relation that v is an input of u. Figure 3.8(a)(b) shows an example of a network and its network graph representation respectively. The network graph is levelized first by using a topological sort procedure. The nodes in V_n are arranged in increasing order according to the network level. Considering the fanout branches, the network can be decomposed into fanout–free regions by splitting a fanout node into a number of nodes equal to the number of its fanout branches. Figure 3.8(c) shows the network graph after decomposition. There are a total of four fanout–free regions for the example shown in figure 3.8(a).

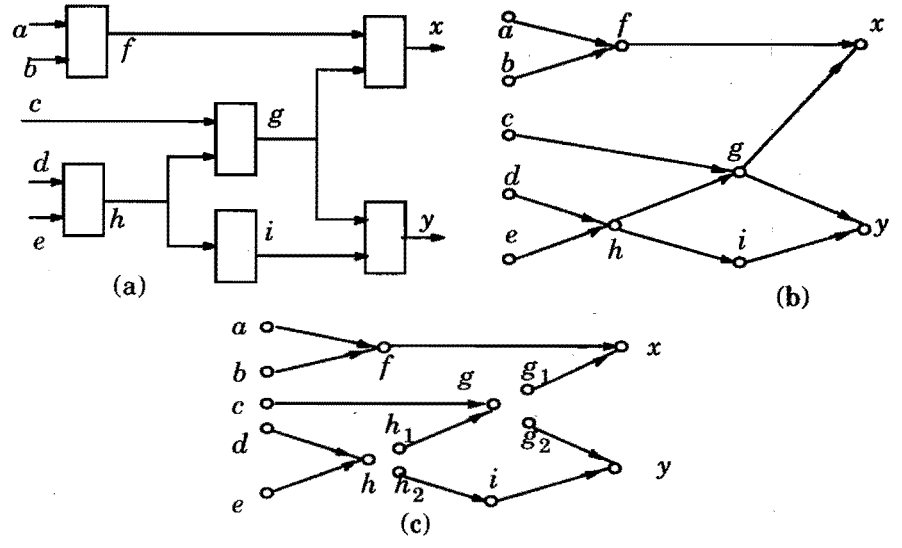


Figure 3.8 (a) A network. (b) Its network graph. (c) fanout free regions.

In the sequel, for a node F in the network, let ψ denote its function. Assume it has $a, b, \ldots, q$ as its inputs and let $F = \psi(a, b, \ldots, q)$. For an input a, let F_a denote the Boolean difference of F with respect to a. That is,

$$F_a = \psi(a, b, \ldots, q) \oplus \psi(\bar{a}, b, \ldots, q) \tag{3.17}$$

F_a is also called the **local observable function** of a. The value of F_a is called the **local observability** of a. If F_a=1, a is observable at F. Assume $P, Q, \ldots, Z$ are the functions representing the primary outputs, then for a node a, its **global observable function** O_a can be derived as

$$O_a = P_a + Q_a + \ldots + Z_a \tag{3.18}$$

Here P_a denotes the Boolean difference of P with respect to a(so does Q_a, etc). The value of O_a is called the **observability** of the node a. If O_a=1, the node a is observable at at least one of the primary outputs. That is, any change of the logic value of the node a results in a change of at least one of the primary outputs.

As the first step of fault simulation, the fault free simulation is conducted in order to determine the logic value for each node. Since the function of each cell is represented symbolically, a parallel pattern evaluation can be performed in a bit–vector manner. Let $vec(F)$ denote the bit–vector value of F. Then the parallel pattern evaluation of a function $F = \psi(a, b, \ldots, q)$ can be formulated as

$$vec(F) = \psi(vec(a), vec(b), \ldots, vec(q)) \tag{3.19}$$

Obviously the number of the patterns that can be simulated in parallel is the number of bits in the bit–vector, in our case, the length of an integer (machine word).

The next step, which is the major problem of fault simulation, is to determine a set of nodes $\{a \in V_n \mid O_a = 1\}$ for each input pattern. In theory, the observability of a node a can be obtained by evaluating its global observable function as defined in eq.(3.18). In practice, the observability of each node can be recursively determined using the following rules:

1) For each primary output P, $O_P = 1$ is always true (e.g. for x in figure 3.8, O_x=1).
2) For each node a in the fanout–free region, assume a is a predecessor of b, then

$$O_a = O_b \cdot b_a \tag{3.20}$$

b_a is the local observability of a. That is, they can be recursively determined from the top level nodes of each fanout–free region.

3) For each non–reconvergent fanout node a (e.g. the node g in figure 3.8), its fanout branches $a_1, a_2, \ldots, a_m$ are independent (they do not

reconverge). It is observable at the primary outputs if one of its branches is observable at the primary output. Thus

$$O_a = O_{a_1} + O_{a_2} + , ..., + O_{a_m} \tag{3.21}$$

4) For each reconvergent fanout node a (e.g. the node h in figure 3.8), since two fanout branches from a carrying faulty values may converge at some point, the faulty value may be masked out. Thus O_a should be determined by explicitly simulating the fault from the fanout node. That is, the value $\bar{a}$ (opposite to the fault free value a) should be explicitly propagated to the primary outputs. O_a is finally determined according to eq.(3.18).

After the observability of each node is determined for the current input pattern, the last step of determining the detectability of each fault can be carried out. For a single stuck–at fault at a node a, it is straightforward. It can be determined by evaluating

$$D_{a\text{-}s\text{-}1} = O_a \cdot a \tag{3.22}$$

or

$$D_{a\text{-}s\text{-}0} = O_a \cdot \bar{a} \tag{3.23}$$

If $D_{a\text{-}s\text{-}1}$ =1, a stuck–at–1 is detected and if $D_{a\text{-}s\text{-}0}$ =1, a stuck–at–0 is detected.

It is not difficult to observe that all the steps can also be performed for patterns in parallel. The bit–vector operations can be formulated as in eq.(3.19) for fault free evaluations.

The above two steps can be summarized as two traversals over the network graph described below:

1) In the forward traversal, 32 (the machine integer length in our case) patterns are applied to the primary inputs and simulated in increasing order of circuit level until the primary outputs are reached.

2) In the backward traversal, the observability of each node is evaluated in the way as described above for current inputs from primary outputs to primary inputs in the decreasing order of circuit level. At the meantime, the detectability of single stuck–at faults are determined according to eq.(3.22) and eq.(3.23).

Now let us examine how a bridging fault is simulated. Since each bridging fault is modeled as a Boolean function, bridging faults can be easily simulated in the same framework. In the backward traversal, after the observability of each node is obtained, the detectability of each bridging fault can be easily determined as follows:

1) For a bridging fault affecting only one cell, assume the fault free and the faulty Boolean function are obtained as a and $\tilde{a}$ respectively. Then its detectability can be obtained by evaluating

$$D_{bri} = O_a \cdot (\tilde{a} \oplus a) \tag{3.24}$$

If D_{bri} =1, then the bridge is detected.

2) For a bridging fault affecting two different cells, assume their fault free functions and faulty Boolean functions are a, b and $\tilde{a}, \tilde{b}$ respectively. From the corollary 3.1, it is know that $\tilde{a} \oplus a$ and $\tilde{b} \oplus b$ cannot be true at the same time. That is, for a given input pattern, either a has a faulty behavior or b has a faulty behavior but both of them cannot have faulty behavior simultaneously. In other words, the fanout branches from a carrying a faulty value would never converge with the fanout branches from b also carrying a faulty value. Thus the fanout branch of a and b are independent in terms of the faulty value propagation. Therefore the detectability of this bridge can be obtained by evaluating

$$D_{bri} = D_{bri}(a) + D_{bri}(b) = O_a \cdot (\tilde{a} \oplus a) + O_a \cdot (\tilde{b} \oplus b) \tag{3.25}$$

Again, if D_{bri} =1, then the bridge is detected.

Obviously, eq.(3.24)(3.25) can be evaluated for patterns in parallel as well. Since the detectability of each bridge is determined locally, the complexity of simulating a bridge remains the same as for simulating a single stuck–at fault.

3.7 Experimental results

The above system was implemented in C on a HP–9000/755 workstation. The ISCAS85 benchmark circuits and their bridging faults as described in chapter 2 are used for the experiments. Table 3.5 summarizes several extraction results. To have a certain safety margin, the highest logic "0" voltage V^0 is set to 1.5V and the lowest logic "1" voltage V^1 is set to 3.2V.

Table 3.5 Faulty Boolean function extraction results

circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
#bridge	1424	3454	3938	4327	5294	15121	17022	43236	18301	57198
#undete.	306	492	507	773	821	2044	2446	4898	4724	9702
time(s)	0.6	3.8	4.3	3.3	6.8	15.6	41.3	80.9	12.0	99.4
#Mbyte	0.06	0.22	0.26	0.22	0.41	1.00	1.45	3.83	0.78	5.44

#undete. : number of undetectable bridges. #Mbyte: memory requirement in Mbyte.

The set of bridges considered comprises only the output to output, internal to internal and internal to output type of single bridges. The CPU times listed

include preprocessing time, the times of extracting the fault free expressions and the times of extracting faulty Boolean functions for all the bridges. The amount of memory required to store the extracted faulty Boolean functions is shown in units of Mbytes. Both the CPU time and memory requirements versus the number of bridges are shown in figure 3.9 and figure 3.10 respectively. It can be observed that for these benchmarks both entities grow almost linearly with the number of bridges.

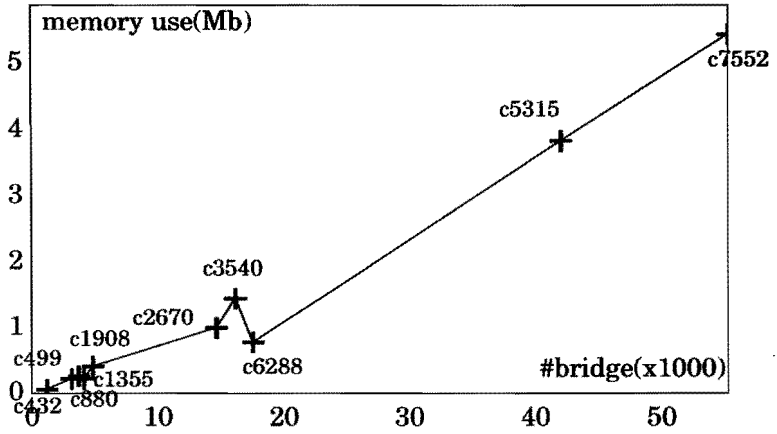


Figure 3.9 Memory requirement versus number of bridges.

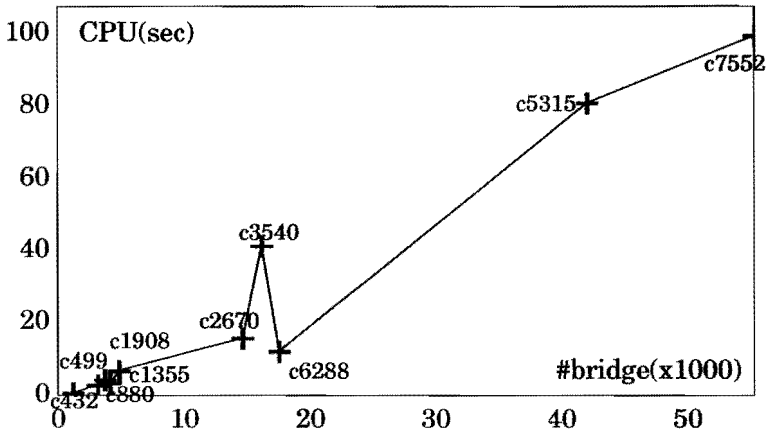


Figure 3.10 Analysis time versus the number of bridges.

In the course of the local analysis, some of the "undetectable bridges" are identified. The number of them is shown in table 3.5 for each circuit. Most of those "undetectable bridges" are internal to internal node bridges. The reason that these bridges are undetectable can be that for those bridges there does not exist any valid conducting circuit. In this case, these bridges are

redundant. It might be the case that there are valid conducting circuits but according to the modeling method in this chapter, the output of the bridged cell has the same logic value as the fault free value. Note, in this case, "undetectable" here only indicates that the bridge is not detectable in the defined logic ranges. But the detectability of such a bridge may not be entirely estimated since the method in this chapter is not able to model the output voltage in the undefined range.

The PPSFP simulation results for the modeled faulty Boolean function are shown in table 3.6. All of the faulty Boolean functions (i.e. except the "undetectable" ones) are simulated for the test patterns developed at the gate level for single stuck–at faults in MCNC. The percentage of the covered single stuck–at faults is based on the ones assumed at the output nodes of the actual CMOS implementation and not on the ones assumed within the gate–level representations. As already stated, the complexity of simulating a bridging fault is the same as simulating a single stuck–at fault. The fault simulation for bridging faults can be done very fast. Within a few dozen of CPU seconds, nearly 60000 bridges were simulated on a circuit of about 9000 transistors for more than 300 patterns. The bridging fault coverages are much lower than the stuck–at fault coverages. Since the modeling approach is approximate, the detectability of some of the undetected bridges cannot be fully determined for the given test pattern set. But this method gives definitely more confidence than the switch–level fault simulation does. Yet it has the full logic fault simulation speed. The above results cannot be achieved both at circuit and switch level.

Table 3.6 PPSFP simulation results

circuit	SSA test pattern set				21x32 random patterns		
	#patterns	SSA%	bridge%	time(s)	SSA%	bridge%	time(s)
c432	75	99.7	74.2	0.6	99.73	75.2	0.86
c499	71	100	85.1	1.2	99.5	85.2	1.8
c880	95	100	85.2	1.9	98.7	87.2	2.6
c1355	101	100	74.2	1.6	99.4	74.2	1.9
c1908	147	100	79.1	2.9	94.5	77.7	3.5
c2670	160	98.8	82.4	5.9	87.7	79.9	6.0
c3540	242	98.4	78.0	14.8	97.9	79.9	17.2
c5315	211	100	87.3	28.9	99.9	87.4	30.5
c6288	44	99.9	65.1	5.7	99.9	65.5	22.71
c7552	318	99.7	80.7	35.8	92.9	79.7	37.56

To verify the random testability of the extracted bridging faults, the extracted faulty Boolean functions are also simulated for 21x32 randomly generated patterns. The results are also listed in table 3.6. It can be seen that the bridging fault coverage is no better than for the single stuck–at test pattern set. Only five of the examples show a little improvement. This may imply that the random testability of bridging faults is poor.

Another interesting observation is that for the ISCAS85 benchmark set, many untestable single stuck–at faults assumed in the gate–level representation do not exist in this CMOS implementation. These faults may require major part of the Automatic Test Pattern Generation (ATPG) time. Therefore for the purpose of both the test quality and the test development time, the test patterns had better be generated from the physical design of a circuit instead from its gate–level representations.

3.8 Conclusions

The modeling and simulation method presented in this chapter is effective for most of the bridges. This supplies a fast and practical tool to analyze the testability of a relatively large CMOS design for bridging faults. The modeling accuracy is higher than switch–level and other approximation approaches. The idea of modeling bridging faults as a set of faulty Boolean functions before fault simulation achieves both the modeling accuracy and simulation efficiency. This divide and conquer approach can be used for any other upcoming technology and for non–structured designs as well. The limitation of the method, however, is also obvious. If the output of a bridged cell just remains in the undefined range (between V^1 and V^0), this method cannot predict the detectability of a bridge. As for the memory requirement, considering the overall gain of accuracy and simulation efficiency, the method is considered feasible.

4 Bridging Fault Modeling and Simulation with Circuit-level Accuracy

4.1 Introduction

In the previous chapter, a modeling and simulation method for bridging faults has been developed based on some statistical and experimental observations. It allows a very fast simulation of bridging faults for a large design and yet obtains higher modeling accuracy than switch-level or other approximation approaches. The limitation of this method is that if the output of a bridged cell is in undefined range, the detectability of the bridge cannot be fully determined. This chapter will tackle this problem and propose an efficient approach to solve it.

To examine the problem in detail, figure 4.1 shows an output to output bridge between a complex cell and a 2-in-NAND. The bridged output voltages computed by SPICE are listed in table 4.1. The logic levels of the bridged output predicted by using the modeling method proposed in chapter 3 are also listed under the column *level**.

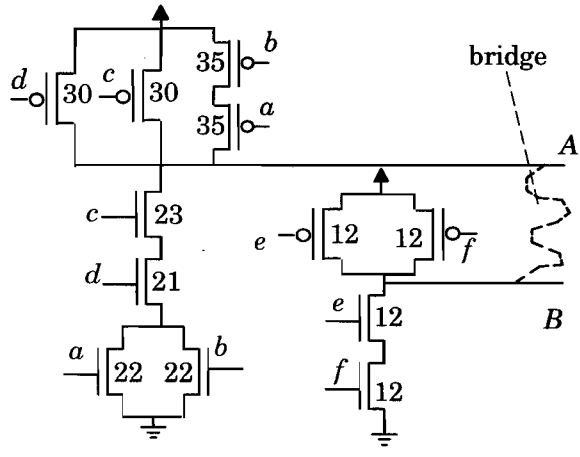


Figure 4.1 An example of a bridge.

Table 4.1 Behavior of the bridge and its impact on fanout cells

inputs		fault free	bridged output		$x=\overline{dA}$		$y=a+\bar{c}+\bar{B}$	
a b c d	*e f*	*A B*	SPICE(V)	level*	fault free	bridge	fault free	bridge
1 1 1 1	0 1	0 1	0.63	0	1	1	1	1
1 0 1 1	1 0	0 1	1.42	0	1	1	1	1
0 0 1 1	1 1	1 0	2.15	x*	0	0	1	1
1 1 1 1	0 0	0 1	2.47	x*	1	1	1	1
0 1 0 1	1 1	1 0	3.35	1	0	0	1	1
0 0 0 0	1 1	1 0	4.71	1	1	1	1	1

level*: using the method in chapter 3. x*: no faulty value is modeled.

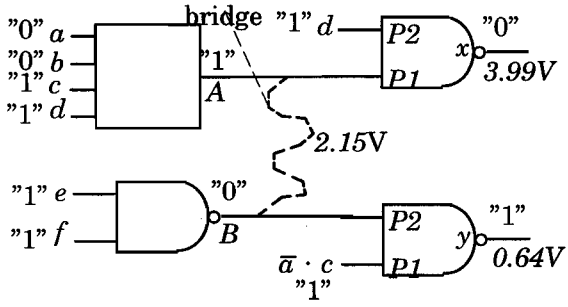


Figure 4.2 Impact of an undefined input on fanout cells.

According to our convention, any output higher than V^1=3.0V is interpreted as "1" and any output lower than V^0=2.0V is interpreted as "0". For outputs higher than 2.0V but lower than 3.0V, we agreed not to decide. Instead the outputs are assumed to be fault free. Now consider the bridged output shown in figure 4.1 to drive two cells x and y as shown in figure 4.2. The outputs of x and y both in the fault free and in the case of the bridge are shown in the last two columns of table 4.1. It can be concluded that using the modeling method in chapter 3, the bridging fault shown in figure 4.2 is not detectable for the inputs listed in table 4.1. However, as indicated in figure 4.2 (quoted 1s and 0s are fault free values), for inputs *abcdef*=001111, the SPICE analysis shows the output at the bridge to be 2.15V. As a result the output of x is 3.99V which can be interpreted as "1". The output of y is 0.64V which can be interpreted as "0". Both outputs contradict the fault free values. The bridge is in fact detectable! Obviously the conclusion that the bridge is not detectable comes from the inability of the method in chapter 3 to propagate the undefined voltage value correctly. Thus the exact solution still relies on the following two issues:

1) how to evaluate the bridged output voltage accurately;

2) how to propagate or interpret an undefined input voltage accurately.

It can be seen from the above example, as some few centivolts difference of the input voltage can cause different outputs, any kind of approximation can easily lead to wrong decisions. Thus to guarantee the correct simulation, circuit–level accuracy must be obtained. Usually as there are many more bridging faults than single stuck–at faults, both the procedures 1) and 2) above must be efficiently solved.

The problem of resolving the undefined input has been notified by many researches. However no adequate solutions have been found for large circuits. Only very recently, some work has been published with the intention to provide efficient solutions. In [14,23], a mixed or multi–level simulation technique is suggested in which the simulator switches from the normal logic simulation to a circuit–level simulation whenever a bridging fault is encountered. The bridge is simulated along its fanout cone until the undefined inputs can be safely interpreted as logic "1" or "0". Then the simulation is switched back to logic level. This method is very accurate. But for lengthy test patterns a large circuit may not be efficiently simulated. For instance, in figure 4.2 the inputs *abcdef* = 100111 and *abcdef*=110111 cause the same conducting circuit. In such a case this method would invoke the expensive circuit simulation twice while this is not necessary. It is also not efficient to evaluate all the bridges connecting two cells having the same combination of cell types, for example, a 2–in–NOR connected to a 3–in–NAND. Some recent improvement [41] of the mixed–level simulation approach uses so called precomputed tables derived by a circuit–level simulator to avoid unnecessary evaluations. The cell(gate) logic threshold voltages are used to propagate an input voltage. However, the precomputed tables that are derived by enumerating all the combinations of a cell library may be too time and memory consuming. It is also not easy to maintain such a huge database. Furthermore the strategy of propagating an undefined input in [41] is still inaccurate. Very recent improvements of the "voting model" [2,38] unfortunately are still approximate in nature and the faulty value propagation procedure is also not accurate.

In line with [38,41], this chapter presents another alternative method for accurate modeling and fast simulation of bridging faults [19].

4.2 Fault simulation using generic–bridge and generic–cell tables

The general strategy of the proposed method is outlined in this section. The circuit chosen for the study is still a CMOS combinational one. Each CMOS circuit is represented by a connection graph as described in chapter 3. The bridging faults are *output to output* type of bridges. Again defects causing

these bridges are fatal defects. That is, the resistance of the bridge is negligible. Furthermore only static analysis is performed.

4.2.1 Evaluation of bridged output

In order to obtain both high accuracy and efficiency, let us examine the design procedure first. In modern CMOS design, it is a common practice that most designs are based on a given cell library. In a specific design, the number of instantiated cells is usually much larger than the size of the library. One type of a cell may be repeatedly used in the design. Thus it is very likely that many bridges may connect the same combination of the cell types in the same manner. These bridges can be represented by one bridge, called the **generic–bridge**. A set of generic–bridges can be derived for all the extracted bridges in a design.

This observation can help to simplify the evaluation task since the evaluation of all the bridges can be restricted to the generic–bridges. Usually the number of generic–bridges is far smaller than the number of all bridges. Each generic–bridge can be evaluated by using a circuit–level simulator, such as SPICE in our case. Then the bridged output is computed with the accuracy of SPICE. Yet a large amount of computational tasks is avoided.

For each generic–bridge, a **generic–bridge–table** is introduced for all the cell inputs that activate this bridge. A generic–bridge–table consists of a set of pairs <*b*, *d*> as its entries. Let T_{bri} be a set denoting all the entries:

$$T_{bri} = \{ < b_1, d_1 >, < b_2, d_2 >, ..., < b_n, d_n > \} \tag{4.1}$$

For each $< b, d > \in T_{bri}$, b is the bridged output voltage value and d is a Boolean expression that represents a set of input vectors activating the bridge and generating a voltage value b at the bridged output. The generic–bridge–table has a property of mutual exclusiveness. That is, for any two $< b_1, d_1 >$ and $< b_2, d_2 >$ in T_{bri} if d_1 is true, then d_2 is not true and vice versa. This is obvious since for one input vector of the two bridged cells, the bridged output cannot have two different voltage values simultaneously. Thus such a table can be viewed as a function

$$F_{bri} = b_1 \cdot d_1 + ... + b_n \cdot d_n \tag{4.2}$$

If d_i is satisfied, the F_{bri} takes one voltage value b_i.

For the bridge shown in figure 4.1, its generic–bridge–table is obtained as:

$$\begin{aligned} F_{bri} = {} & 0.00 \cdot (e \oplus f) \cdot a \cdot b \cdot c \cdot d + 1.42 \cdot (e \oplus f) \cdot (a \oplus b) \cdot c \cdot d \\ & + 2.15 \cdot e \cdot f \cdot \overline{a} \cdot \overline{b} \cdot c \cdot d + 2.45 \cdot \overline{e} \cdot \overline{f} \cdot a \cdot b \cdot c \cdot d \\ & + 2.89 \cdot \overline{e} \cdot \overline{f} \cdot (a \oplus b) \cdot c \cdot d + 3.35 \cdot e \cdot f \cdot (a+b) \cdot (c \oplus d) \\ & + 5.00 \cdot e \cdot f \cdot (\overline{a} \cdot \overline{b} \cdot \overline{c \cdot d} + \overline{c} \cdot \overline{d}) \end{aligned} \tag{4.3}$$

4.2.2 Propagation of undefined inputs

Let us examine how a CMOS cell transfers an input voltage. First, the **logic (switch) threshold voltage** of a cell is defined. For an inverter, the logic threshold voltage is the input voltage value such that the output is equal to the input. The logic threshold voltage of a cell having more than one input can be defined in the similar way. Obviously such a cell may have several different logic threshold voltages. For instance, a NAND with two inputs a and b has a logic threshold voltage 1.89V when a changes while b=1. Vice versa, it has a logic threshold voltage 2.20V when b changes while a=1. When both inputs a and b change simultaneously, it has a logic threshold voltage 2.60V. In the sequel, a logic threshold voltage when only one input changes is classified as **single–input** logic threshold voltage. Otherwise it is classified as **multi–input** logic threshold voltage. For the above example NAND, the logic threshold voltage 1.89V and 2.20V are single–input logic threshold voltages but 2.60V is a multi–input logic threshold voltage. A complex cell may have many logic threshold voltages when certain inputs change simultaneously.

In modern technology, it is known that the CMOS cell has a very high gain around its logic threshold voltage. A small variation at the input yields a very big swing at the output. It is very likely that an input voltage lower than the logic threshold voltage would cause an output large enough to be a logic "1" and vice versa. This implies that most of the undefined input voltages can be interpreted as logic levels just by propagating them one level up along their fanout cones. Without any computation, the fault propagation can be done by comparing the input voltage with the logic threshold voltages of some cell. Obviously it is possible that an input voltage is equal to or very close to the logic threshold voltage. Then the output may still be in the undefined range and cannot be interpreted as a logic value at this stage. This undefined input has to be propagated further before it can be completely resolved. However, more computations are needed. In our experiments on the benchmarks described in chapter 2, such situations hardly occur and add up to only 0.2% of all the cases during the whole fault simulation procedure. Therefore, to obtain fast fault simulation, it is sufficient to propagate a bridging fault just up to the outputs of its immediate fanout cells. The above discussion is summarized as a modeling principle described below for inverted cells (for other types of cell, similar modeling principle can be easily derived).

> **Modeling principle:** Assume a cell has an undefined input. If the input voltage is higher than a logic threshold voltage of the cell, a logic "0" would be read at the output. Vice versa, if the input voltage is lower than the cell logic threshold voltage, a logic "1" would be read. Otherwise it is considered that the fault effect will not appear at the output.

Now let us examine what this modeling principle implies for a specific design. It is usually the case that a specific design uses only a subset of cells from the given cell library. Each of them may have many instantiations. To be consistent with the definition of the generic–bridge, each cell in such a subset of a cell library is called a **generic–cell** of this design. Such a subset of a cell library is called the set of generic–cells of this design. The advantage of the above modeling principle is then obvious. For a specific design, only the logic threshold voltages of each generic–cell are required for the fault propagation. They can be computed accurately by a circuit–level simulator. Again a large amount of computations can be avoided.

To formulate and keep the derived logic threshold voltages of each generic–cell, a **generic–cell–table** is introduced. The generic–cell–table of a generic–cell consists of a set of labeled pairs $< w, O >_l$ as its entries. The label l represents an input terminal or any combination of the input terminals of the generic–cell. For each $< w, O >_l$, w is the value of a logic threshold voltage when the inputs l change simultaneously. O is a Boolean expression representing a set of input vectors such that input terminals l are observable at the output of this generic–cell. If O=1, any change at l also causes a change at output. Let $T_{cell}(l)$ denote all the threshold voltages when terminals l change simultaneously. The set $T_{cell}(l)$ has a property of mutual exclusiveness as well. That is, for any two $< w_1, O_1 >_l$ and $< w_2, O_2 >_l$ in $T_{cell}(l)$, if O_1 is true, then O_2 is not true and vice versa. This is because for an input vector such that terminals l are observable, the generic–cell cannot have two different logic threshold voltages simultaneously when inputs at l change simultaneously.

Let L be a set denoting all the combinations of the input terminals of a generic–cell.

Then the set containing all the entries in the generic–cell–table can be expressed as:

$$T_{cell} = \bigcup_{l \in L} T_{cell}(l) \tag{4.4}$$

It is not difficult to prove that any two entries of a generic–cell–table are also mutual exclusive.

Thus the generic–cell–table can be viewed as the function F_{cell} defined by

$$F_{cell} = \sum_{l \in L} ((w_1 \cdot O_1)_l + (w_2 \cdot O_2)_l + \ldots + (w_m \cdot O_m)_l) \tag{4.5}$$

For specific terminals l, if O_i is satisfied, F_{cell} takes the logic threshold voltage w_i.

For a NAND with two inputs a and b, its generic–cell–table is expressed as

$$F_{bri} = (1.89 \cdot b)_a + (2.20 \cdot a)_b + (2.60)_{ab} \quad (4.6)$$

$(1.89 \cdot b)_a$ indicates that the cell has a logic threshold 1.89V when a changes while b is satisfied. The last one $(2.60)_{ab}$ indicates that the logic threshold voltage is 2.60V when a and b change simultaneously. Obviously in this case any transition at a and b simultaneously is always observable.

4.2.3 Fault simulation strategy

With the introduction of these two concepts, i.e., the generic–bridge–table and the generic–cell–table, bridging faults can be simulated in the procedure described below:

1) For each bridge, find its generic–bridge–table. Evaluate the table according to the applied input pattern and obtain the respective output voltage value.
2) For each fanout cell of the bridged outputs, find its generic–cell–table. For the applied input pattern, evaluate the entries labeled with the inputs that are connected with the bridged outputs. Obtain the respective logic threshold voltage value.
3) Compare the bridged output voltage with the logic threshold voltage and interpret it as logic value at the output.
4) After all the fanout cells are processed, start the normal logic fault simulation from these fanout cells until it can be decided that the bridge is detected.

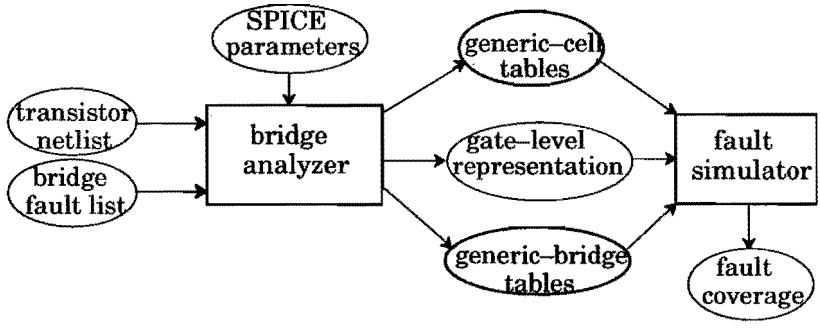


Figure 4.3 An overview of the modeling and simulation system.

Since these two sets of tables can be computed in advance, there is no expensive circuit simulation involved during the fault simulations. Thus the fault simulations can be done solely at logic level with only some costs incurred with the above interpretation procedure. Consequently both

accuracy and efficiency are obtained. Figure 4.3 illustrates the whole system. The inputs of the *bridge Analyzer* (figure 4.3) are a flat representation of the transistor netlists and all possible bridging faults. Both are extracted from the layout of a design using the method in chapter 2. The SPICE simulator is chosen for the computation of circuit responses. Thus SPICE parameters for a specific process are also taken as an input. The method actually employs the same philosophy as the one in the previous chapter. The difference is that a circuit–level simulator is used to evaluate a bridging fault and the logic threshold voltage is used for the fault propagation. Below some details of deriving these two sets of tables are discussed.

4.3 Dynamic derivation of generic–bridge and generic–cell tables

The derivation of the generic–bridge–tables and the generic–cell–tables is performed by analyzing the extracted bridging faults for a specific design instead of enumerating the given cell library. Thus the derivation is dynamic. This is because of the following reasons:

1) The number of the generic–cells in a specific design is usually smaller than the size of a given cell library. Consequently the number of all possible generic–bridges in the design is small. Thus, the task of characterizing both tables for a design is easier.

2) The occurrence of bridging faults depends highly on the layout topology of a specific design. It is very likely that a generic–bridge derived by enumerating the cell library may actually never occur in a design. Such information can only be obtained by analyzing the extracted bridging faults for a specific design.

3) The number of all possible multi–input logic threshold voltages for a set of cells is usually very large. The actual number of multi–input situations depends on how many bridging faults actually connect more than one input of a cell and how a cell is actually connected in a design. Again such information can only be obtained by analyzing the extracted bridging faults for a specific design.

Since both speed and memory are crucial for the simulation, the derivation is performed for each design by analyzing the extracted bridges. In this way, it is guaranteed that each derived generic–bridge–table corresponds to at least a bridge fault that may actually occur in this design. Each derived multi–input logic threshold voltage also corresponds to a case of more than one input being connected together due to possible bridging faults or the actual connection of a cell. Thus the amount of circuit simulations can be

greatly reduced compared to flatly enumerating the cell library. Such derived tables can make the use of some efficient techniques, such as parallel pattern simulation easier as will be shown.

Before the actual computation, the set of generic–cells for a design has to be identified. Let the connection graph corresponding to a cell be called a cell graph. Then it is relatively easy to check if two cells are the same by just checking the isomorphism of two cell graphs. Since at least the power supply and ground nodes are known as the equivalent nodes between these two graphs and the output nodes are the potential equivalent ones, the checking can be done by exhaustively comparing the nodes and edges starting from the output node of each cell graph. Usually the cell graph is relatively small. The comparison of nodes and edges can be started by first checking some cell graph information, such as the number of inputs and the length of the longest path of each cell graph. Eventually such an exhaustive comparison can be done very fast. The checking is repeated until all the instantiated cells in the design are processed. Then all the generic–cells in this design are obtained. After this step, the bridging fault list can be passed to derive the generic–bridge–tables and generic–cell–tables.

4.3.1 Derivation of generic–bridge–table

The derivation procedure of a generic–bridge–table is rather straightforward. For each identified generic–bridge, first all the possible input combinations of these two generic–cells that create a conducting circuit from power supply to ground are enumerated. The respective SPICE format input of each conducting circuit is accumulated in a file. Then, a SPICE call is invoked to compute the bridged output voltages. Upon the completion of the SPICE computation, the results are collected to construct the table. The enumeration of all conducting circuits can be done in the way described in chapter 3. That is, the problem is formulated as finding all the path–connected subgraphs and is solved by running the enumeration algorithm described in Appendix A. The major cost of this procedure is obviously the execution of SPICE. To speed up, the following techniques are used.

The first technique makes use of the fact that if the bridged output voltage is very close to the potential of power supply or ground, it can certainly be interpreted as a logic value. For example, for a typical 5.0V CMOS technology, an input above 4V, which is usually defined as the lowest "hard" logic "1" value, or below 1V, which is usually defined as the highest "hard" logic "0" value, can be definitely interpreted as "1" or "0" respectively. Obviously the approximation method presented in chapter 3 is a good option to analyze those situations. As has been discussed earlier, the output voltage of a

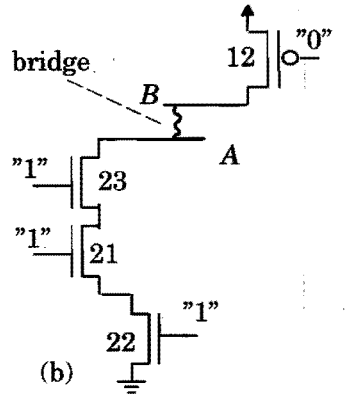
conducting circuit can be predicted by just computing the equivalent beta ratio β of the conducting circuit. Here the difference is that, to set a certain safety margin, two ratios β^0_{hard} and β^1_{hard} corresponding to the highest hard logic "0" and lowest hard logic "1" voltages for a technology are used for the comparison. Such estimation appears to be accurate enough. The entries 0.00V and 5.00V in eq(4.2) have actually been computed by this technique.

	a	b	c	d	e	f
input1	0	1	1	1	0	1
input2	0	1	1	1	1	0
input3	1	0	1	1	0	1
input4	1	0	1	1	1	0

(a)

$< 1.42V,\ (e \oplus f) \cdot (a \oplus b) \cdot c \cdot d >$

(c)



(b)

Figure 4.4 Illustration of the structural equivalence.

The second reduction technique is based on equivalent structures. For a bridge, many conducting circuits activated by a different combination of input excitations have the same structure. Consequently, the bridged output for these different excitations is the same. The conducting circuits are then said to be "structurally equivalent" for these inputs. For instance, with the bridge in figure 4.1, the four different input combinations shown in figure 4.4(a) cause the same conducting circuit as shown in figure 4.4(b) with the bridged output being 1.42V. For those structurally equivalent conducting circuits, there is no need to repeat the SPICE simulation. In the course of analyzing a bridging fault, all the conducting circuits for which the output voltages are already obtained by simulation, are kept in a temporary set. During the enumeration of conducting circuits, if a new conducting circuit is found to be equivalent to a one already in the temporary set, its SPICE format input would not be generated and only its input condition is merged with the corresponding one in the temporary set. To compare two conducting circuits, each conducting circuit can be viewed as a subgraph of the cell graphs. Thus checking whether two conducting circuits are equivalent can be done by comparing the two subgraphs representing the two conducting circuits. Since the subgraph of a cell graph is small, such a comparison process can be done very fast. The result shown in figure 4.4(c) for the example in figure 4.4 will appear in the final generic–bridge–table.

It will be shown by experimental results that above two techniques are every effective.

4.3.2 Derivation of generic–cell–table

A generic–cell–table is constructed in two steps. In the first step, the single–input logic threshold voltages of each generic–cell are derived. The derivation procedure is straightforward. Depending on how it is driven, a generic–cell may have many different logic threshold voltages when one input changes. For a specific input terminal, each cell configuration that results in a different logic threshold voltage when this input changes corresponds to a conducting circuit from power supply to ground comprising of both N–type and P–type transistors driven by this input. For each input terminal, after all such conducting circuits are enumerated and their respective SPICE input formats are accumulated in a file, a SPICE call is invoked to compute the logic threshold voltages. Upon the completion, the results are collected to construct the table. This procedure is repeated for each input of every generic–cell. To enumerate all the different conducting circuits that may lead to different logic threshold voltages, the problem can also be formulated as finding all the path–connected subgraphs as described in Appendix A. The enumeration algorithm in Appendix A can be used.

It seems that other methods did not pay enough attention to the phenomenon that a cell may have many different logic threshold voltages when a single input changes. It is worthwhile to show the effect of this phenomenon on the fault propagation. Figure 4.5(a) shows a generic–cell which has three different logic threshold voltages when input a changes. Their values are listed in table 4.2. Assume the input voltage of a is 2.15V, then it can be interpreted as an "0"(in case the threshold is 2.08V), undefined (in case the threshold is 2.15V) or "1" (in case the threshold is 2.17V) at the output depending on the values of other inputs. Some complex cell may have up to 7 different logic threshold voltages when a single input changes. Thus those effects cannot be ignored.

In a second step, the multi–input logic threshold voltages of each generic–cell are derived. Before the derivation procedure is described, let us examine in which situation the multi–input logic threshold voltages are needed. This is illustrated by an example shown in figure 4.5. Figure 4.5(b) shows a possible use of the cell in figure 4.5(a) in an actual design. It can be seen that one signal can drive two inputs (a and b in the original cell) simultaneously. Assume that a bridge between a and c in figure 4.5(b) occurs. Then one signal can drive three inputs (a, b and c in the original cell) simultaneously. Table 4.2 also lists the logic threshold voltages for those two situations. The table shows clearly that ignoring the dependencies between various inputs can be very deceptive.

Thus it is necessary to know the multi–input threshold voltages in order to interpret the input correctly.

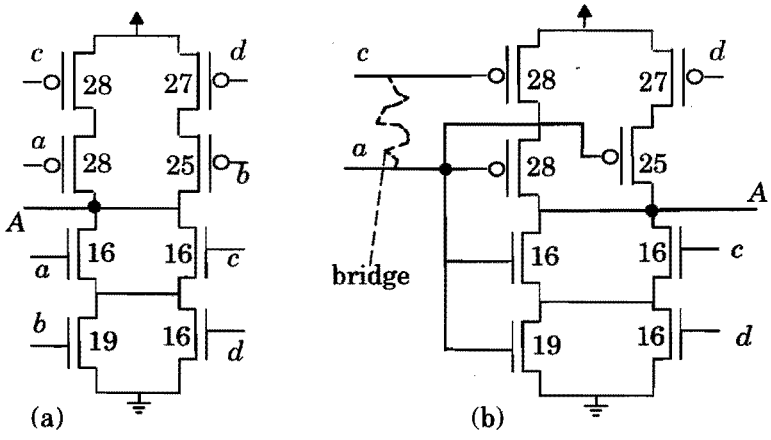


Figure 4.5 (a) A complex cell. (b) illustration of multi–input thresholds.

Table 4.2 Multi–threshold values

input a		input a & b		input a & b & c	
threshold(V)	$b\ c\ d$	threshold(V)	$c\ d$	threshold(V)	d
2.08	1 0 1	1.89	1 0	1.58	1
2.15	1 0 0	2.11	0 1	2.44	0
2.17	0 0 1	2.62	0 0		

The derivation is computed while the bridging faults are analyzed. In the course of the analysis, each multi–input case is individually identified. If more than one input in the fanout cell is bridged or one input signal drives more than one input terminal, then all the possible cell configurations are enumerated. Their logic threshold voltages are computed by SPICE and the generic–cell–table is updated. The procedure is repeated for every bridge. Eventually all the necessary multi–input logic threshold voltages are obtained in the tables.

It should be noted that multi–input logic threshold voltage effects are not considered by the methods in [38,41]. Instead, the single–input logic threshold voltage is used for the fault propagation. This can easily lead to a wrong decision. For instance, for a 2–in–NAND, it has two single–input logic threshold 1.89V and 2.20V. It has a multi–input threshold 2.60V when both inputs change simultaneously. Now assume two inputs are bridged together and the input voltage value is 2.45V. In this situation, using the multi–input threshold voltage 2.60V, the input is propagated as "1" to the output which is consistent with the real value 4.99V. But if the single–input logic threshold

voltage, either 1.86V or 2.20V, is used, then the input is propagated as a "0" to the output which is not correct. Thus our method is more accurate.

4.3.3 Boolean function representations

During the analysis and the derivation of the two sets of tables, the Boolean function of each generic–cell and each table entry involves symbolic Boolean expressions and manipulations. The results need to be stored for the simulations. This seems not an important issue since it is claimed before that the number of generic–bridges and generic–cells for a design is small. However as stated in the previous chapter if this is not properly handled, it may still cost unnecessary memory. To be efficient, ROBDD data structures are used. It is not difficult to observe that the Boolean expression in each entry of a generic–bridge–table is established by a pull–up term of one generic–cell and a pull–down term of another generic–cell. Let each of them be stored separately. Then the canonical property of the ROBDD can result in a very compact representation.

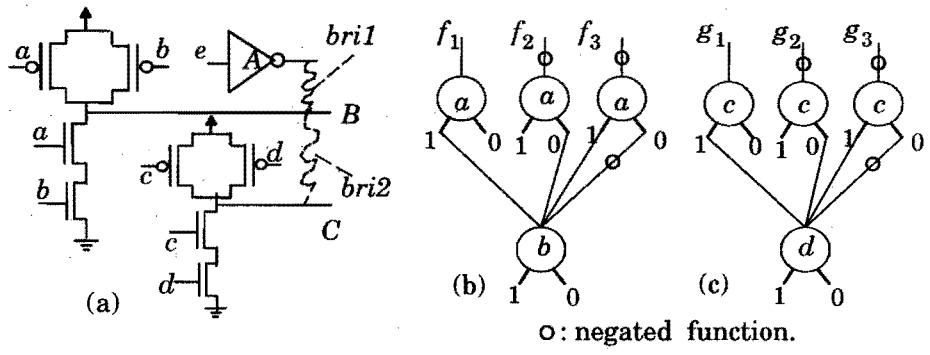


Figure 4.6 Illustration of compact storage.

To illustrate this, figure 4.6(a) shows a generic–cell B involved with two generic–bridges. After analysis, all the pull–down ($f_1=a \cdot b$) and pull–up terms ($f_2=\overline{a} \cdot \overline{b}$ and $f_3=a \cdot \overline{b} + \overline{a} \cdot b$) of B are required to construct the tables. Their ROBDD representations are shown in figure 4.6(b). The generic–bridge–tables are obtained as:

$$F_{bri1} = 1.35 \cdot \overline{e} \cdot f_1 + 1.57 \cdot e \cdot f_3 + 3.39 \cdot e \cdot f_2$$
$$F_{bri2} = 1.45 \cdot g_3 \cdot f_1 + 2.67 \cdot g_2 \cdot f_1 + 1.89 \cdot g_1 \cdot f_3 + 3.45 \cdot g_1 \cdot f_2.$$

Here $g_1=c \cdot d$ is the pull–down term of C and $g_2=\overline{c} \cdot \overline{d}$ and $g_3=c \cdot \overline{d} + \overline{c} \cdot d$ are the pull–up terms of C. Their ROBDD representations are also shown in figure 4.6(c).

During the whole process, the generic–cell B is only needed to be processed once to create f_1, f_2 and f_3. They are shared by both the

generic–bridge–tables. The g_1, g_2 and g_3 are also created once. f_1 and g_1 are also shared inside F_{bri2}. Thus in theory, the upper bound of the memory requirement for all the tables is the number of the different pull–up and pull–down terms of all the generic–cells in a design. Consequently the memory required grows linearly with the number of generic–bridges and generic–cells.

4.4 Fault simulation

This section examines how the generic–bridge–tables and generic–cell–tables are used to perform the fast fault simulations. It will be shown that although the evaluation procedure is different, the fault simulation works just like with any other normal logic fault simulator. Thus any efficient technique can be applied. To show the advantage of using these two sets of tables, the PPSFP technique as described in chapter 3 is used.

The fault simulation is conducted on the network graph of a circuit as described in chapter 3. The first two steps of the fault simulation can be executed as in chapter 3. That is, in the first step, the forward traversal, the fault free simulation is carried out for parallel patterns. In the second step, the backward traversal, the observability of each node is determined according to the applied input pattern in parallel as well. Then the detectability of each bridging fault is determined. This procedure is different from the one in chapter 3 in which each bridge is simulated implicitly. Here each bridging fault is explicitly simulated. This is because in most cases, a voltage value at a bridged output can be propagated as a set of different faulty values to different fanout cells. The fanout branches carrying faulty values may reconverge later at some point. That is, regarding the fault propagation, a non–reconvergent node may behave like a reconvergent node. Therefore each bridging fault should be simulated explicitly from the fanout cells up to the primary outputs or up to a point where its detectability can be determined. Below the fault propagation to fanout cells of the bridged outputs is derived symbolically in order to show the parallel techniques.

The basic operation is the derivation of the faulty Boolean function for each fanout cell of the bridged outputs. As it is discussed in chapter 3, the faulty Boolean function can be constructed by the faulty–on set and the faulty–off set of each cell. This is also the case for a fanout cell of the bridged outputs. For the ease of discussion, a bridging fault between B and C and one of their fanout cells A as depicted in figure 4.7 is used for illustration. Let F_A, F_B and F_C be the fault free functions of these three cells respectively. In the fault free situation, A can be viewed as a function of inputs in I and K. In the presence

of the bridge, A becomes a function of not only the inputs in I and K but also inputs in J. Let the faulty–on and faulty–off set of A are obtained as f_A^1 and f_A^0 respectively. Then applying theorem 3.1, the faulty Boolean function of A is specified as:

$$\tilde{F}_A = F_A \cdot \overline{f_A^0} + f_A^1$$

Below it will be shown how the faulty–on and the faulty–off set of a fanout cell are derived from the generic–bridge and generic–cell tables.

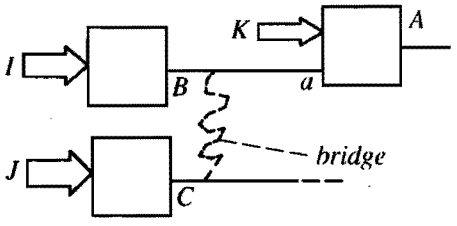


Figure 4.7 Illustration of a bridging fault propagation procedure.

For the bridge shown in figure 4.7, first its generic–bridge–table and the local cell input ordering of the bridge are found. Let all entries of the table be represented by a set T_{bri}. For the fanout cell A in figure 4.7, its generic–cell–table and local cell input ordering are found. Let all the entries labeled with a be represented by a set $T_{cell}(a)$.

According to the definition, the generic–bridge–table T_{bri} can be partitioned into two parts T_{bri}^0 and T_{bri}^1.

$$T_{bri}^0 = \{ < b,d > \in T_{bri} \mid d \Rightarrow \overline{F}_B \wedge d \Rightarrow F_C \} \tag{4.7}$$

$$T_{bri}^1 = \{ < b,d > \in T_{bri} \mid d \Rightarrow F_B \wedge d \Rightarrow \overline{F}_C \} \tag{4.8}$$

For each $< b,d > \in T_{bri}^0$, if any input vector satisfies d, F_B= 0 and F_C = 1.
For each $< b,d > \in T_{bri}^1$, if any input vector satisfies d, F_B=1 and F_C=0.

For any $< b,d > \in T_{bri}^0$, it is known that the fault free value of B is "0" (a=0). In the presence of the bridge, if a is observable at the output of A, to have a faulty value at the output of A, obviously the input voltage at a should be higher than the logic threshold voltage of A when a changes. Let the Boolean expression representing all the input vectors that generate the bridged output higher than a value w be expressed as:

$$C^0(w) = \sum_{<b,d> \in T_{bri}^0(w) \mid b > w} d \tag{4.9}$$

Then for any $< w, O > \in T_{cell}(a)$, a is observable if O is satisfied. Thus a faulty–off behavior is caused at A if any input satisfies $C^0(w) \cdot O$.

By complementary reasoning, let the Boolean expression representing all the input vectors that generate the bridged output lower than a value w be expressed as:

$$C^1(w) = \sum_{<b,d> \in T^1_{bri}(w) \mid b<w} d \tag{4.10}$$

Then for any $< w, O > \in T_{cell}(a)$, if any input satisfies $C^1(w) \cdot O$, a faulty–on behavior is caused at A.

Consider A has more than one logic threshold voltage when a changes, then the final faulty–on and faulty–off sets of A are obtained as:

$$f^0_A = \sum_{<w,O> \in T_{cell}(a)} C^0(w) \cdot O \tag{4.11}$$

$$f^1_A = \sum_{<w,O> \in T_{cell}(a)} C^1(w) \cdot O \tag{4.12}$$

That is, if any input satisfies eq.(4.11), then the output A has a faulty value "0". Vice versa any input satisfying eq.(4.12) introduces a faulty "1" at the output A. Therefore, according to theorem 3.1, the faulty behavior of A is characterized as

$$\tilde{F}_A = F_A \cdot \overline{f^0_A} + f^1_A \tag{4.13}$$

The eq.(4.13) can be evaluated according to the applied input patterns. If the value of $\tilde{F}_A$ indeed differs from F_A, that is, $\tilde{F}_A \oplus F_A$=1, the output A has a faulty value. For the case of more than one input of a fanout cell is bridged together, the propagation procedure is very similar. After all the fanout cells are processed, the logic fault simulation can be started from the fanout cells carrying faulty values.

It is not difficult to observe that the above formulas can be evaluated for patterns in parallel as well via bit–vector operations. Thus the whole procedure can be done for parallel patterns.

4.5 Experimental results

The whole system is implemented in C on a HP–9000/755 workstation. For experiments, the ISCAS85 benchmark circuits as described in chapter 2 again are used. For the SPICE simulator [40], the level 3 MOS model is used for the analysis. Only the part of output to output bridges from extracting results in chapter 2 are used here.

In table 4.3 the analysis results are summarized. In general, the number of generic–cells in each circuit is far less than the size of the actual cell library. The circuit c6288 having 1848 instantiated cells uses only 7 generic–cells. The actual number of generic–bridges derived from the extracted bridging faults is also far less than the number of extracted bridges. It is even less than the number of combinations of the generic–cells in each design. For instance, c7552 has 51773 bridges but only 309 generic–bridges are derived. The number of two combinations of 31 generic–cells in c7552 is already 465.

Table 4.3 Results of bridging faults analysis

circuit	#total cell	#GC	#bridge	#GB	memory (Kb)	time(s)
c432	152	18	1025	68	34	7.3
c499	284	9	2625	36	17	2.5
c880	236	20	3254	146	74	26.5
c1355	366	10	3421	44	24	6.4
c1908	411	20	4132	111	51	20.9
c2670	604	31	13483	238	141	58.0
c3540	791	29	14499	283	134	48.4
c5315	1288	36	39412	345	238	91.0
c6288	1848	7	14298	24	10	4.7
c7552	1795	31	51773	309	197	83.0

#GC : number of generic–cells; #GB: number of generic–bridges.

Table 4.4 Reduction of SPICE calls

circuit	#conducting circuits			#multi–input thresholds		
	actual	total	reduce%	#actual	enumerate	reduce%
c432	458	1237	62.9%	31	120	74.0%
c499	128	275	53.0%	18	34	47.0%
c880	1276	3310	61.4%	68	157	56.7%
c1355	317	569	44.3%	28	58	51.7%
c1908	845	1861	84.6%	52	178	70.8%
c2670	2414	6420	62.4%	82	274	70.1%
c3540	2173	6788	68.0%	107	246	56.5%
c5315	3589	12223	70.6%	131	446	70.6%
c6288	139	224	38.0%	4	12	66.7%
c7552	3369	10110	66.7%	155	368	57.9%

Table 4.4 shows the effectiveness of using the techniques described in section 4.4. The table shows the total number of conducting circuits caused by the set of generic–bridges in each design. They have to be computed by SPICE to characterize generic–bridge–tables. The actual number of them after using

the reduction techniques in section 4.1 are also shown. On average 65% SPICE computations are bypassed. The dynamic derivation of multi–input logic threshold voltages also bypasses on average about 65% of the SPICE simulations compared to enumerating the set of generic–cells in each design. Consequently the generic–bridge–tables and generic–cell–tables are derived very fast.

The times listed in table 4.3 are the actual CPU times in seconds used for the derivation. The dynamic derivation for a specific circuit instead of enumerating the whole cell library is not only very fast but also requires a small amount of memory. In table 4.3 the total size of both the tables is listed. Only up to 250 kbytes are required for the largest circuit. Both the CPU time and the memory requirement have almost a linear relation with the number

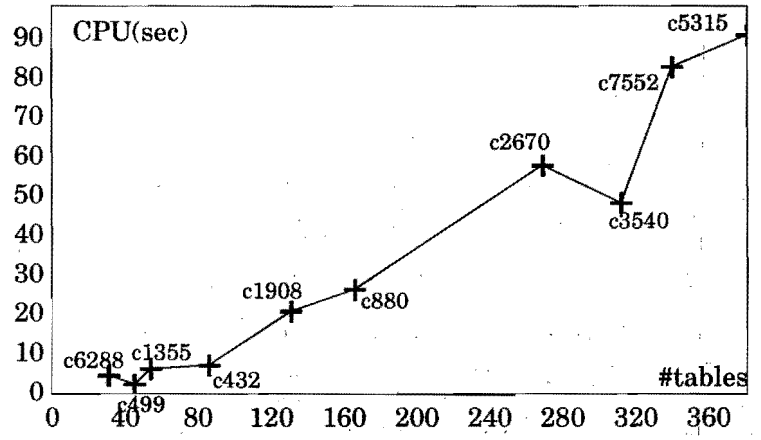


Figure 4.8 Analysis time versus size of tables.

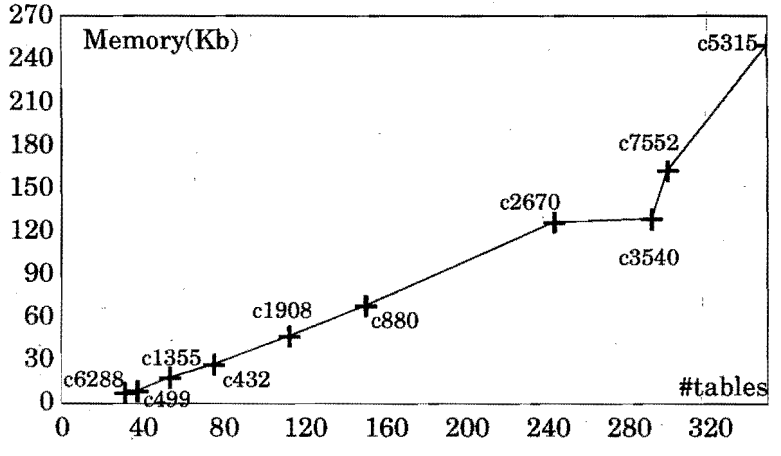


Figure 4.9 Memory requirement versus size of the tables.

of generic–bridges and generic–cells as shown in figure 4.8 and figure 4.9. The fault simulation results are shown in table 4.5. The bridges are simulated for the single stuck–at test pattern sets that are used in chapter 3. The fault simulation is performed in one run for both stuck–at and bridging faults. The fault coverage for bridging faults is the percentage of detected bridging faults divided by the number of all simulated bridging faults. In general, the bridging fault coverages are slightly lower than the respective single stuck–at fault coverages. However, considering that the total number of bridging faults is much higher than the number of single stuck–at faults, the number of undetected bridges is still quite large. The simulation time, however, is very short. The column *errors%* indicates the possible false interpretation percentages during the whole fault simulation. This is, the percentage of the situations where the input is the same as or very close to the cell logic

Table 4.5 Results of PPSFP simulation for SSA test pattern set

circuit	#patterns	SSA%	bridge%	time(s)	error%	bridge%(in chap. 3)
c432	75	99.7	96.9	0.3	0.14	91.0
c499	71	100.0	98.0	0.3	0.04	97.4
c880	95	100.0	99.3	0.8	0.30	92.0
c1355	101	100.0	99.3	0.6	0.25	82.1
c1908	147	100.0	98.4	1.2	0.11	88.1
c2670	160	98.8	98.6	2.5	0.39	86.2
c3540	242	98.4	99.3	6.0	0.25	86.7
c5315	211	100	98.9	7.7	0.29	90.7
c6288	44	99.9	99.8	8.5	0.18	72.5
c7552	318	99.7	99.4	11.3	0.28	84.0

Table 4.6 Results of PPSFP simulation for 21x32 random test patterns

circuit	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
SSA%	99.7	99.5	98.7	99.4	94.5	87.7	98.0	99.9	99.9	92.9
bridge%	97.2	98.9	99.2	99.2	96.4	96.6	99.2	98.9	99.9	98.6
time(s)	0.6	0.6	1.1	0.9	1.8	3.3	7.3	8.8	22.7	12.7
bridge*%	92.3	97.4	92.0	82.1	86.7	84.3	86.8	90.8	73.1	83.2

bridge*% : bridging fault coverage obtained by using the method in chapter 3.

threshold voltage. We choose,

$$|V_{in}-V_{threshold}| \leq 0.02\text{V}$$

V_{in} is the input voltage value and $V_{threshold}$ is the logic threshold voltage. This error is not a substantial problem for this set of benchmarks.

For a comparison, these bridging faults are also modeled and simulated by using the approximate method using equivalent beta ratios (chapter 3). Here only the bridging fault coverages are shown. It can be observed that the bridging fault coverages can be improved up to 20% for some circuits. This indicates that the modeling accuracy indeed has an significant impact on the results of fault simulation.

To verify the random testability for the bridging faults, again 21x32 randomly generated test patterns are simulated for the bridging faults. The results are shown in table 4.6. The bridging fault coverage for the same 21x32 patterns obtained by using the approximate method using equivalent beta ratios (chapter 3) are also shown in the last row. The fault coverages are also improved a lot. This once again shows the impact of the accurate modeling on the fault simulation.

4.6 Conclusions

It is hard to compare with other methods since most of the documentation of those methods do not include the preprocessing time and memory requirement. Further selecting the bridging faults, the test pattern sets, the design approach and the process parameters (SPICE parameters) can make a lot of difference as well. Nevertheless, intuitively this method is much more accurate than other approximation methods. Compared to the methods of using precomputed tables by enumerating the given cell library, our method is also more accurate since the multi–input logic threshold voltages are considered. The introduction of the generic–bridge–table and the generic–cell–table greatly facilitates the use of any efficient simulation technique. In principal they make our fault simulation fast. Furthermore the dynamic derivation by analyzing the extracted bridges for a specific design makes the analysis fast and requires much less memory than competing methods. The technique of bypassing unnecessary SPICE simulations proves to be effective. In addition, this method does not require the cell library information and the input is a flat representation of extracted transistors. The generic–cells can be derived for each specific design. Thus the method is to be used for any design style. Lastly, the idea can be used for bridging faults involving internal nodes. We believe that it is a good way of simulating bridging faults.

This method, however, does have a limitation. If the bridged outputs bear voltage values close to the threshold very frequently, many errors may be induced. The results may not be reliable although our experiments suggest that it is unlikely.

5

Open Fault Modeling and Simulation

5.1 Introduction

The previous two chapters outlined two possible alternatives to perform accurate modeling and fast simulation for bridging faults. In recent years, another alternative of detecting bridging faults by measuring excess quiescent power supply current (I_{ddq}) has attracted a lot of attention [32,33]. However, as one of the disadvantages, I_{ddq} current testing is not effective for open faults. The importance of the open faults should be recognized, however, by the following facts:

1) As the data shown in chapter 2, although the number of open faults is not as large as the number of bridges, the probability of the occurrence of an open fault can be large.

2) The random defects cause opens more likely in one product than in other [51,56]. It is reported experimentally that IC's passed single stuck–at, I_{ddq} or even delay test pattern sets still did not operate correctly. One of the reasons is the presence of opens on the conducting paths [37,50].

On the other hand, most previously proposed methods of modeling and simulating open faults have the shortcoming that both hazard and charge–sharing effects are not completely analyzed. Therefore a method is still demanded to perform accurate modeling and efficient simulation for opens. In following sections, the open fault and its testing problems are first examined in detail. Then another alternative for modeling and simulating open faults is proposed which overcomes the shortcomings of the previously proposed methods. The method proposed in this chapter was previously published in [18].

5.2 Open fault and its testing problems

5.2.1 Open faults

The first mentioning of open faults was in the late 70's [55]. Since then, the single transistor stuck–open fault model is used by most of researchers. The inadequateness of such a model was described in [36]. In this thesis, the open is analyzed at circuit level. That is, assume the spot defect condition as described in chapter 2. Then if an open occurs it causes the tree structure of some node to be split into a number of subtrees. The node can be the gate (e.g., open #3 in figure 5.1), the drain (e.g., open #1 in figure 5.1) and the source (e.g., open #2 in figure 5.1) of a transistor, an output node (e.g., open #4, #5 in figure 5.1) or an input node (e.g., open #6 in figure 5.1) of a cell. The open considered here is assumed to be fatal fault. That is, the capacitive coupling of the open is deemed insignificant.

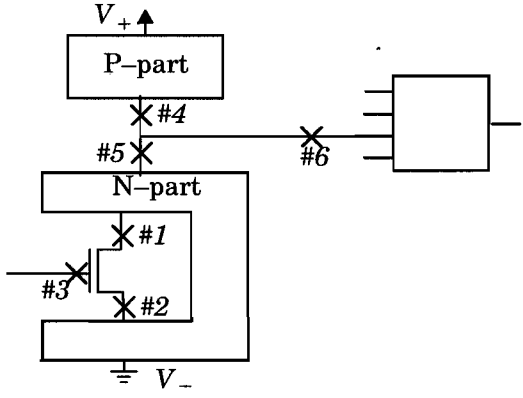


Figure 5.1 Opens in a CMOS circuit.

The physical mechanism of opens and their electrical behavior have been well studied [36,47] by measuring artificial opens introduced by manufacturing into real circuits. It appears that the behavior of the open at the gate of a transistor largely depends on the local topology and is rather sensitive to the gate capacitive signal coupling[36]. Usually either stuck–at behavior or increase of cell propagation delays are observed. The open at an input or an output node of a cell most likely behaves like a stuck–at fault [36,47]. If an open only occurs at the gate of the N–type transistor and shows stuck–at fault behavior (usually stuck–at 0), the N–type transistor behaves like a real transistor stuck–open. In this thesis, such an open is considered as being equivalent to an open at the drain(source) of the respective transistor. If an open occurs at the gate of the P–type transistor and shows stuck–at fault behavior (usually stuck–at 0), the transistor behaves like conducting all the

time. Thus it can be considered as a bridging fault between drain and source of the transistor.

The opens at the drain or source usually prevent the path from conducting under normal clock rates and show the well–known memory behavior [55]. Such kind of opens need a different testing approach. This chapter concentrates on such memory behavior opens and assumes a normal voltage testing environment.

5.2.2 The problem of testing opens

The best method of testing opens is the "two test patterns approach" [55]. Consider an example in figure 5.2 and a two consecutive test patterns (*test1*) as shown in table 5.1 for open #1 in figure 5.2.

Table 5.1 Test patterns for open #1

test1			
	inputs	outputs	
	a b c d e	*A*	faulty output
pat.1	1 0 1 0 1	1	1
pat.2	1 0 1 1 0	0	1
test2			
	inputs	outputs	
	a b c d e	*A*	faulty output
pat.1	1 1 1 0 0	0	0
pat.2	1 0 0 0 1	1	1
pat.3	1 1 0 0 1	0	1

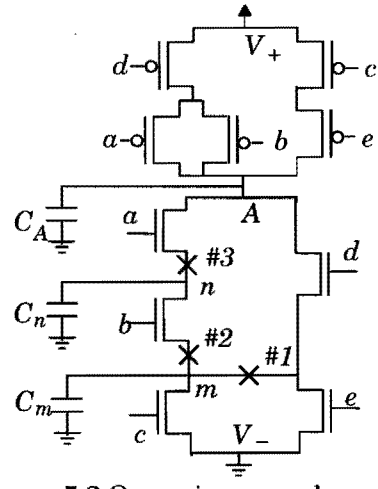


Figure 5.2 Opens in a complex cell.

The first pattern pat.1 sets a pull–up conducting path charging the output to V_+. The second pattern pat.2 intends to set a pull–down conducting path across the broken node. Since the open prevents the pull–down path conducting, A remains in a high impedance state and intends to keep the precharged value. This value contradicts the fault free output "0". Thus the open is considered as detected. The procedure of applying the first pattern pat.1 is usually called the *initialization phase* and applying the second pattern pat.2 is called the *test phase*.

The problems associated with this approach are known as well. The first is hazards effects [42]. For the specifically selected two consecutive test patterns *test1* in table 5.1, during the transition from pat.1 to pat.2, if some delays turn d to be "1" too early or keep e to be "1" too long, a temporal leakage path from A to V_- would be created which may set A to a voltage value too low

to be "1". This problem usually can be avoided by carefully choosing the two consecutive test patterns such that temporal leakage path cannot exist during the transition from initialization phase to test phase. The two consecutive test patterns pat.2 and pat.3 of *test2* in table 5.1 form an example since only one signal b changes during the transition. But a second problem with this approach may occur [5]. That is, after pat.1, nodes n and m possibly have the same potential as $V_{\_}$. If C_n and C_m are comparative with C_A, then in test phase (pat.3), charge–sharing between n, m and A may occur which may still cause the precharged value to be not close enough to "1". Eventually the test is still invalid.

In this thesis, a two consecutive test patterns is said establishing a **robust** test if the test can not be invalidated by any possible delays or charge–sharing effects. In the analysis, this thesis distinguishes between the following tests:

1) non–robust test.

2) robust test under hazard effects.

3) robust test under both hazard and charge–sharing effects.

5.3 General strategy

Most of the previous methods of modeling and simulating opens can be classified as one of the following two approaches. The first approach uses a switch–level model [8,9]. Even with parallel techniques [10,26,44,52], this kind of approach seems to prove inefficient for lengthy test patterns for large circuits. Another approach [5,15,27,30,42] intends to convert the transistor level representation of the circuit into an equivalent gate–level circuit such that each single transistor stuck–open or stuck–short fault can be mapped to corresponding single stuck–at faults at gate–level. The advantage of the approach is that the existing gate–level tools can be directly used. Unfortunately such transformed gate–level circuit is usually rather large. The assumption of single transistor stuck–open fault makes it also impossible to model some opens such as #1 in figure 5.2. Furthermore both of the approaches have the same shortcoming that hazard and charge–sharing effects are not completely considered (except [5]). Thus they are not very robust. For example, the commonly used fault equivalent technique such as in [30] would consider the opens #2 and #3 shown in figure 5.2 as being equivalent. However, the charge–sharing effects may make the two opens behave completely different.

The approach proposed in this chapter follows the same philosophy as developed for bridging faults. Figure 5.3 illustrates the whole strategy. The transistor netlist and capacitance of each node are extracted from the circuit

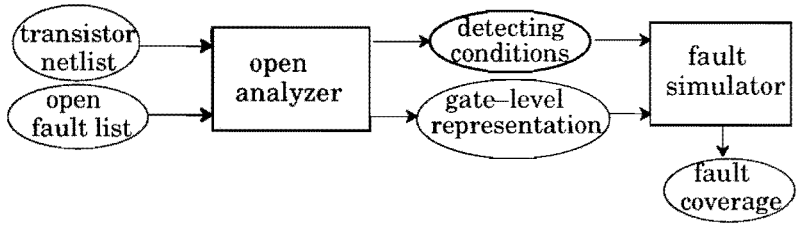


Figure 5.3 Modeling and simulation strategy for opens.

layout and the open fault list is also available. First the transistor network is further abstracted up to gate–level. Then for each open fault, a local circuit analysis is performed by considering both the hazard and charge–sharing effects for this open. As a result of the analysis, its behavior is modeled in terms of a **detecting condition** at logic level. After all the open faults are processed, a set of detecting conditions are obtained at logic level. Then the logic fault simulation can be performed by just manipulating these detecting conditions. Since there is no circuit level computations involved in the course of fault simulation, both accuracy and efficiency are obtained and yet more types of opens can be handled. The following sections demonstrate how the detecting conditions can be derived and stored efficiently for an arbitrary open. It is also shown how a detecting condition can be used by a logic fault simulator.

5.4 Derivation of detecting conditions

5.4.1 Non–robust test and robust test under hazard effects

Let the CMOS network be represented by a undirected graph as described in previous chapters. In the sequel, only opens in the N–part of a cell are discussed. But the similar procedure can be applied to any open in the P–part of a cell.

Consider a general open in N–part of a cell as illustrated in figure 5.4. The fault free function F of the cell can be easily constructed by either the pull–up terms or the pull–down terms as in previous chapters:

$$F = \sum_{s \in P_{AV_+}} T_s = \overline{\sum_{s \in P_{AV_-}} T_s} \tag{5.1}$$

If the open occurs, the basic phenomena caused by such an open is that the node n is split into two parts. Some of the paths from output node A to ground V_- do not exist anymore. Let P_{AV_-} and $\tilde{P}_{AV_-}$ denote all pull–down paths in the fault free situation and in case of the open respectively. Then all the missing

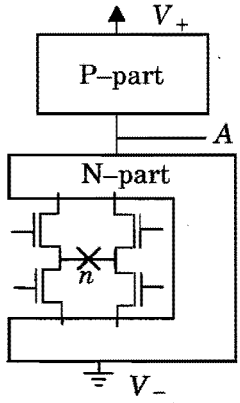


Figure 5.4 Illustration of an arbitrary open.

paths denoted by $M_{AV\_}$ are given by the difference of $P_{AV\_}$ and $\tilde{P}_{AV\_}$. That is,

$$M_{AV\_} = P_{AV\_} - \tilde{P}_{AV\_}.$$

If $\tilde{P}_{AV\_} = \emptyset$, then all the pull–down paths are disconnected from the output to ground. Such an open very likely behaves like a single stuck–at fault at A. Thus it is referred to as **single stuck–at open** here. Open #4 and #5 in figure 5.1 are such examples. There is no need to derive the detecting conditions for a single stuck–at open. Each $s \in M_{AV\_}$ consists of a path from A to one split part of n and a path from another split part of n to ground. In the presence of the open, a Boolean expression is defined for all the remaining pull–down paths as follows:

$$\tilde{F} = \sum_{s \in \tilde{P}_{AV\_}} T_s \tag{5.2}$$

Any input vector satisfying eq.(5.2) establishes a pull–down conducting paths in spite of the open. Then, for each missing path $s \in M_{AV\_}$, a Boolean expression Q_s is defined as

$$Q_s = \tilde{F}_{T_s=1} \tag{5.3}$$

Eq.(5.3) is derived from eq.(5.2). Eq.(5.3) is the Boolean expression of $\tilde{F}$ under the constraint $T_s = 1$. Then for each $s \in M_{AV\_}$, following expression can be obtained

$$X_s = T_s \cdot \overline{Q}_s \tag{5.4}$$

with Q_s derived from eq.(5.3). Obviously if X_s is satisfied, only the missing path s is supposed to conduct but no other paths. Due to the open preventing such a conducting path, the output remains in the high impedance state. For "non–robust test", an open is considered as detected if any two consecutive

test patterns are able to initialize A to "1" and set the output into the high impedance state in case of the open is present. Thus any input satisfying X_s is a test pattern. The complete test patterns for this open can be expressed as

$$\sum_{s \in M_{AV\_}} X_s$$

Since any input satisfying the fault free function F establishes at least a pull–up conducting path and thus is an initialization pattern. Consequently for the "non–robust test", the detecting condition is represented as:

$$d = \sum_{s \in M_{AV\_}} X_s^t \cdot F^{t-1} \tag{5.5}$$

X_s is derived in eq.(5.4). The superscript $t-1$ denotes the initialization interval and t denotes the testing interval. Any two consecutive test patterns satisfying eq.(5.5) establish a non–robust test of the open.

Now let us derive the detecting condition under hazard effects. The main principle of preventing hazard effects is that the two consecutive test patterns should be chosen in such a way that during the transition from the initialization phase to test phase, there is no possible temporal leakage path. Assume during the test phase, missing path $s \in M_{AV\_}$ is activated. That is, $X_s^t = 1$. According to eq.(5.4), this implies that $T_s^t = 1$ and $Q_s^t = 0$. Since in the initialization phase $F^{t-1} = 1$, no pull–down path is supposed to conduct. That is, $T_s^{t-1} = 0$. Now assume that the expression Q_s is not stable during this transition. That is, $Q_s^{t-1} = 1$ in the initialization phase. If any possible delays that turn T_s from "0" to be "1" too early or keep Q_s at "1" too long, then $(T_s \cdot Q_s)^{t-1\pm\varepsilon} = 1$ would hold for a short period ε. According to eq.(5.3),

$$T_s^{t-1\pm\varepsilon} \cdot (\tilde{F}_{T_s=1})^{t-1\pm\varepsilon} = 1$$

would be obtained which implies $\tilde{F}^{t-1\pm\varepsilon} = 1$. That is, some pull–down conducting paths exist temporally. This temporal path may drain the precharged node not high enough to be "1" and the test may be invalid. Thus in order to eliminate such situation, the Q_s should be stable during the transition. That is, $(\overline{Q}_s)^{t-1}$ should be satisfied as well. The same reasoning applies to other missing paths as well. Finally the detecting condition of robust test under hazard effects is obtained as:

$$d = \sum_{s \in M_{AV\_}} X_s^t \cdot \overline{Q}_s^{t-1} \cdot F^{t-1} \tag{5.6}$$

X_s is derived from eq.(5.4) and Q_s is from eq.(5.3). Any two consecutive test patterns satisfying eq.(5.6) establish a robust test under hazard effects for this open.

5.4.2 Robust test under both hazard and charge–sharing effects

To take both the hazard and charge–sharing effects into account, let us first look at how to analyze the charge–sharing effects. In general, it is difficult even with a circuit simulator to analyze and model the charge–sharing effects accurately since they strongly depend on the topology of a design [29]. In this thesis, a method similar to [29] is used to estimate the voltage level after charge–sharing. It is stated below.

Let $V_1, V_2, ..., V_i$ and $C_1, C_2, ..., C_i$ be the voltage levels and capacitances of some nodes that are connected through conducting transistors respectively. No V_+ or V_- is connected to the source or drain of any transistor. Then after charge–sharing all nodes have the same voltage level V as:

$$V = \frac{C_1V_1 + C_2V_2 + ... + C_iV_i}{C_1 + C_2 + ... + C_i} \tag{5.7}$$

Table 5.2 Test patterns for open #1

test2			
	inputs	outputs	
	a b c d e	*A*	faulty output
pat.1	1 1 1 0 0	0	0
pat.2	1 0 0 0 1	1	1
pat.3	1 1 0 0 1	0	1

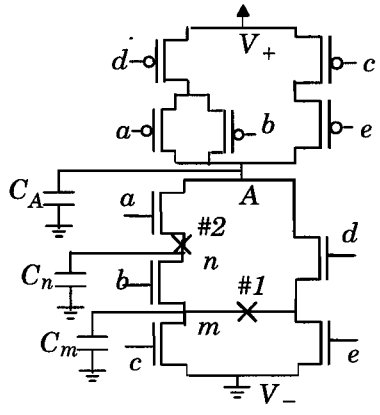


Figure 5.5 Opens in a complex cell.

Before going into detail, for each missing path $s \in M_{AV}$, the potential charge–sharing part of s is identified first. It is the part between the output node A and the split node n and is denoted as $\tilde{s}$. To illustrate this estimation method, consider the open #1 in figure 5.2 and the test patterns *test2* in table 5.1. Both are reproduced in figure 5.5 and table 5.2 respectively. In the test phase, the missing path $a \cdot b \cdot e$ is activated. The potential charge sharing part of this missing path is identified as $a \cdot b$. Applying the above method, A has a voltage level given by

$$V'_A = \frac{C_AV_A + C_nV_n + C_mV_m}{C_A + C_n + C_m} \tag{5.8}$$

after charge sharing. From eq.(5.8), it would be expected that if $V'_A > V^1$, where V^1 is the lowest logic "1" voltage level as defined in chapter 3, then the output still has a voltage level high enough to be "1". The test would be valid. Reasoning in such a way, two conditions implying charge–sharing not to invalid the test patterns can be derived from eq.(5.8). These two conditions are:

1) Assume before the test phase the internal nodes n and m are charged to or very close to the potential V_+. That is $V_n = V_m = V_A = V_+$. According to eq.(5.8), in the test phase the output would have a voltage value still high enough to be "1" in spite of charge–sharing.

2) Assume that the potential of n and m are very low (close to V_-). But if C_n and C_m are sufficiently smaller than C_A such that after charge–sharing V'_A is still higher than V^1 . Here according to eq.(5.8), V'_A is estimated as

$$V'_A = \frac{C_A \times V_+ + (C_n + C_m) \times 0}{C_A + C_n + C_m} \tag{5.9}$$

Then the output is still high enough to be "1" even with charge–sharing.

One possible way of satisfying the first condition is to select such an initialization pattern that not only a pull–up conducting path exist but also the potential charge–sharing part of the activated missing path should conduct as well. In such a way, the internal nodes of the potential charge–sharing part are very possibly charged to the level of the output node before the test phase. For example, the two consecutive inputs $(abcde)^{t-1} = 11000$ and $(abcde)^t = 11001$ is such test patterns for the open #1 in figure 5.5. However, such a restriction on a potential charge–sharing path may limit the number of possible initialization patterns to be selected. For instance, for the open #1 in figure 5.5, there is only one solution.

The method proposed in [20] does not make the distinction between the above two situations. Only the worst case is considered in the simulation in which the capacitance of each node is assumed to be of size comparative with the one of the output node. As it is pointed out, the disadvantage of such a consideration is very clear. In case the second condition described above is satisfied, this approach may unnecessarily restrict the number of initialization patterns that can be selected. Eventually a robust testable open fault under charge–sharing effects may be considered as untestable.

The method proposed in this thesis uses the principle that the restriction on the potential charge–sharing part is only applied when it is necessary such that there are more choices for selecting initialization patterns. To apply this principle, an analysis is performed in the course of analyzing each open fault

so that these two different situations can be distinguished. Such an analysis is straightforward. For each missing path $s \in M_{AV\_}$, its potential charge–sharing part $\tilde{s}$ is analyzed by using the estimation method described above. That is, presumably A is charged to V_+ and all other nodes in $\tilde{s}$ are precharged to V_-. Then the estimation method is applied to evaluate the potential charge–sharing part $\tilde{s}$. If it appears that the capacitances of internal nodes are small enough such that even after charge–sharing $V'_A > V^1$ is true, then path s is put in a set $M^{nc}_{AV\_}$. Otherwise the missing path s is put in a set $M^{c}_{AV\_}$. After all the missing paths are analyzed, then $M^{nc}_{AV\_}$ contains all those paths not rendering any extra action. But $M^{c}_{AV\_}$ contains all the paths that need the restriction on the potential charge–sharing part. That is,

$$M^{c}_{AV\_} = \{s \in M_{AV\_} \mid V'_A > V^1, \text{ if } X_s = 1\} \tag{5.10}$$

and

$$M^{nc}_{AV\_} = \{s \in M_{AV\_} \mid V'_A \leq V^1, \text{ if } X_s = 1\} \tag{5.11}$$

Obviously $M_{AV\_} = M^{c}_{AV\_} \cup M^{nc}_{AV\_}$.

To set the restriction on the potential charge–sharing part of each missing path $s \in M_{AV\_}$ during the initialization phase, it simply implies that $T^{t-1}_{\tilde{s}}$ should be satisfied.

From above discussion, the detecting condition of a robust test under both hazard and charge–sharing effects can be derived as:

$$d = \Big(\sum_{s \in M^{nc}_{AV\_}} X^t_s \cdot \overline{Q}^{t-1}_s + \sum_{s \in M^{c}_{AV\_}} X^t_s \cdot (\overline{Q}_s \cdot T_{\tilde{s}})^{t-1} \Big) \cdot F^{t-1} \tag{5.12}$$

Any two consecutive test patterns satisfying eq.(5.12) establish a robust test under both hazard and charge–sharing effects.

5.4.3 Representation of detecting conditions

The idea of deriving the detecting conditions for all the opens before the fault simulations is simple and straightforward. The key issue of this technique is still the representation of the detecting conditions and storage for a large circuit. For the open illustrated in the previous sections, it is not difficult to observe that all the detecting conditions are in fact constructed from three sets, $\tilde{P}_{AV\_}$, $M^{c}_{AV\_}$ and $M^{nc}_{AV\_}$. Thus only these three sets need to be stored for an open. The detecting conditions can be constructed from them in the course of fault simulation. Again benefiting from the strong canonical feature of the ROBDD, efficient representation of these sets can be obtained. To illustrate, after analysis, the required path sets for #1 and #2 shown in figure 5.5 are

obtained as

$$\tilde{P}_{AV\_}(\#1) = \{ f_1, f_3 \}, M^{nc}_{AV\_}(\#1) = \{ f_4 \} \text{ and } M^{c}_{AV}(\#1) = \{ f_2 \}$$

$$\tilde{P}_{AV\_}(\#2) = \{ f_3, f_4 \}, M^{nc}_{AV\_}(\#2) = \{ f_1, f_2 \} \text{ and } M^{c}_{AV}(\#2) = \emptyset$$

respectively. Here $f_1 = a \cdot b \cdot c$, $f_2 = a \cdot b \cdot e$, $f_3 = e \cdot d$ and $f_4 = c \cdot d$ are pull–down terms of A. They are created only once and are shared by these required sets listed above. Their ROBDD representations are shown in figure 5.6. The shaded nodes a and b indicates that they are the potential charge–sharing part of f_2. It can be seen that each required set contains some of the pull–up or pull–down terms of the cell, the upper bound of the memory requirement for an open is the number of all the different pull–up and pull–down terms of the cell. Thus in theory the memory requirement is linear to the number of opens.

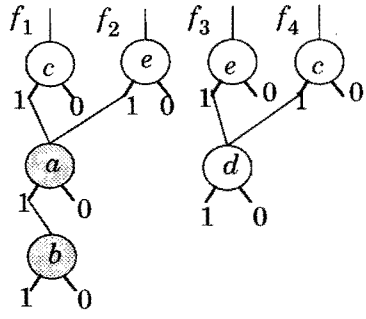


Figure 5.6 Illustration of compact storage.

5.5 Fault simulation for opens

The detecting conditions derived in previous section considering both the hazard and charge–sharing effects for an open can be easily used by any logic fault simulator. For illustration, here the PPSFP algorithm is adapted to simulate opens.

The fault simulation is performed on the network graph (see chapter 3). The preprocessing of obtaining the detecting conditions is very simple. For each open, the missing paths M and remaining paths $\tilde{P}$ can be easily collected by using a depth–first search routine. The missing paths M are further partitioned into a part M^{nc} for which no restriction on the potential charge–sharing path is required and its complement M^{c}. All the single stuck–at opens can be also identified in the meantime.

The main operations of the PPSFP are also two traversals as described in chapter 3. That is, in the forward traversal, the fault free simulations are performed for applied patterns in parallel. In the backward traversal, the

observability of each node is evaluated for applied input patterns just as in chapter 3. In the meantime, the detectability of each open is determined. Since there is only one cell involved, the detectability of each open can be determined as follows. For each open, assume the cell a is affected by this open. The detecting condition d can be constructed from its respective remaining path set $\tilde{P}$ and missing path sets M^{nc} and M^{c} in the same way as deriving eq.(5.5), (5.6) and (5.12). Then its detectability can be determined by evaluating

$$D_{open} = O_a \cdot d \tag{5.13}$$

for applied input pattern. O_a is the global observability of a (see chapter 3). If $D_{open} = 1$, the open is detected. Obviously the evaluation of eq.(5.13) can be performed in parallel for applied patterns via bit–vector operations.

5.6 Experimental results

The above modeling and simulation system was implemented in C on a HP–9000/700 workstation. The ISCAS85 benchmark circuits are again used for experiments. Open faults are assumed on all possible paths of each cell. For the purpose of just verifying the effectiveness of this technique, each node is assumed to have the same capacitance.

Table 5.3 Results of extracting detecting conditions

circuits	#opens	#SSA opens	time(sec)	memory(Kb)	memory overhead
c432	238	36	0.1	17.7	40%
c499	520	104	0.1	31.9	9.4%
c880	516	57	0.17	28.7	60%
c1355	850	3	0.27	37.8	45%
c1908	632	117	0.1	44.8	21%
c2670	1110	103	1.5	71.7	56%
c3540	1453	168	0.4	88.0	5.7%
c5315	2325	272	1.0	149.8	54%
c6288	3792	16	2.3	177.7	53%
c7552	3745	501	1.4	200.0	46%

Table 5.3 summarizes some extraction results. Among the analyzed opens, on average about 9% opens are single stuck–at (SSA) opens. The analysis and collection of the missing and remaining paths are both very fast. Compared to the fault simulation times, they are almost negligible. The amount of memory required to represent the fault free circuit is also listed in the table

(the column of *memory*). The *memory overhead* of representing the path sets by ROBDDs are shown in the last column. They are the percentages of the extra memory over the total memory needed just for fault free representations. The memory overhead largely depends on the number of different types of cells and the number of different types of opens. The maximum memory overhead is up to 60% for the circuit c880. On average, 39% more memory compared to the fault free logic representations is required. The total memory requirement is not a substantial problem for this set of benchmarks.

Table 5.4 Results of PPSFP simulation

	SSA test pattern set						1000 random patterns			
circuit	#pat.	SSA%	NR%	R1%	R2%	time(s)	SSA%	NR%	R1%	R2%
c432	75	99.7	71.8	65.9	54.0	0.54	99.7	90.6	89.6	87.1
c499	71	100.0	87.5	75.0	75.0	0.78	100.0	94.4	86.8	86.8
c880	95	100.0	89.1	79.1	71.5	1.40	99.2	93.3	88.9	86.3
c1355	101	100.0	81.2	78.9	78.4	1.80	99.7	82.3	80.3	80.3
c1908	147	100.0	88.4	82.3	81.0	3.10	97.7	87.4	81.2	81.2
c2670	160	98.8	75.6	66.4	60.7	7.50	87.7	69.3	67.1	64.5
c3540	242	98.4	74.1	65.8	60.4	21.30	97.8	79.0	73.6	68.4
c5315	211	100.0	91.3	85.1	81.7	27.40	100.0	94.0	91.4	89.4
c6288	44	99.9	83.1	79.3	67.5	9.30	99.9	86.3	85.8	85.3
c7552	318	99.7	89.9	84.7	82.0	49.10	93.6	89.3	86.8	84.5

Table 5.4 shows the fault simulation results. The opens are simulated for both single stuck–at test pattern sets and 1000 random test patterns as in chapter 3 and 4. The CPU time of simulating opens is almost the same as simulating the single stuck–at faults. The fault coverage of non–robust test (denoted as *NR%*), the robust test under hazard (denoted as *R1%*) and the robust test under both hazard and charge–sharing effects (denoted *R2%*) are evaluated in one pass.

As already expected, the test pattern sets having very good coverage for single stuck–at faults, in general, have rather poor coverage for open faults. Not more than 95% coverage can be achieved even for a non–robust test. The robustness of the test pattern sets is even more poor. The lowest coverage is only 54%. It is interesting to notice that though the difference between the coverage of non–robust test and the coverage of robust test is rather large, the difference between the coverage of robust test under hazard effects and the one of robust test under both hazard and charge–sharing effects is small. This may indicate that the hazard effects should be considered for test pattern

generations but charge–sharing effects is not a severe problem for such type of design.

For 1000 random test patterns, as listed in Table 5.4, the fault coverages in general are greatly improved both for non–robust and for robust tests. The random testability of the benchmark circuits for opens is better than the one for bridging faults. Figure 5.7 shows the simulation time versus circuit size. There is no clear relation observed. It largely depends on the number of opens and number of different types of cells in a design. It should be noted that no accelerating techniques such as in [3] are used for the fault simulations. It is expected that the simulation can be much faster if they are applied. Here, for the purpose of verifying the developed method, it is important to notice that the difference of the simulation times between single stuck–at faults and opens is very small and yet the total simulation is done very fast.

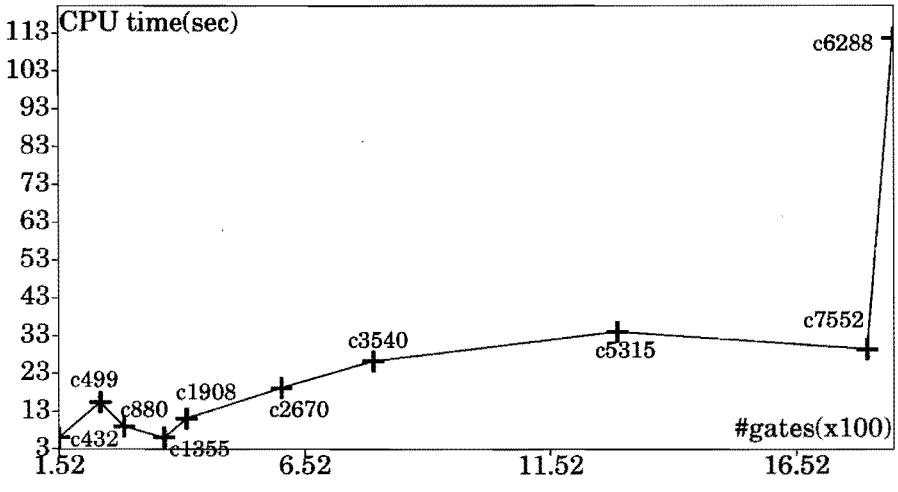


Figure 5.7 Simulation time of 1000 random patterns vs. size of the circuit.

5.7 Conclusions

The electrical behavior of open faults is very complex which makes it difficult to perform accurate modeling and fast simulation. The method proposed in this chapter is based on several assumptions as most other approaches are. But a different strategy is used here. The advantage is obvious; both the hazard and charge–sharing effects are modeled for any path open faults and yet fast logic fault simulations can be achieved. The use of ROBDDs for the storage of the preprocessed results proves to be feasible. This method can be used together with the method of modeling and simulating bridging faults in the previous chapters. Together they establish a basis for generating test patterns for both open and bridging faults.

6 Concluding Remarks

The objective of this work was the development of accurate and efficient tools to study the logic behavior of defect–induced faults for CMOS circuits and further study their impact on practically used testing methods. Through the research work conducted in this period, our knowledge over this issue is definitely increased and the problems are clearly identified. The possible solutions for modeling and simulating bridging and open faults are investigated in depth. They can serve as the basis of an ATPG system for defect–induced faults as well. The experimental results helped us to build a better vision of shortening the gap between fabrication defects and single stuck–at faults used at logic level. This chapter makes a few additional remarks regarding the methods and results presented in this thesis. Suggestions for further investigation are discussed.

6.1 Remarks

With the aid of a system to extract critical areas [57] and the simple probability relation between critical areas and defect statistics presented in chapter 2, we have extracted all possible faults for a set of benchmark circuits. In view of both the number of faults and the probability of the occurrence of the faults, the results in chapter 2 clearly show that single bridging and single open faults are the primary faults for most CMOS circuits. The preference of using single faults for testing is further supported by the following two considerations.

1) From the testing point of view, a defect affecting a relatively large part of layout can be easily detected by conventional testing methods. This is because if a large portion of the circuit does not function the chance of recognizing this is large. This is not the case for defects affecting only one or two nodes.

2) From the point of view of testing tool development, the single bridging and open faults already change a digital combinational circuit into a circuit

with undefined behavior or even a circuit with sequential behavior. In this thesis, it can be observed that it is not an easy procedure to develop accurate and efficient tools to generate tests for those faults. More complex fault models may even make this procedure too complex to be practical.

It should be mentioned here that the probability measure derived for the extracted faults is based on the extracted critical areas combined with a typical defect statistical data. The results are not biased for a particular process line. It can be the case that one defect mechanism is more likely in a particular process line than another one. But the generality of the method and tools developed in this thesis remains unaffected.

As for the modeling and simulation methods developed in this thesis, all three approaches employ a very simple "divide and conquer" philosophy for both bridging and open faults. That is, the accurate modeling is performed first before fault simulations. Such a "divide and conquer" philosophy not only leads to a very fast fault simulation procedure but also makes it easy to develop an ATPG procedure on the same framework.

As for the two bridging fault modeling and simulation approaches presented in chapter 3 and chapter 4, the approach in chapter 3 is more suited for a design where the frequency of repeated use of each generic–cell is not high. In such a design, probably every bridging fault is a generic–bridge by itself in the worst case. This approach is also suited for the situation where more complex faults need to be included. However the approach presented in chapter 4 is more suited for a design where the frequency of repeated use of each generic–cell is high.

6.2 Suggestions for further investigation

As it is pointed out, defects can cause very complex situations in CMOS circuits. This thesis only focuses on some of the identified problems, that is, the undefined behavior caused by bridging faults and the sequential behavior caused by opens. One of the important faults, feedback bridging faults, is not treated here. Further investigation is necessary since both the number and the probability of the occurrence of feedback bridging faults can be large for some designs.

The fault simulation procedure developed can only tell if a bridging or an open fault is detected for a given test pattern set. There is no proof if an undetected bridging or open fault is testable or not in the entire input space. Thus it is necessary to develop an efficient ATPG procedure. This ATPG procedure can be integrated together with the modeling and simulation approach presented

in this thesis so that a compact test pattern set achieving maximal fault coverage for both bridging and open faults can be given.

This thesis only outlined a bottom–up flow of modeling defects from layout level to circuit faults and further up to logic level. More ambitiously, a top–down method can be developed so that for given test data, a logic fault can be diagnosed down to the defect level on the layout or a test pattern set for fault diagnosis down to defect level can be generated.

References

[1]. J. M. Acken, "Driving Accurate Fault Models," *Computer System Lab. Standford Univ.*, pp. CSL–TR–88–365, October 1985.

[2]. J. M. Acken and S.T. Millman, "Fault Model Evolution for Diagnosis: Accuracy vs Precision", *Proc. of Custom Integrted Circuits Conf.*, pp.13.4.1–13.4.4, 1992.

[3]. K. Antreich, M.H. Schulz, "Accelerated fault simulation and fault grading in combinational circuits", *IEEE Trans. on Computer–Aided Design,* Vol. CAD6, No. 5, pp. 704–712, September 1987.

[4]. P. Banerjee, and J. A. Abraham, "Characterization and Testing of Physical failures in MOS Logic Circuits," *IEEE Design & Test*, pp. 76–86, August 1984.

[5]. Z.Barzilai , J.L. Carter, V.S. Iyengar, I. Nair and et al, "Efficient Fault Simulation of CMOS Circuits with Accurate Models", *Proc. Int. Test Conf.*, pp.520–529, 1986.

[6]. K.S. Bracs, R.L. Rudell and R.E. Bryant, "Efficient Implementation of a BDD Package", *Proc. 27th ACM/IEEE Design Automation Conf.*, pp.40–45, 1990.

[7]. F. Brelez, H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran", *Proc. IEEE Int. Symp. circuits and Systems*, 1985.

[8]. R. E. Bryant and M.D. Schuster. "Fault Simulation of MOS Circuits", *VLSI Design*, 4(6):24–30, October 1983.

[9]. R. E. Bryant, "A Switch–level Model and Simulator for MOS Digital Systems", *IEEE Trans. on computers*, C–33(2), pp.160–177, Feb. 1984.

[10]. R.E. Bryant, "Data parallel switch–level simulation", *Proc. IEEE Int. Conf. Computer–aided Design*, pp.354–357, 1988.

[11]. E. Bruls, F. Camerik, H. Kretschman and J. Jess, "A Generic Method To Develop a Defect Monitoring System for IC Processes," *Proc. Int. Test Conf.*, pp. 218–228, 1991.

[12]. B. Chess and T. Larrabee, "Bridg Fault Simulation Strategies for CMOS Integrated Circuits", *Proc. 30nd ACM/IEEE Design Automation Conf.*, pp.1503–1507, 1993.

[13]. K. Cho and R.E. Bryant, "Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation" *Proc. 26nd ACM/IEEE Design Automation Conf.*, 1989.

[14]. W. Chuang, I. N. Hajj, "Fast Mixed–mode Simulation for Accurate MOS Bridging Fault Detection", *Proc. International Conf. on Circuits and Systems*, pp.1503–1507, 1993.

[15]. H. Cox and J. Rajski, "Stuck–Open and Transition Fault Testing in CMOS Complex Gates", *Proc. Int. Test Conf.*, pp.688–694, 1988.

[16]. C. Di and J.P. de Gyvez, "A Spot Defect to Fault Collapsing Technique," *33rd Midwest Symposium on Circuits and Systems*, pp.580–583, August, 1990.

[17]. C. Di, and J. Jess, "On CMOS Bridge Fault Modeling and Test Pattern Evaluation" *Proc. 11th IEEE VLSI Test Symp.*, pp.116–119, 1993.

[18]. C. Di and J.A.G. Jess, "On Accurate Modeling and Efficient Simulation of CMOS Open Faults," *Proc. of International Test Conf.*, pp.875–882, October, 1993.

[19]. C. Di and J.A.G. Jess, "An efficient CMOS Bridging fault Simulator: with SPICE Accuracy," *submitted to the IEEE transactions on on Computer–Aided Design,* 1993.

[20]. A A.V. Ferris–Prabhu, "Role of Defect Size Distribution in Yield Modeling", *IEEE Tran. on Electron Devices*, Vol.ED–32, no.9, pp.1727–1736, September 1985.

[21]. F. J. Ferguson, and J. P. Shen, "A CMOS Fault Extractor for Inductive Fault Analysis," *IEEE Trans. Computer–Aided Design*, vol. CAD–7, no. 11, pp. 1181–1194, November 1988.

[22]. F. J. Ferguson, and T. Larrabee, "Test Pattern Generation for Realistic Bridge Faults in CMOS ICs," *Proc. Int. Test Conf.*, pp. 492–499, 1991.

[23]. G. S. Greenstein, and J. H. Patel, "E–PROOFS: A CMOS Bridging Fault Simulator" *Proc. Int. Conf. on Computer–Aided Design*, pp.268–272, 1992.

[24]. J. Galiay, Y. Crouzet, and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits:Impact on Their Testability," *IEEE Tran. on computers*, vol.c–29, no.6, pp. 527–531, June 1980.

[25]. J. P. de Gyvez, and C. Di, "IC Defect–Sensitivity for Footprint Type Spot Defects," *IEEE Trans. on Computer–Aided Design*, vol.11, no.5, pp. 638–658, May, 1992.

[26]. T–S. Hwang, C–L. Lee and et al, "A Parallel pattern Mixed–level fault Simulator", *Proc. 27th ACM/IEEE Design Automation Conf.*, pp.716–719, 1990.

[27]. S.K. Jain and V.D. Agrawal, "Test Generation for MOS Circuits Using D–Algorithm", *Proc. 20th* ACM/IEEE *Design Automation Conf.*, pp.64–70, 1983

[28]. G. Janssen, "A ROBDD package: user's manual", *Internal report, Department of eletrical engineer, Eindhoven university of technology,* 1993.

[29]. K–J. Lee and M. A. Breuer, "On the Charge Sharing Problem in CMOS Stuck–Open Fault Testing", *Proc. Int. Test Conf.*, pp.417–425, 1990.

[30]. H. I. Lee and D.S. Ha, "SOPRANO: An Efficient Automatic Test Pattern Generator for Stuck–open Faults in CMOS Combinational Circuits", *Proc. 27th ACM / IEEE Design Automation Conf.*, pp.660–666, 1990.

[31]. K. J. Lee, C.A. Njinda and M.A. Breuer, "Switch Level Test Generation Sytem for CMOS Combinational circuits", *Proc. 29nd ACM/IEEE Design Automation Conf.*, pp.26–29, 1992.

[32]. M.W. Levi, "CMOS is most testable", *Proc. Internaltional Test Conf.*, pp. 316–321, 1981.

[33]. Y.K. Malaiya and S.Y.H. Su, "A New Fault Model and Testing Technique for CMOS Devices," *Proc. of International Test Conference*, pp. 25–34, 1982.

[34]. W. Maly, "Realistic Fault Modeling for VLSI Testing," *Proc. 24th Design Automatic Conference*, pp. 173–180, 1987.

[35]. W. Maly, M. Thomas, J. Chinn and D. Campbell, "Characterization of Type, Size and Density of Spot Defects in the Metalization Layer", *Yield Modeling and Fault Tolerance in VLSI*, edit. by W. Moore, W. Maly and A.J.Strojwas, pp.71–91, Adam Hilger 1988.

[36]. W. Maly, P.K. Nag and P. Nigh, "Testing Oriented Analysis of CMOS ICs With Opens", *Proc. IEEE Int. Conf. Computer–aided Design*, pp.344–347, 1988.

[37]. P.C. Maxwell, R.C. Aitken, V. Johansen, and I. Chiang, "The Effectiveness of Iddq, Functional and Scan Tests: How Many Fault Coverage Do We Need ?", *Proc. Int. Test Conf.*, pp.168–177, 1992.

[38]. P.C. Maxwell and R. Aitken, "Biased Voting: a Method for Simulating CMOS Bridging Faults in the presence of Variable Gate Logic Thresholds", *Proc. of International Test Conference*, pp.63–72, 1993.

[39]. S. D. Millman, and J. P. Garvey, "An Accurate Bridging Fault Test Pattern Generation," *Proc. Int. Test Conf.*, pp. 411–418, 1991.

[40]. L.W. Nagel, "SPICE2: a computer program to simulate semiconductor circuits", *Memo ERLM520, University of California– Berkeley*, 1975.

[41]. J. Rearick and J. H. Patel, "Fast and Accurate CMOS Bridging Fault Simulation", *Proc. of International Test Conference*, pp.54–62, 1993.

[42]. S.M. Reddy, M.K. Reddy and V.D. Agrawal, "Robust Test for Stuck–Open Faults in CMOS Combinational Logic Circuits", *Proc. 14th ISFTC*, pp.44–49, 1984.

[43]. M. K. Reddy, S. K. Reddy, and P. Agrawal, "Transistor Level Test Generation for CMOS Circuit," *Proc. 22nd ACM/IEEE Design Automation Conf.*, pp. 825–828, 1985.

[44]. D. Saab and I. Hajj, "Parallel and Concurrent Fault Simulation of MOS Circuits", *Proc. Int. Conf. Computer Design*, pp.752–756, 1984.

[45]. M.H. Schulz and F. Brglez, "Accelerated Transition Fault Simulation", *Proc. 24th ACM / IEEE Design Automation Conf.*, pp.237–243, 1987.

[46]. H–C. Shih, J.T. Rahmeh, and J.A. Abraham, "An MOS Fault Simulator with Timing Information," *Proc. Int. Conf. on Computer–Aided Design*, pp. 45–47, 1985.

[47]. J.M. Soden and C.F. Hawkins, "Electrical Properties and Detection Methods for CMOS IC Defcets", *Proc. European Test Conference*, pp. 159–167, 1989.

[48]. C.H. Stapper, "Modeling of Integrated Circuit Defect Sensitivities", *IBM J. Res. Develop.* vol.27, No.6, pp.49–557, November 1983.

[49]. T.M. Storey, and W. Maly, "CMOS Bridging Fault Detection" *Proc. Int. Test Conf.*, pp. 842–851, 1990.

[50]. T. M. Story, W. Maly, J. Andrews and M. Miske, "Stuck Fault and Current Testing Comparison Using CMOS Chip Test", *Proc. Int. Test Conf.*, pp.311–318, 1991.

[51]. C.C. Tomic and W.R. Scott, "Simulation of Stuck–Open Faults in CMOS Integrated Circuits", *Proc. Int. Symp. Test and Failure analysis Conf.*, pp.53–56, 1981.

[52]. E. Vandris and G. Sobelman, "Algorithms for fast and memory efficient switch–Level fault simulation", *Proc. 28th ACM / IEEE Design Automation Conf.*, pp. 138–143, 1991.

[53]. R.L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", *The Bell System Technical Journal, vol.57, no.5*, pp.1449–1474, 1978.

[54]. J.A. Waicukauski, E.B. Eichelberger, D.O. Forlenza, E. Lindbloom and Th. McCarthy, "Fault Simulation for Structured VLSI", *VLSI Systems Design*, pp.20–32, Dec. 1985.

[55]. H. Walker, A S. Director, "VLASIC: A Catastrophic Yield Simulator for Integrated Circuits", *IEEE Trans. Computer–Aided Design*, CAD–5, No.4, pp.541–556, October 1986.

[56]. B.W.Woodhall, B.D. Newman and A.G. Sammuli, "Empirical Results on Undetected CMOS Stuck–open Failures", *Proc. Int. Test Conf.*, pp.166–170, 1987.

[57]. H. Xue, C. Di and J.A.G. Jess, "A Net–oriented Method for Realistic Fault Analysis" *IEEE / ACM Proc. of International Conf. On Computer–aided Design*, pp. 78–83, November, 1993.

Appendix A:

On enumerating path–connected subgraphs

Definition A.1: Let $G(V,E)$ be a directed acyclic graph and $a,b \in V$ and $a \neq b$. $G(V,E)$ is said to be $(a–b)$ path–connected if every edge in the graph belongs to a simple path from a and b.

Definition A.2: Let $G(V,E)$ be a directed acyclic graph and $a,b \in V$ and $a \neq b$. Any subgraph $G(V',E')$ that is itself $(a–b)$ path–connected, is called a $(a–b)$ path–connected subgraph of $G(V,E)$.

There may be exponentially many $(a–b)$ path–connected subgraphs of $G(V,E)$. In some applications, such as in this thesis, it is necessary to find all the $(a–b)$ path–connected subgraphs of $G(V,E)$.

Figure A.1 shows an example of $(a–b)$ path–connected graph. It follows from the definition A.1 that for a $(a–b)$ path–connected graph $G(V,E)$, every $v \in V\backslash\{a\}$ has at least one incoming edge in E and every $v \in V\backslash\{b\}$ has at least one outgoing edge in E.

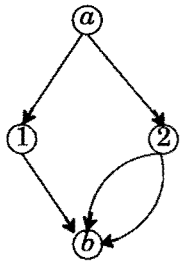


Figure A.1 An example of $(a–b)$ path–connected graph.

In the sequel, for each node $v \in V$, let $E(v) \in E$ denote all the edges from v. Let $E'(v)$ denotes a specific edge subset of $E(v)$.

Definition A.3: For a $(a–b)$ path–connected graph $G(V,E)$, given a non–empty node set $S \subset V$, if a subgraph $G(V' \cup S, E')$ of $G(V,E)$ has the property:

1) every edge of the subgraph belongs to a simple path from a to a node in S;

2) there are no edges between any pair of nodes in S and each node in S is the end–point of at least one path.

Then $G(V' \cup S, E')$ is said to be a $(a\text{–}\hat{S})$ path–connected subgraph where S is regarded as a supernode denoted as $\hat{S}$.

Lemma A.1: For a $(a\text{–}b)$ path–connected graph $G(V,E)$, given a non–empty node set $S \subset V$, a $(a\text{–}\hat{S})$ path–connected subgraph $G(V' \cup S, E')$ is a $(a\text{–}b)$ path–connected subgraph if $S = \{b\}$.

Proof: follows directly from the definitions A.1 and A.3.□

Observe that in a $(a\text{–}b)$ path–connected graph every node has a path to b. This means that given S, a $(a\text{–}\hat{S})$ path–connected subgraph can always be extended to be a $(a\text{–}b)$ path–connected subgraph by adding paths starting from each node of $S \backslash \{b\}$ to b. A possible extension step is given by algorithm A.1.

Algorithm A.1 subgraph extension step

{ Invariant: $G(V' \cup S, E')$ is $(a\text{–}\hat{S})$ path–connected. }
choose $v \in S \backslash \{b\}$;
$V' \leftarrow V' \cup \{v\}$;
choose $E'(v) \subset E(v) \wedge E'(v) \neq \emptyset$
$E' \leftarrow E' \cup E'(v)$;
$S \leftarrow (S \backslash \{v\}) \cup \{u \mid e(v,u) \in E'(v) \wedge u \notin V'\}$;

Lemma A.2: In algorithm A.1, the extended subgraph is still a $(a\text{–}\hat{S})$ path–connected subgraph. By repeating this extension step, eventually a $(a\text{–}b)$ path–connected subgraph can be found.

Proof: For each edge $e(v,u) \in E'(v)$, if $u \in (V' \cup S)$, the set S is not changed. By adding $e(v,u)$ into E', $G(V' \cup S, E')$ still has the $(a\text{–}\hat{S})$ path–connected property. If $u \notin (V' \cup S)$, that is, u is a new node not considered before, since u is added into S, by adding $e(v,u)$ into E', $G(V' \cup S, E')$ still has the $(a\text{–}\hat{S})$ path–connected property.

The repeating process converges because each node $v \in V$ can only appear once in S. At each step one node is removed from $S \backslash \{b\}$ and some new nodes from $V \backslash (V' \cup S)$ may get added into S. Eventually, no nodes that can be added into S are left. The nodes can only be deleted from S. Node b will always appear in S since it is reachable from every node.

Because it is never selected or removed eventually $S=\{b\}$. According to lemma A.1, a $(a–b)$ path–connected subgraph is found.□

With lemma A.2, we can develop an algorithm described in algorithm A.2 that generates a $(a–b)$ path–connected subgraph when initially $S=\{a\}$, $V'=\emptyset$, and $E'=\emptyset$.

The algorithm is based on successive extensions of a $(a–\hat{S})$ path–connected subgraph by choosing a non–empty edge subset from a node in S. To help devise an algorithm that enumerates all possible $(a–b)$ path–connected subgraphs, below a lemma is presented.

Algorithm A.2 generation of a $(a–b)$ path–connected subgraph

```
{ Invariant: G(V' ∪ S,E') is (a–Ŝ) path–connected. }
procedure subg(S);
    if S = {b} then
        V' ← V' ∪ {b};
        G(V',E') is (a–b) path–connected subgraph;
    else
        choose v ∈ S\{b};
        V' ← V' ∪ {v};
        choose E'(v) ⊂ E(v) ∧ E'(v) ≠ ∅;
        E' ← E' ∪ E'(v);
        S ← (S\{v}) ∪ {u | e(v,u) ∈ E'(v) ∧ u ∉ V'};
        subg(S);
```

Lemma A.3: By running algorithm A.2 for all combinations of possible non–empty edge subset from each selected node v, all possible $(a–b)$ path–connected subgraphs are generated. Furthermore no $(a–b)$ path–connected subgraph is ever generated more than once.

Proof: 1) Suppose a certain subgraph is missing. This implies that some edges from a selected node are never selected during the process. This contradicts the assumption in lemma A.3.

2) There are two runs that generate the same result. Assume that in both runs the same deterministic mechanism is used to select a node $v \in S\backslash\{b\}$. Then this means that at some point in the execution a different choice of non–empty edge subset from the same selected node v must be made. Let these two edge subset be $E'(v)$ and $E''(v)$ respectively. Obviously a subgraph containing $E'(v)$ and a subgraph containing $E''(v)$ are not equivalent.□

Algorithm A.3 shows the routine *subgs* that embodies the iteration over all non–empty edge subsets from a selected node in its for–loop. After all

subgraphs corresponding to a certain selection have been generated (coming out of the recursive call *subgs(S')*) we must of course restore the edge set E'. The node set V' must be restored after all such selections for a certain node v have been considered.

Theorem: The *enum* routine precisely enumerates all $(a$–$b)$ path–connected subgraphs of $G(V,E)$.

Proof: Follows directly from lemma A.2 and algorithm A.2.□

Algorithm A.3 enumeration of all (a–b) path–connected subgraphs and initial call

```
{ Invariant: G(V' ∪ S,E') is (a–Ŝ) path–connected. }
procedure subgs(S);
    if S = {b} then
        V' ← V' ∪ {b};
        G(V',E') is (a–b) path–connected subgraph;
        V' ← V'\{b};
    else
        choose v ∈ S\{b};
        V' ← V' ∪ {v};
        for each E'(v) ⊂ E(v) ∧ E'(v) ≠ ∅
            E' ← E' ∪ E'(v);
            S' ← (S\{v}) ∪ {u | e(v,u) ∈ E'(v) ∧ u ∉ V'};
            subgs(S');
            E' ← E'\E'(v);
        V' ← V'\{v};

procedure enum(G,a,b);
    V' ← ∅; E' ← ∅;
    subgs({a});
```

Figure A.2 shows the results of applying algorithm A.3 to the example in figure A.1.

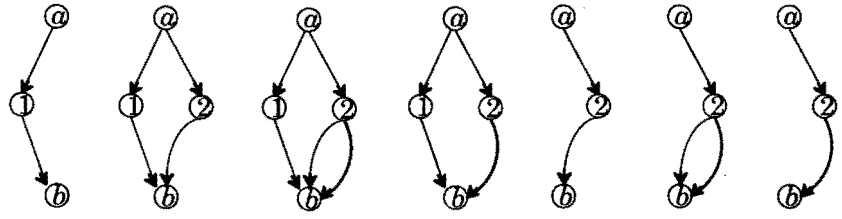


Figure A.2 All $(a$–$b)$ path–connected subgraphs.

Stellingen

behorende bij het proefschrift van Chennian Di

1. The quality of IC testing depends on the actual design in silicon and the manufacturing site.

2. For CMOS digital circuits, the complexity of I_{ddq} testing is analogous to the complexity of conventional voltage testing with respect to the extraction of realistic defect–induced faults, the obtaining of an "optimal" test pattern set and the handling of resistive bridges, feedback bridges and open faults.

3. IC testing should impose as few design rules as possible for the designer. This is reflected by the recent efforts to minimize the scan–path and the increasing interest for sequential ATPG.

4. While using the logic threshold voltage to model the logic value of an undefined state caused by bridging faults, most kind of approximation leads to incorrect and unreliable results.

5. With the success of testability preserving techniques in logic synthesis systems, one may expect that a testability preserving technology mapping technique will emerge. This may be even more important since such a technique operates closer to silicon.

6. The criterion for judging a method or a tool is its efficiency in solving the targeted problem and not the assessment of the age or the elegance of the underlying theory.

   [Deng Xiaoping: it does not matter if a cat is white or black as long as it catches mice.]

7. The design process of a large complex system is more a kind of an art rather than a piece of science. The "beauty" and the degree of automation of the process cannot be enjoyed at the same time.

8. A recently found bug in the Intel's Pentium double–precision divide instruction showed once again that design verification is crucial for the quality of a large and complex design.

9. A lot of technical problems are created just by the reluctance of accepting the knowledge from different subjects or fields.

10. The Dao never does, yet through it everything is done.
    (道 常 无 为 而 无 不 为)

    [Lao Tzu, "the exercise of government" in Chapter 37 of "*Dao De Jing*"]