Dictionary-Less Defect Diagnosis as Real or Surrogate Single Stuck-At Faults

by

Chidambaram Alagappan

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 5, 2013

Keywords: Dictionary-less fault diagnosis, fault simulation, combinational circuits, stuck-at-faults, surrogate faults

Approved by

Vishwani D. Agrawal, Chair, James J.Danaher Professor of Electrical Engineering
Charles E. Stroud, Professor of Electrical and Computer Engineering
Victor P. Nelson, Professor of Electrical and Computer Engineering

Abstract

With Moore's law prediction already reaching its final stages, the number of transistors present in a chip has already touched more than a billion. At this stage, if a chip fails during manufacturing testing, it is becoming highly difficult to diagnose the root cause of the failure with precision. The other major contributing factor to this problem is that the actual defects are always not the same as the modeled defects considered during diagnosis. A diagnostic procedure should be able to give the possible locations where the defect could be present, given the failing occurrences of a chip. In this thesis, we propose a diagnostic algorithm that identifies classic single stuck-at faults as surrogate fault suspects for non-classical faults like multiple faults by analyzing failing circuits.

Several diagnostic procedures have been proposed previously, but not many of them could directly associate the diagnosis results to the actual fault present in the circuit, the reason being, the diagnosis procedure is aimed at diagnosing pre-modeled defects. Few diagnostic procedures that somewhat try to associate with the actual defect are extremely complex. Hence, in this thesis no specific type of defects are assumed. Complexity is kept under control by considering only single stuck-at faults, which are highly analyzable. The single stuck-at faults are identified not as real suspects but as surrogates for actual defects. The diagnostic algorithm, therefore, works with the possibility that the observed behavior of the defective circuit may not exactly match with that produced by the surrogate single stuck-at faults.

The algorithm is based on effect-cause analysis and proves to be less complex than existing methods. This diagnostic procedure involves adding or removing faults from a set of suspected faults based on the observed circuit outputs and minimal fault simulation to finally obtain a small set of candidate faults. Since the proposed algorithm does not require

a fault dictionary, it is memory efficient. Only a single fault simulator is required and is run just a few times. Hence it proves to be CPU time efficient too. The procedure makes use of information from both passing and failing patterns as observed at both erroneous and error-free primary outputs to arrive at the final set of candidate faults.

The proposed procedure was evaluated by conducting experiments on ISCAS85 benchmark circuits using various test pattern sets and was found to be effective. A commercial fault simulation tool Mentor Graphics FASTSCAN was used for automatic test pattern generation and fault simulation purposes. The entire algorithm was implemented using Python programming language and Visual Basic Macros were used for bit manipulation in the test pattern set. The results obtained prove that the proposed diagnostic procedure works as expected and demonstrate its ability to provide real or surrogate faults related to the actual defect present in the circuit.

Acknowledgments

Every treasurable moment of my graduate school career has been shared with many people. It has been a great privilege to spend the last couple of years in the Department of Electrical and Computer Engineering at Auburn University, and its members will always remain dear to me.

My first debt of gratitude must go to my advisor, Dr. Vishwani D. Agrawal. He patiently provided the vision, encouragement and advice necessary for me to proceed through my Master's program and complete my thesis. Being a strong and supportive advisor, he has always given me great freedom to pursue independent work.

Special thanks to my committee, Dr. Charles E. Stroud and Dr. Victor P. Nelson for their support, guidance and valuable suggestions. Their guidance has served me well and I owe them my heartfelt appreciation.

I am deeply grateful to Mr. Charles Ellis, for providing me with an opportunity to work in the AMSTC laboratory. I should also mention Dr. Adit D. Singh for his enlightening class talks.

My friends in US, India and other parts of the world were sources of joy and support. Special thanks to Rucha Gurjar, Suraj Sindia, Yu Zhang, Karthick Alagarsamy and Lixing Zhao.

I wish to thank my parents Alagappan Chidambaram and Nachal Alagappan and my sister Sivagami Swaminathan. Their love provided me inspiration and was a driving force. I owe them everything and wish I could show them just how much I love and appreciate them. I dedicate this work to them and hope it makes them proud.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIP	Binary Integer Programming
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CUD	Circuit Under Diagnosis
CUT	Circuit Under Test
DFT	Design For Test
DUD	Device Under Diagnosis
DUT	Device Under Test
FWHM	Full Width at Half Maximum
IC	Integrated Circuit
ILP	Integer Linear Programming
LUT	Look Up Table
NFET	N-MOS Field Effect Transistor
PFET	P-MOS Field Effect Transistor
PICA	Picosecond Imaging Circuit Analysis

QBF Quantified Boolean Formula

RU Replaceable Unit

SA0 Stuck-At-0

SA1 Stuck-At-1

SAT Boolean Satisfiability

SSBDD Structurally Synthesized Binary Decision Diagrams

Chapter 1

Introduction

With scaling down of device features in the semiconductor fabrication process, to an extent that feature sizes can be expressed in two digit number of nanometers, the humongous amount of effort and money to fabricate them and put them into a single IC will all be wasted if they fail to work as expected. This would even more hurt the manufacturer if these faulty chips are sent out to the market by mistake, because it causes a fortune to recall the faulty chips from the market, after being sold. To compensate for the cost of manufacturing these failing chips, the manufacturer adds these to the cost of the good chips that are being sold. So not only does the consumer pay more, the discarded failing chips contribute to electronic waste.

To reduce the occurrences of manufacturing a faulty chip, all the chips are thoroughly tested at the end of each stage of the manufacturing process. The faulty chips identified at each stage are stopped being put through the next stage of manufacturing, to avoid wastage of resources. Even after thorough testing, few faulty chips escape and are manufactured. The faulty chips are the ones which affect the yield of the process. Every manufacturing firm, strives to achieve 100% yield. But the truth is that none of them have achieved it to date. It is difficult to achieve, because of the many factors, which include process variation, die area and number of process steps. Since the size of the transistors keeps reducing drastically, various sources of failure which did not affect the bigger transistors, now become prominent. To discover the faulty chips and to ramp up the yield, failure analysis is a vital procedure.

An ideal fault diagnosis algorithm or procedure should be able to report the true failures with highest accuracy, i.e., the *Resolution* (the number of true failures reported among

the total number of faults reported) and *Diagnosability* (the percentage of correctly identified failures) of a fault diagnosis technique should be high [12]. Various research works on fault diagnosis procedures fight to achieve this optimum trade-off between the resolution, diagnosability and CPU time.

The actual defects present in the circuit can be anything. To aid in diagnosing, there are several pre-modeled defects known as *Fault Models*. These fault models have been built and improved over time to try and relate as close as possible to the actual defect present in the circuit. Since all the actual faults cannot be perfectly modeled, it is highly impossible to expect test pattern generators to target actual faults and fault simulators to be able to simulate them. Hence the success of a diagnostic procedure is in how close it relates the actual fault to the reported fault. In this thesis, we present a diagnosis procedure which will diagnose the actual fault, if it is a classic single stuck-at fault, and will provide surrogate faults if the actual fault is a non-classical fault.

Fault diagnosis procedures are usually complex and employ heuristics. But in this thesis, we propose a procedure which is very less complex and does not use any heuristics. Simple single stuck-at fault simulations are used to decide the final set of candidate faults.

1.1 Problem Statement

The goal of this research work is to give potential fault(s) or surrogates of the potential fault(s) causing a circuit to fail when given the test vector set, failing responses and good circuit netlist.

1.2 Thesis Contributions

The contribution of the work described in this thesis is to provide a basic core algorithm which will diagnose failing circuits with the help of single stuck-at fault simulation information. If the circuit fails because of a classic single stuck-at fault, the algorithm will be able to identify the actual single stuck-at fault. If the circuit fails because of a non-classical fault,

the algorithm will be able to provide surrogate single stuck-at faults which will have at least a few, if not all of the characteristics of the actual non-classical fault.

The diagnosis method is based on effect-cause analysis and requires significantly less memory, since it does not have to store a full response dictionary, a pass/fail dictionary or any other information. Also, it does not involve re-running simulations to move faults to and from the final candidate list. The procedure works with very minimal fault simulations and proves to be efficient. This work is also documented in [7] and [8].

1.3 Organization of Thesis

Chapter 2 will introduce the reader to basic concepts of VLSI fault diagnosis in order to understand the significance of the problem solved by the proposed work. Chapter 3 discusses the previous contributions in the field of fault diagnosis. The diagnosis algorithm is explained in Chapter 4 and the analysis of the same is done in Chapter 5. The experimental results obtained during the implementation of the proposed algorithm on different benchmark circuits is explained in Chapter 6. Chapter 7 concludes the thesis.

Chapter 2

Fault Diagnosis - Background and Overview

Electronic circuits have evolved in leaps and bounds from a start where discrete components were connected by individual pieces of wire. The current technological advancements allow the components and interconnections to be formed on a same substrate, typically a semiconductor [21].

Any electronic circuit can be basically classified as an Analog circuit or Digital circuit or a Mixed Signal circuit. Analog circuits operate on continuous valued signals, i.e., the current or voltage may vary continuously with respect to time, corresponding to the information being presented. These circuits are mostly custom made and lack flexibility. Digital circuits operate on signals that exist only at two levels - 0''s and 1''s. The signals take either of these values to represent logic and numerical values. Designing digital circuits involves minimum human interaction and they are highly flexible. Mixed signal circuits are a combination of both analog and digital circuits. Most radio and communications circuitry use mixed signal circuits.

Even though digital circuit design can be automated and is highly flexible, the effectiveness of the design automation needs to be verified. Fault verification is the key here.

2.1 Digital Circuits

Digital circuits are widely used because of their advantage over analog circuits being that the signals represented digitally can be transmitted without degradation due to noise. Digital circuits can be sub-classified into Combinational logic circuits and Sequential logic circuits.

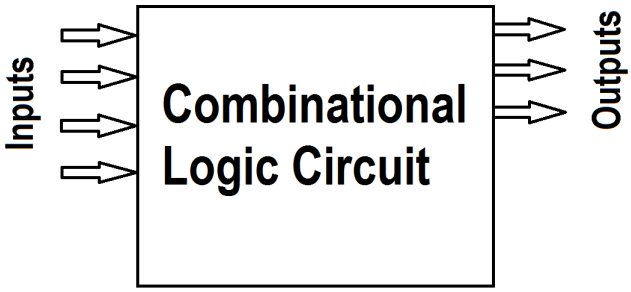


Figure 2.1: Combinational Logic circuit.

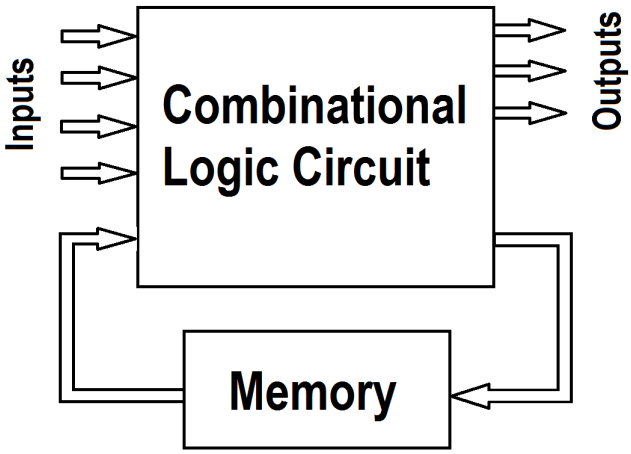


Figure 2.2: Sequential Logic circuit.

A combinational logic circuit implements a Boolean function that maps an input bit string onto an output bit string. As shown in Figure 2.1, the output of such a circuit is purely dependent only on the inputs to the circuits. Because of this reason, they are also known as time-independent logic circuits.

A sequential logic circuit, as shown in Figure 2.2, has inputs and outputs like any combinational circuit. But it also has a clock signal and memory elements, usually flip-flops which are updated whenever the clock signal is triggered. The outputs of sequential circuits are not only dependent on the present value of the inputs, but also on the past history of the inputs.

Testing any circuit involves applying a few selected inputs and observing the outputs for verification of the circuit functionality. This is very evidently easier to perform on any combinational circuit. The sequential circuits, as they involve memory elements, look to be difficult to test. But ever since the advent of the scan based design technique [12], a DFT structure, it has become easy. This technique, for a full-scan-based design, has all the memory elements connected in a chain fashion, known as a scan chain. The contents of these memory elements can be scanned in or out via the scan chain. Hence we will be able to verify the circuit after every test input application, thereby making the testing of sequential circuits easy and exactly same as testing a combinational circuit. For this reason, this thesis deals with fault diagnosis on combinational circuits alone.

2.2 Detection of failures

As explained in the previous section, every circuit is tested after manufacturing with a set of selected inputs. Ideally, all the possible inputs to a circuit should be applied and outputs should be verified. But with the circuits having a large number of inputs, this becomes highly impossible. Hence a small set of inputs are chosen in such a way that the maximum possible faults, if not all of the possible faults in the circuits will be exercised. This set of inputs are known as `test vectors`, `test patterns` or a `test set`. There are ATPG programs which will produce the test pattern set when provided with a circuit netlist. The problem here is that these test patterns will only let you know if the circuit is working as expected or not. If the circuit is not working as expected, they do not actually give you the cause of the failure. `Fault Diagnosis` is the process of finding the actual cause of the failure when a circuit fails.

The test patterns are driven into the circuit and the outputs are monitored by a `tester` machine which is popularly known as ATE . The ATE makes use of a `test program` which has the set of instructions that the ATE should follow while the testing procedure is done. The ATE also stores the test pattern set and the expected and the observed output responses.

Whenever the expected output response is not the same as the observed output response, the ATE flags it as an error. The patterns in a test pattern set which produce an error are known as `failing patterns`. The patterns in a test pattern set which produce an observed output response which is the same as the expected, are known as `passing patterns`.

2.3 Automatic Test Pattern Generation

Automatic test pattern generation is the process of generating patterns to test a circuit, which is described strictly with a logic level netlist or schematic. These algorithms usually operate with a fault generator program, which creates the minimal collapsed fault list, so that the designer need not be concerned with fault generation.

An ATPG algorithm will inject every possible fault, one after another, in the good circuit schematic and will try to activate the fault and propagate the fault effect to one or more of the primary outputs. This way, the observed output response will differ from the expected output response. So the fault is said to be detected. The fault is activated when the fault site holds a value which is opposite to the actual value because of the fault. To propagate this fault effect through an input of a logic gate, the other input(s) of the logic gate are set to a `non-controlling value`, i.e., the output of the gate does not depend on these inputs because of the value on them. The non-controlling value for a AND/NAND gate is '1' and for a OR/NOR gate is '0' and for a XOR/XNOR gate is $\overline{input}$.

In certain sense, ATPG algorithms are multi-purpose, in that they can generate circuit test patterns, they can find redundant or unnecessary circuit logic and they can prove whether one circuit implementation matches another circuit implementation [12].

To test for the faults present in a circuit, three types of test sets can be used.

- Exhaustive Test Set - Apply every possible input combination to a DUT. It is guaranteed to detect all detectable faults, but it is not practical due to the number of test patterns required.

- Functional Test Set - Exercise every functional node with every possible data set. If the test pattern set is complete, this will also detect every detectable fault. But it takes too much time to find test patterns that ensure that the test pattern set is complete.
- Fault Model Derived Test Set - Find a test pattern for every `Modeled Fault`.

As seen, the most efficient method for test pattern set generation is to develop a model of the physical defects that can occur in a fabricated device at a higher level of abstraction, typically the logic level and then develop test patterns to detect these modeled faults [25].

2.4 Fault Simulation

Fault simulation is almost a reverse process of ATPG. The main difference here is that the test pattern set and a fault list are the inputs to a fault simulation program. A fault simulator is used to detect the faults that can be detected by a given test pattern.

The fault simulator will inject a fault (or a set of faults) and drive all the input test patterns. It then compares the expected output response and observed output response. If there is a mismatch, then the injected fault is said to be `detected`. If there are no mismatches, the fault is said to be `undetected`. Usually most fault simulators, as soon as a fault is detected, stop simulating the fault and input another fault (or a set of faults) and start the simulation from the first test pattern again. This is also known as `fault dropping`.

Once the fault simulation has been performed for all the faults, fault coverage of the test pattern set can be calculated. It is a quantitative measure of the effectiveness of the test pattern set in detecting faults [36]. It is given as

$$FC = \frac{Number\ of\ detected\ faults}{Total\ number\ of\ faults\ in\ fault\ list} = \frac{D}{T} \tag{2.1}$$

To be ideal, Fault coverage should be 100%, but it is highly impractical. Hence a fault coverage above 95% is acceptable. But this also depends on the intended application. To improve the fault coverage, the faults that are undetected are given to the ATPG program

to generate test patterns that will detect the fault. Hence the ATPG and fault simulator work in tandem to produce a higher fault coverage, thereby improving the quality of testing.

2.5 Fault Models

As previously discussed in chapter 1, the actual defect present in a circuit can be anything. So we seek the help of fault models to develop test patterns to detect the actual defect. Depending on the quality of the fault model, a test set developed in this manner will most often cover a large percentage of the actual physical defects.

Fault models are used to emulate actual defects, in order to evaluate the effectiveness of a test pattern set. It is very important that they be computationally efficient with respect to the fault simulation environment. The most widely used fault models include gate-level faults, transistor-level faults, bridging faults, and delay faults. All of these fault models can be emulated in a simulation environment and, for the most part, they closely approximate the behavior of actual defects and faults that can occur during the manufacturing process and system operation [36].

A circuit may fail because of a defect present, which either affects the logical function of the circuit or the timing of the circuit. Hence various fault models were developed to represent both the logical function incorrectness and also timing incorrectness. Some of them are explained as follows.

2.5.1 Stuck-at fault model

The Stuck-at fault model is a gate-level fault model. It represents gate inputs or outputs fixed (stuck) at a particular logic value, irrespective of the signals applied to them. Since we are dealing with digital circuits, the gate inputs or outputs are stuck at '0' or '1' which are represented as `Stuck-at-0` or `Stuck-at-1`, respectively.

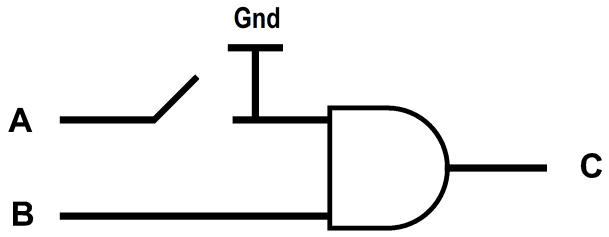


Figure 2.3: Stuck-at-0 fault model.

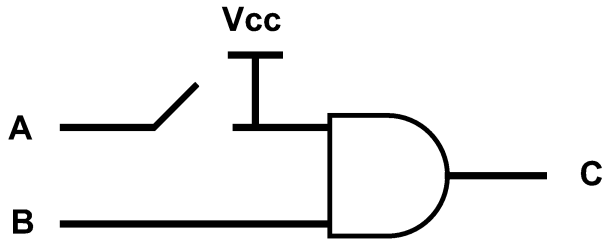


Figure 2.4: Stuck-at-1 fault model.

Stuck-at faults are modeled by disconnecting the input or output from the rest of the circuit and connecting it to the Vcc or Gnd for stuck-at-1 and stuck-at-0, respectively, as shown in Figure 2.3 and Figure 2.4.

Stuck-at fault models are sub-classified into two categories.

- Single stuck-at fault

  The most widely used fault model and is known as the classical stuck-at fault model. The major assumption here is that there is at most one faulty line in the circuit, which is either stuck-at-0 or stuck-at-1. That is, the effect of the fault is as if the faulty input or output node is tied to either Vcc or Gnd. The fault is considered to be permanent as opposed to transient. Also, the point to be noted here is that the function of the gates is unaltered by the fault. The fault only affects the faulty node and not any of the gates.

  This fault model type can be applied at the logic or even module level. It is technology and design style independent. For a circuit with 'n' nodes, the number of possible faults

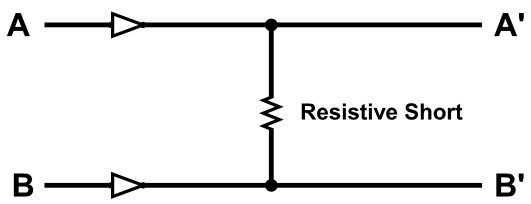


Figure 2.5: Bridging fault model.

is '2n'. Hence complexity is less. Research indicates that the single stuck-at fault model covers 90% of possible physical manufacturing defects in CMOS circuits [19]. Also, other fault models like stuck-open and bridging faults can be mapped into sequences of stuck-at faults. However, the single stuck-at fault model cannot cover all possible CMOS defects.

- Multiple stuck-at fault

  The biggest draw back of single stuck-at fault is that it assumes there is only one fault in the circuit at a time, which is not true all the time. Hence the multiple stuck-at fault model was introduced. It assumes there can be two or more stuck-at faults in the circuit, at a time. But the trade-off here is that the complexity is high. For a circuit with 'n' nodes, the number of possible faults is '3n - 1'. Because of the possibility of multiple stuck-at fault combinations, this model does not significantly increase the defect coverage enough to offset the complexity introduced.

2.5.2 Bridging fault model

Due to under etching during the VLSI fabrication process, common defects like short between two wire segments occur. These defects produce a low resistance short (hard short) between the two wire segments as shown in Figure 2.5. Such defects can be modeled as bridging faults. It is more applicable as spacing between lines gets smaller, since there is a higher probability of two lines getting shorted. Three classes are typically considered for bridging faults - Bridging within the logic element (transistor gate, source or drain shorted

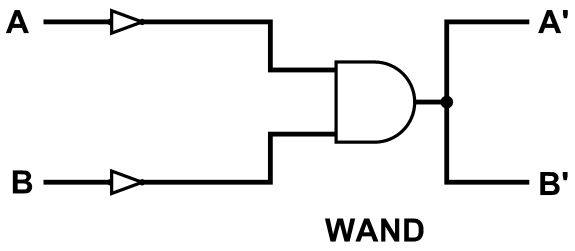


Figure 2.6: Wired-AND bridging fault model.

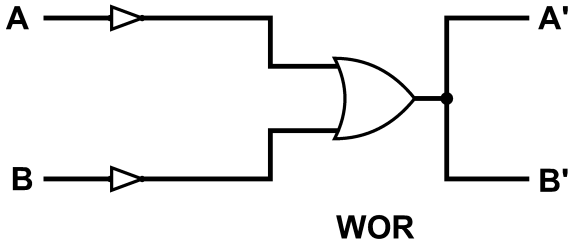


Figure 2.7: Wired-OR bridging fault model.

together), bridging between logic nodes (input or output of logic elements) without feedback and bridging between logic nodes with feedback. Bridging of non-logical nodes between logic elements (transistor shorts across logic elements) is not considered. The most commonly considered is bridging between logic nodes. The other two are not considered here because of the fact that a more complex model is required and they are also covered most often by other fault models, like the stuck-at fault model. There are two major types of bridging fault models, wired-AND/ wired-OR bridging fault model and dominant bridging fault model [36]. Testing bridging faults require setting the two bridged nodes to opposite values and requires a lower level circuit description for faults within the logic elements.

- Wired-AND/wired-OR bridging fault model

  This fault model represents the fault when the end of the shorted wires receive the logical AND of the values on the independent wires before they were shorted, as shown in Figure 2.6. It is also known as 0-dominant bridging fault since a logic '0' on either of the shorted wires will result in a '0' on both the wires. The wired-OR fault model is the same as the wired-AND, but with the logical OR as shown in Figure 2.7.

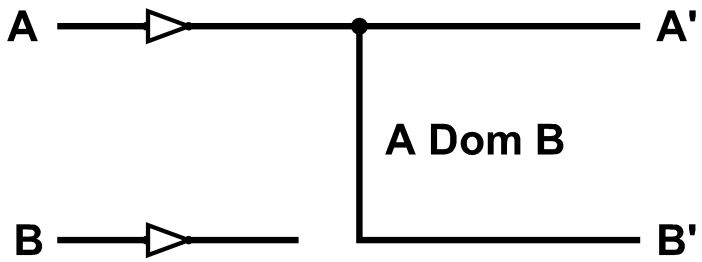


Figure 2.8: Dominant bridging fault model - A Dom B.

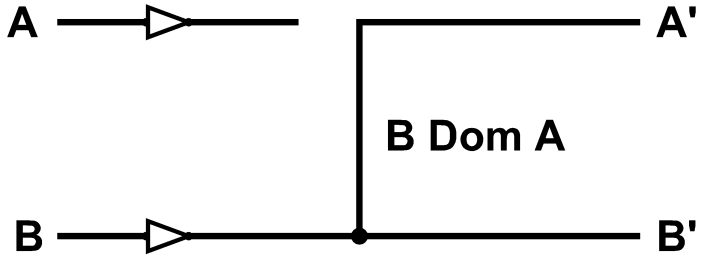


Figure 2.9: Dominant bridging fault model - B Dom A.

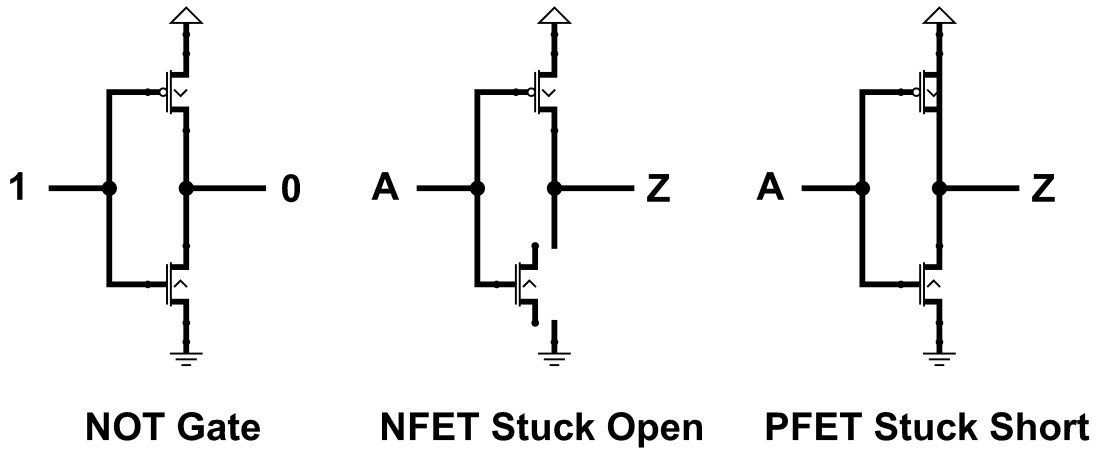
- Dominant bridging fault model

  The dominant bridging fault model represents faults where the logic value at the end is determined by the source gate with the strongest drive capability, as shown in Figure 2.8 and Figure 2.9.

2.5.3 Transistor level stuck fault model

The transistor level stuck fault model has been very helpful in getting defect coverage higher in CMOS circuits and requires a lower level circuit description. The transistors can either be stuck-on/stuck-short or stuck-off/stuck-open. Figure 2.10a shows the fault free transistor level NOT gate.

Figure 2.10b shows the NOT gate with the NFET stuck-open. The assumption with the stuck open fault model is that a single physical line in the circuit is broken. But the resulting unconnected node is not tied to either Vcc or Gnd as in a stuck-at fault model. The broken line retains the previous value it held for some undetermined discharge time.



(a) Not gate. (b) Stuck-open fault. (c) Stuck-short fault.

Figure 2.10: Transistor level stuck fault models.

This creates a `memory-effect`. Hence to test these kind of faults, we can use a sequence of stuck-at fault tests. Because of this, larger number of tests are required and algorithms for ATPG and fault simulation become more complex and are less well developed.

Figure 2.10c shows the NOT gate with the PFET stuck-short. This fault represents the source and drain of a transistor being shorted. Hence it is switched ON and conducting all the time. For certain inputs, the circuit with stuck short fault will produce a conducting path from the source to ground through the output node during steady state. Hence the voltage at the output will be the function of the voltage divider formed by the channel resistance of the conducting transistors. There are specialized testing techniques like `IDDQ testing` to test for these faults.

2.5.4 Delay fault model

All defects need not affect the function of a circuit. Even if the timing of a circuit is affected by a defect, it has to be modeled. Some physical defects such as process variations makes some delays in the CUT greater than some defined acceptable bounds. The two delay fault models are `gate delay fault model` and `path delay fault model`.

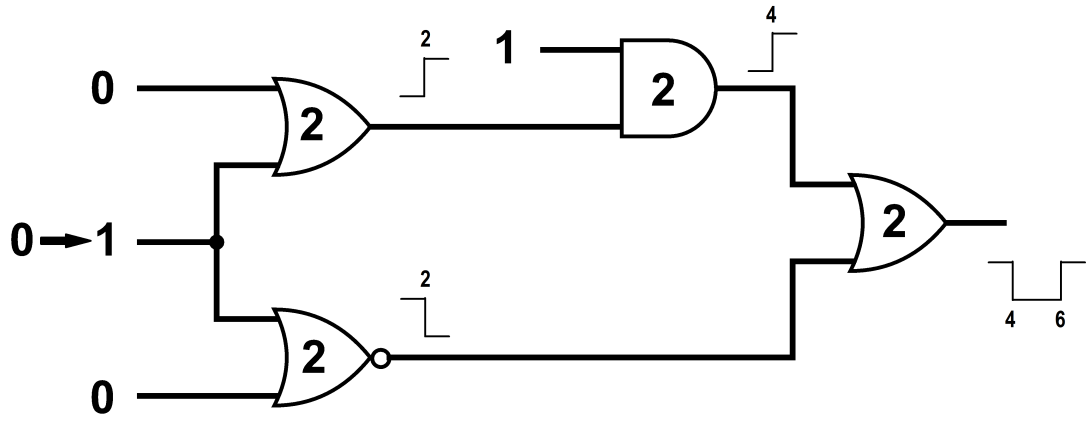


Figure 2.11: Gate delay fault model.

Gate delay fault model represents the defect that delays either a falling transition (slow-to-fall) or a rising transition (slow-to-rise) on a specific line. If the delay fault is large enough, it can be modeled as a temporary stuck-at fault. Again, two patterns are required. One for initialization and other for transition detection.

Consider the input transition shown in figure 2.11 to be a slow-to-rise transition fault. A delay is produced because of the transition fault, while the internal nodes of the circuit settle to their final values. Also, the minimum delay fault size that can be detected is difficult to determine.

Path delay fault model represents timing failures when the total delay in a path from inputs to outputs exceeds some fixed acceptable value. This includes the fact that the delay of a faulty gate can be compensated by gates in the propagation path that have faster than typical delays.

There are several other fault models. But the ones discussed above are most widely used.

2.6 Fault Diagnosis

A CUT is said to be failing when its observed behavior is different from its expected behavior for the test patterns applied. Diagnosis is the procedure of locating the fault in

the circuit using the information available. In case of an IC, repairing it is not economical. But in other cases, repairing a failing CUT will involve substituting the identified faulty `Replaceable unit` (RU) with a new one. In cases where the diagnostic procedure comes up with two indistinguishable RUs, one RU is replaced first and the testing procedure is followed. If the circuit functions correct, then the replaced RU was the faulty one, else the other RU is the one which is faulty. This procedure is called `Sequential Diagnosis`.

The cost of testing and repairing a circuit increases with the stage of fabrication of the circuit. If it costs $1 to test an IC, the cost of testing for the same faulty IC after mounting it on a board will be $10. If the defective board is put into a system, then locating the fault will cost $100. This is called the `Rule of 10`. Hence the quicker we diagnose the fault, the lesser amount of resources we spend.

There are several methods for fault diagnosis and various methods employ different techniques to identify the faults. The next chapter discusses the various researches done in fault diagnosis procedures.

Chapter 3

Previous Contributions in Fault Diagnosis

VLSI fault diagnosis plays a major role in ramping up the yield. Hence there has been a lot of research effort to develop the ideal fault diagnosis procedure. This thesis focuses mainly on classic stuck-at fault and multiple stuck-at fault diagnosis. It is assumed that, bridging fault diagnosis can be done using stuck-at faults. Hence in this chapter we will see previous contributions made in the diagnosis of above mentioned faults.

Fault diagnostic procedures are based on a number of concepts. Any fault diagnostic procedure can be classified into one of the following two categories. (i) `cause-effect based analysis` and (ii) `effect-cause based analysis`.

Cause-effect based analysis has a stored simulated response database of modeled faults. This database is often known as a 'dictionary'. The faulty circuit response is compared against this database to find out which fault might have caused the failure [5, 10, 28, 40].

Fault dictionaries are of two types - `'Pass/Fail Dictionary'` and `'Full Response Dictionary'`. An example of pass/fail dictionary is shown in Table 3.1. All the possible single stuck-at faults are injected into the circuit one at a time and fault simulation is performed on the faulty circuit. If a test pattern fails, a '1' is placed in the corresponding column for the respective injected fault. If a test pattern does not fail, a '0' is placed in the corresponding column for the respective injected fault. The bits in the row corresponding to each fault are known as a `Signature`. For example, the signature of fault F1 is '101'. Now that the information of which fault will fail on which test patterns and pass on which test patterns of the test set is known, comparing this with the test results of the actual faulty CUT will let us know which fault is present in the CUT, if it is a single stuck-at fault [32, 34, 14].

Table 3.1: Pass/fail dictionary.

Faults	Test Patterns		
	T1	T2	T3
F1	1	0	1
F2	1	1	0
F3	0	0	1
F4	0	0	1

Table 3.2: Full response dictionary.

Faults	Test Patterns					
	T1		T2		T3	
	OP1	OP2	OP1	OP2	OP1	OP2
F1	1	0	0	0	1	0
F2	1	1	0	1	0	0
F3	0	0	0	0	0	1
F4	0	0	0	0	1	0

It is quite possible that two faults have the same signature in a Pass/Fail dictionary, since we do not have the complete information recorded. In table 3.1, faults F3 and F4 have the same signature. Hence we are not able to distinguish between them. To overcome this, the Full Response Dictionary is used. Here, we record the entire information of a failing pattern along with the output on which a pattern fails. Table 3.2 shows the full response dictionary for the same example. Even though both the faults F3 and F4 fail only on the test pattern T3, with the full response dictionary, we are able to see that they fail on different primary outputs. Hence we will be able to distinguish between them. Even with a full response dictionary, two or more faults can be indistinguishable. Then the test pattern set should be increased with another test pattern which can distinguish between them. If there exists no such test pattern, those faults are indistinguishable or equivalent faults.

Since the dictionary method uses a stored simulated response database, it requires a large amount of memory and becomes impractical for large circuits. To overcome this memory constraint, several techniques like dictionary indexing or fault dropping are followed to get a compact dictionary. Fault dropping technique does not simulate a fault after it has

been detected by a test pattern, instead it moves on to the next fault in the list. Therefore every fault will be detected exactly only once. Dictionary indexing technique involves storing a index for a particular signature, instead of storing the whole signature. This saves some memory. Dictionaries frequently contain unwanted information and can usually be made smaller. Some techniques do this by limiting fault simulation, reducing size and cost but sacrificing resolution [29].

However, dictionary based fault diagnosis holds good only for single stuck-at faults. For multiple stuck-at faults or other non classical faults, the dictionary method is impractical because of the number of combinations of faults that can be present, which increases the complexity.

In [32] a diagnostic dictionary is built and the expected and failing responses are compared to detect the actual fault. There is a counting procedure used here which adds two counts to a fault every time the simulated response and the good circuit response differs. If the observed response is not known a count of one is added. There is no change to the count in other cases. Finally the faults with the highest count are reported as suspects.

An improved work in [33] utilizes a test pattern set with 100% fault coverage to test the circuit and the fault simulation is performed on the failing patterns to get the common faults. Then a dynamic dictionary is formulated by using a larger test set to continue with the fault diagnosis as stated in [32]. This proves to be lot more memory efficient because of the dynamic dictionary. However, there is a trade-off in terms of simulation time due to increased number of times the faulty circuits are analyzed.

The other famous alternative to fault dictionary is the `Diagnostic Tree` approach [12]. Every step of the diagnostic tree provides partial diagnosis. The test patterns are applied one at a time and the depth of the tree depends on the number of test patterns in the test set. The order of the application of the test patterns is chose based on the previous test's results. As shown in Figure 3.1, if a test pattern passes, the diagnosis will follow a particular route and if the test pattern fails, the diagnosis will follow a different route. At the leaf

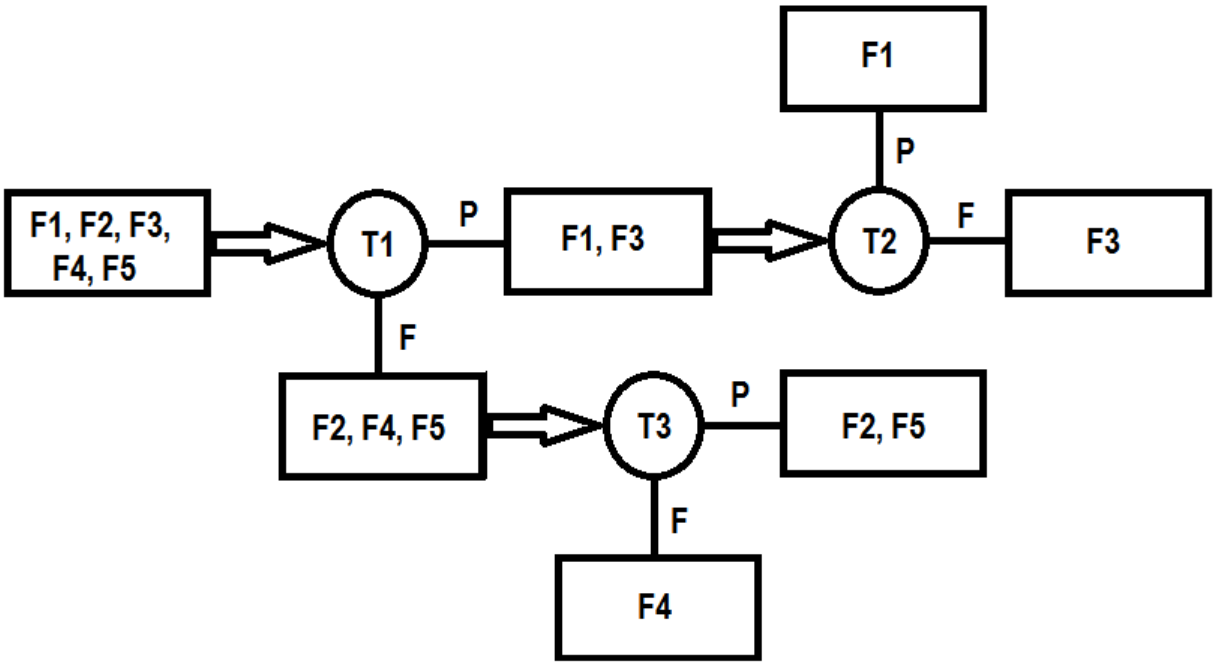


Figure 3.1: Diagnostic tree.

nodes of the diagnostic tree, we will arrive at the suspected fault. If the leaf nodes have more than one fault, those are equivalent faults or indistinguishable faults.

In effect-cause based analysis, as the name suggests, depending on the failing effect produced, the search for the cause of failure is performed [3, 4, 15]. It does not use any stored database. The search for the cause of failure is done by various methods which trace back the error propagation path from the failing primary outputs and consider faults which might have produced the particular failure. Fault location by structural analysis is one of the effect-cause based techniques. If the fault present in the CUT is assumed to be a single fault, then there should exist a path from the site of the fault to each of the outputs where errors have been detected. Therefore it makes sense to state that the fault is most likely present in the intersection of the logic cones which lead to the erroneous outputs. Also, techniques like backward implication and forward propagation are used to detect the fault [15]. These types of diagnosis methods take up reduced or no memory space and can handle any size circuit with just time increasing linearly with the complexity.

The authors in [16] propose X-simulation which uses three value logic simulation - '1', '0' and 'X'. A set of nodes are chosen to have unknown value X on them. The signal propagation is done along with the X values, to the primary outputs. If the X values successfully reach the faulty primary outputs alone, then the nodes with the X value are reported to be suspects. If the X value is lost in mid-way through the circuit or does not reach the actual failing outputs, then the selected set of nodes are relieved from suspicion. This work served as a base for the procedure proposed in [11]. The authors suggest breaking the entire circuit down into small regions and three value logic simulation be performed. After the logic simulation, each block is ranked. The rank will be increased for the region if the 'X' value can propagate to the erroneous output. If a contradictory value is put on the primary output, the regions selected are dropped.

An improved work [35] partitions the circuits into several small blocks which have just a couple of gates and then the three value logic simulation is performed. At the output of each block, when all except one output is observed to have a 'X' value, the value on the remaining outputs is flipped. After continuing with the logic simulation, if the primary outputs contain atleast one erroneous output or an additional erroneous output, then the block is to be abandoned. But these works with three value simulation based on a small region provide very poor resolution.

Fault diagnosis by CUT reduction is also a diagnostic procedure that has been available for a long time now. Here the failing CUT is partitioned into two and separately tested. If there is only a single fault, only one of the partitions will fail. So the failing partition is again divided into two parts and the same procedure is followed. It goes on in the same fashion as a binary search algorithm. But this procedure is again time consuming and is nearly impossible for large circuits.

In reality, failures may not be exactly the classical single stuck-at-fault. But the single fault assumption has worked out very well in practice because diagnostic algorithms are developed, keeping in mind that the success of a diagnostic procedure is in how close it

relates to the actual fault. Researches state, most of the multiple faults are covered by testing for single faults [19]. Also, testing for all multiple faults is not an economical procedure. However, due to the current fabrication levels, we need to consider multiple fault analysis too. Works on multiple fault analysis have been around for quite some time now. Instead of testing for all possible multiple faults, if tests are developed for combinations of faults that are constructed from the resulting suspected fault list of single fault tests, the diagnostic resolution can be highly improved [24].

Multiple faults in a combinational or sequential circuit are corrected using a single error correction scheme in iterations, as proposed in [30]. A heuristic measure is used to guide the selection of single, local circuit modifications that reduce the distance between the incorrect implementation and the specification. The distance is measured by the size of a correction hardware.

The authors in [23] put forward a procedure which utilizes fault dropping for multiple fault detection. Fault collapsing is performed initially to reduce the number of faults by eliminating the equivalent faults. The analysis procedure then involves considering frontier faults among the collapsed faults, which are then used to show that they are equivalent to the set of multiple faults. After fault simulation, a fault dropping procedure is used to eliminate faulty conditions on lines that are either absent or may be masked by other faulty conditions, to arrive at the suspected faults.

Using test groups instead of test pairs for detecting multiple stuck-at faults with presence of fault masking (a phenomenon where a test for a fault fails to detect it because of the presence of another fault) was introduced in [38]. Mutual masking of multiple faults were modeled with the use of structurally synthesized binary decision diagrams. The goal of this work was to verify the correctness of a selected part of the circuit instead of targeting the fault sites as test objectives.

A more recent work utilizing fault dropping for multiple fault diagnosis is proposed in [37]. The procedure makes use of both single and multiple fault simulation information

to add or drop faults to or from a suspected fault list. Repeated simulations are performed to come up with the final set of candidates. A heuristic procedure is used to rank the candidates and the top rank candidates are reported as possible suspects. Even though it is very complex and requires a lot of time, the results lack resolution for bigger circuits.

Electron beam probing is also used for fault diagnosis. Guided probe testing extends edge pin testing by monitoring internal signals in the CUT via a moving probe which is usually guided by the test equipment. The principle behind this procedure is to backtrace an error from the failing primary output. It is carried out step-by-step and in each step, an internal signal is probed and compared to the expected value. The next probing depends on the result of the previous step. But it is highly time consuming and very difficult. Hence most of these fault diagnostic procedures are performed as a pre-processing step for electron beam probing. It proves to be very efficient when there are very few suspected nets to be probed. So, the higher the resolution of the fault diagnosis procedure, higher will be the efficiency of electron beam probing.

IBM researchers have demonstrated the ability to measure optical emission of normally biased CMOS logic gates, which occurs only in a short period during the switching transient. The time response of the measuring system is less than 100 psec (FWHM) . Device characterization through the analysis of dynamic light emission can be performed on the IC, which provides the timing resolution needed to diagnose faults. One such diagnosis method known as Picosecond Imaging Circuit Analysis ($PICA$) is described in [26].

Even methods involving Boolean satisfiability problems, which involve satisfying a Boolean expression or equation, has also been used in fault diagnosis [39]. In [22] a novel approach of adding partial programmability in the CUT by replacing some gates with LUTs for aiding in diagnosis has been explained. The configuration of these LUTs is found using boolean SAT solvers instead of using a QBF solver.

Integer linear programming techniques have also been used in fault diagnosis. In [13], a three stage ILP based diagnosis is performed to diagnose byzantine open-segment defects.

The first stage generates a list of net candidates as defect locations based on path tracing on failing patterns. The second stage reports the combinations of the net candidates that can explain all the erroneous primary outputs. By encoding the candidate nets as a binary integer programming, an ILP solver is used to find the net combinations. The final stage verifies and expands the final net fault candidates.

To overcome the problem of fault model dependent diagnosis, a novel approach to adaptive diagnosis was proposed in [18]. It combines a new effect-cause pattern analysis algorithm with high-resolution ATPG and uses stuck-at faults as a basis for diagnosis. Deviations in defect behavior from the behavior of stuck-at faults are accommodated by using scoring algorithms that estimate the likelihood of a defect being associated with the site of a stuck-at fault given the observed response of the failing circuit.

Reference [45] suggests a procedure of diagnosis using stuck-at and transition fault models. Development of test vectors that improve the diagnostic coverage of faults of various types [42, 43, 44] and application of SLAT patterns, which detect only one fault at a time, to perform the diagnosis of failure responses caused by multiple faults [9] are also being considered. Diagnosis using SLAT patterns work on two assumptions. Any defect behaves as some set of stuck-at faults on application of a pattern that detects it. The stuck-at faults will vary for different patterns that detect it. The second assumption is that there will be some detecting patterns that causes the defect to act as a single stuck-at fault. These assumptions reduce the diagnosis problem back to stuck-at fault diagnosis [20].

There are plenty of other fault diagnosis procedures which work on the trade-offs between the complexity, time and resolution. But the basic concept is more or less the same as discussed in the above methods.

Chapter 4

The Diagnosis Algorithm

This chapter explains the algorithm proposed in this thesis. The diagnostic procedure involves fault simulation using fault dropping to result in set of candidate faults to be suspected. Since this procedure is purely simulation based, it does not use any fault dictionary and thereby proves to be memory efficient. Unlike other fault simulation diagnosis algorithms, this proposed procedure is very simple and does not use any heuristic to rank order the faults. Also, the diagnostic time is drastically reduced, because our algorithm does not involve re-running simulations with the same patterns after ranking or repeated switching of faults from and to the suspected fault list.

4.1 Motivation

As seen in Chapter 3, there are various fault diagnostic procedures which perform diagnosis with either dictionaries or diagnostic trees or some other technique. But diagnosing multiple faults has not been mastered to satisfaction yet. This is mainly due to the huge combination of possible faults, which blows up the initial fault candidate set. If this initial fault candidate set is reduced, most of the fault diagnosis will be made simpler and effective.

The other problem is that almost all of the fault diagnosis procedures are based on one or a couple of fault models. That is, they aim at detecting only selected fault models. This is because different fault simulators and different ATPG programs are required to target different fault models. These ATPG programs and fault simulators are not able to achieve perfection yet. As discussed in chapter 2, just because single stuck-at faults cover maximum of the physical defects that can be present in a circuit, all the diagnosis procedures based on just single stuck-at fault model, work as expected. But this is not the case always. With

devices shrinking fast, various defects are seen becoming prominent. All these defects cannot be tested okay by testing with a single stuck-at fault model.

In this thesis we propose an algorithm which is able to diagnose non-classical faults. It works with available fault simulation tools and hence it is based on classical single stuck-at faults. The diagnosed fault set reported by the algorithm contains only the classical stuck-at faults. However, they are classified as either real or surrogate. If we determine the identified fault to be a single stuck-at fault it is called real. Otherwise, the identified faults are called surrogate, meaning that they have some, but not all, characteristics of the actual defect in the circuit. The term `surrogate fault` has been used before in the literature [17, 31, 41]. These surrogate faults are used to represent the non-classical faults based on the correlation with location or functional equivalence to the real fault. The proposed diagnosis algorithm makes only one assumption, that there is no circular fault masking in the circuit under test. Circular fault masking is the situation where two or more faults can mask each other from being detected.

4.2 Output Selection - A preliminary setup

A circuit is said to be faulty when its output response for a test pattern set does not match the expected good circuit response. Before starting to apply the fault diagnosis algorithm there is some initial setup required to help us obtain a high resolution diagnosis. For explaining the complete diagnosis procedure, we will be using the ISCAS'85 benchmark circuit C17, as shown in Figure 4.1, throughout this thesis.

When a test pattern is reported as failing, the circuit has failed to match the good circuit output response on at least one of the primary output pins. From the failing responses of the output we get to know which test pattern fails on which primary output pin. This information is very vital as it provides information about the fault location. For instance, if a test pattern fails on all the primary outputs, the fault is most likely to be located closer to the primary inputs. In other words, the fault location is actually present in the intersection

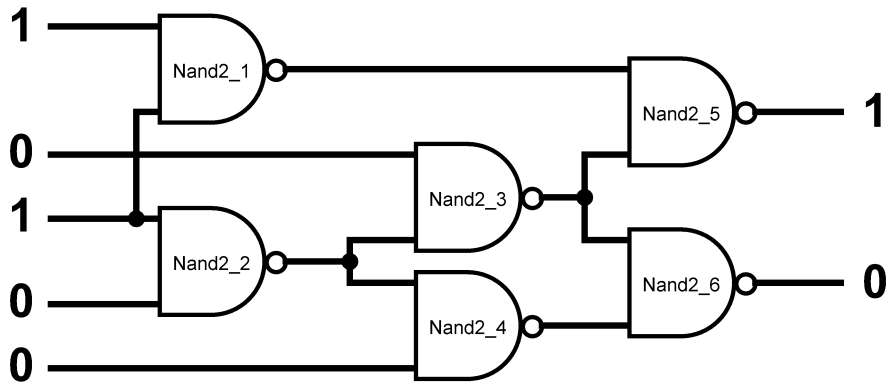


Figure 4.1: C17 benchmark circuit.

of the cones of the failing primary outputs. But if the test pattern fails only on one primary output, the fault must be located close to that primary output or must be present on a line which can affect only this primary output, i.e., the fault which is propagated to other outputs is overwritten by other logic values or the fault does not have a path to propagate to other outputs.

The proposed diagnosis procedure involves single fault simulations with both the failing and passing patterns. During fault simulation with the failing pattern, the fault simulator program will report all the faults that the pattern will be able to detect on all the primary outputs. But considering all these faults when only one or few of the primary outputs are failing will result in waste of resources and precious diagnostic time. Hence, to overcome this difficulty, we use a technique known as 'Output Selection'. Logic AND gates are added at each primary output and the other input of the AND gate becomes another primary input. Now the failing test pattern can be duplicated as many times as the number of primary outputs with just activating only one primary output at a time, i.e., the primary inputs that directly go to the added AND gates should all be forced to 0 except for one AND gate at a time, to activate that particular primary output.

Consider the C17 benchmark circuit shown in Figure 4.1. The test pattern is "10100", whose good circuit response is "10". Assume this circuit fails on this pattern only on the

second output. The pattern's fault simulation will produce faults that can be detected at both the outputs. After implementing output selection as shown in Figure 4.2, the test pattern will be duplicated as "1010010" and "1010001". Since the original circuit passed on the first primary output, test pattern "1010010" will pass too, and will be considered as a passing pattern. Now if we simulate the failing pattern "1010001", we will end up with faults that can be detected only on the second primary output. Hence the faults detected on the first primary output can be absolved of being the suspected faults. This way, we split a single failing test pattern into two, a failing pattern and a passing pattern. The same technique can be implemented by using OR gates instead of AND gates and the forced primary outputs to be '0' instead of '1'. On a large circuit with many primary outputs, this will prove to improve diagnostic resolution and diagnostic time. But this comes with a cost, since all the test patterns are multiplied by the number of primary outputs. This technique was mainly implemented because the fault simulator program used, was not able to provide the information about which faults will be detected on which primary outputs for each test pattern, while performing on-the-fly simulations. There are other fault simulator programs in the market, which can provide this information. Using them, will eliminate the need for 'output selection' implementation.

4.3 The Diagnosis Algorithm

Now that we have the information of failing patterns and output selection setup ready, we can proceed with the algorithm. The algorithm is aimed to be simple and effective. Hence, there is no heuristic or ranking involved. There is also no need for re-simulations with added or removed faults from the suspected fault list. The algorithm uses the most basic concept that a test pattern fails because a fault that it can detect is present in the circuit and a test pattern passes because none of the faults that it can detect are present in the circuit. For this algorithm to be effective, the only assumption we make is that there is no circular fault masking present in the circuit under diagnosis.

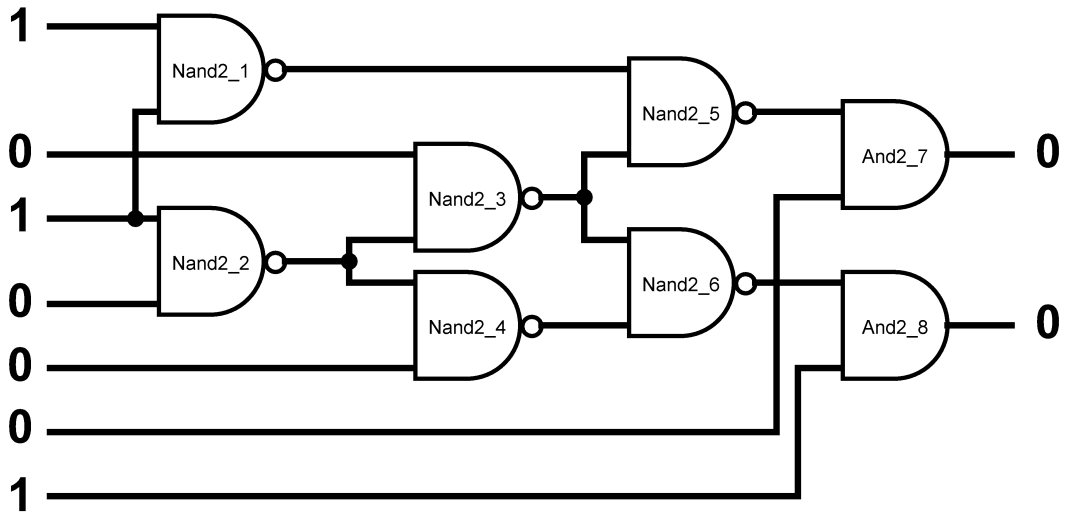


Figure 4.2: C17 benchmark circuit with output selection.

The following nomenclature is used in the diagnosis algorithm:

passing_set - Set of passing test patterns

failing_set - Set of failing test patterns

sus_flts - Suspected fault list

set1_can_flts - Set of prime suspect faults

set2_can_flts - Set of surrogate faults

The proposed diagnosis algorithm works in four phases:

Phase 1:

Step1-1: If *failing_set* is empty, then restore *failing_set* to initial stage and go to Phase 2. Else select and remove a failing test pattern from *failing_set*.

Step1-2: Perform fault simulation for the selected failing test pattern to identify the faults detected by the test pattern.

Step1-3: Add all the faults detected by the pattern to the *sus_flts* set and go to step1-1.

Phase 2:

Step2-1: If *passing_set* is empty, go to Phase 3. Else select and remove a passing test pattern from *passing_set*.

Step2-2: Perform fault simulation for the selected passing test pattern to identify the faults detected by the test pattern.

Step2-3: If the detected faults are present in the *sus_flts* set, remove them and go to step2-1.

Phase 3:

Step3-1: Copy faults from *sus_flts* set to *set1_can_flts* set and *set2_can_flts* set.

Step3-2: If *failing_set* is empty, go to Step3-5. Else select and remove a failing test pattern from *failing_set*.

Step3-3: Perform fault simulation for the selected failing test pattern, to identify the faults not detected by the test pattern among the *set1_can_flts* set of faults.

Step3-4: Update *set1_can_flts* set by deleting the faults that are not detected by the pattern. Go to step3-2.

Step3-5: If faults present in the *set1_can_flts* set, are present in *set2_can_flts* set, delete them from the *set2_can_flts* set and Go to Phase 4.

Phase 4:

Step4-1: If there is no unselected fault in *set1_can_flts* set, repeat Phase 4 for *set2_can_flts* set and then STOP. Else select a fault and uncollapse it to obtain its corresponding equivalent set of faults.

Step4-2: Add the equivalent set of faults to *set1_can_flts* set.

Step4-3: Add the opposite polarity fault for the selected fault and its equivalent set of

faults to *set1_can_flts* set.

Figure 4.3, shows the flowchart of the entire diagnosis procedure. The initial phase of the algorithm, Phase 1, involves performing fault simulation on each one of the failing patterns present in the *failing_set* and adding all the faults that could be detected by the failing patterns to the *sus_flts* set. This is nothing but taking the union of all the faults that are detected by all the failing patterns. The actual fault should be present in this vast set. But since this set is huge, we need a criterion to reduce the number of suspected faults. Hence, in the second phase, fault simulation of each one of the passing patterns present in the *passing_set* is performed. Since these patterns are passing, the faults detected by these patterns can be absolved of being suspect faults. So these faults are deleted from the *sus_flts* set, if present. This phase is nothing but taking the union of all the faults that are detected by the passing patterns and subtracting it from the union of all the faults that are detected by the failing patterns. The Phase 1 and Phase 2 of the algorithm are interchangeable in order, i.e., it does not really matter, if simulation of failing patterns is performed first or the simulation of passing patterns. Generally, starting with the simulation of failing patterns, will result in working with fewer faults, when the passing patterns are more than 50% of the total test pattern set. Phase 3 of the algorithm is all about setting up a prime suspect fault candidate list. This helps in achieving a higher diagnostic resolution. In this phase, fault simulation of each one of the failing patterns present in the *failing_set* is performed again. But this time the faults that are commonly detected by all the failing patterns are stored in *set1_can_flts* set. This is equivalent to taking the intersection of all the faults detected by the failing pattern set. The resulting faults are considered to be 'Prime Suspects'. The remaining fault suspects present in *sus_flts* set are moved to *set2_can_flts* set. These faults are of low priority, but there is a chance that they can be a surrogate of the actual fault or one of the actual faults.

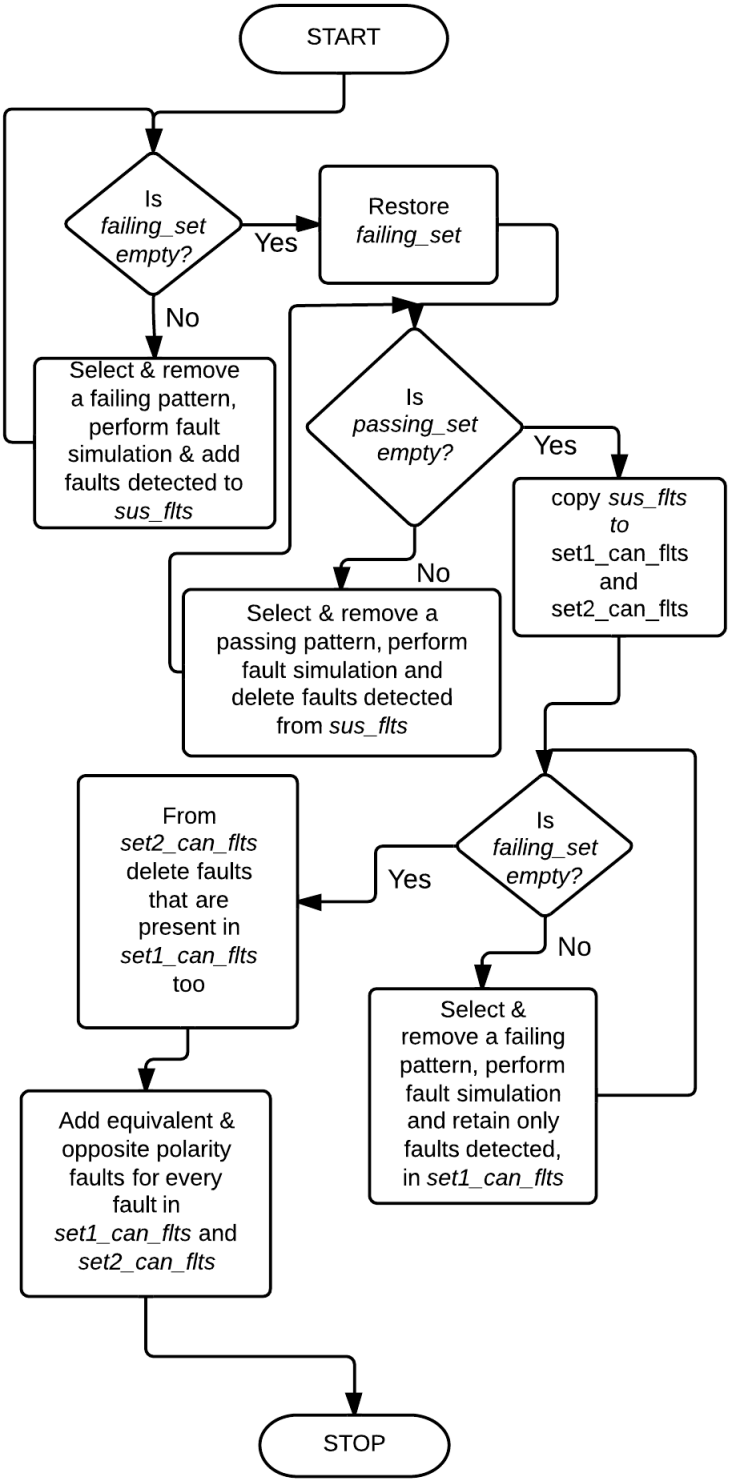


Figure 4.3: Flowchart of diagnosis procedure.

If the actual fault present is a classical single stuck-at-fault, the algorithm would have identified it by the end of Phase 2. But in practice, the actual fault present is not always a

classic single stuck-at-fault. To detect non-classical faults using a test pattern set that was generated to detect classic single stuck-at-fault will yield an inaccurate diagnosis.

Hence in order to help in detection of non-classical faults like multiple stuck-at-faults, the algorithm provides a list of surrogate faults which might represent the actual fault or the actual fault's behavior, when it is not a single stuck-at fault. The use of surrogate faults to model faults that do not belong to the fault model for which the test pattern set was generated was utilized in [31] and [41].

We proceed further into the algorithm to the final phase, Phase 4. The faults that are present in the *set1_can_flts* set and *set2_can_flts* set are extracted and uncollapsed, i.e., equivalent faults of the set of suspected faults are also brought into the suspects list. When dealing with multiple stuck-at-faults, fault masking is a phenomenon of concern. Many diagnostic procedures proposed are put forth with the assumption of fault masking not present. Even though fault masking is a rare phenomenon in practice, in large circuits, one can never completely be sure that fault masking is not present.

In order to achieve diagnosability when considering non-classical faults to be present in the circuit, we perform the final step to produce surrogate faults that might represent the non-classical multiple stuck-at-faults. We add the opposite polarity faults of the faults that are present in the *set1_can_flts* set and *set2_can_flts* set, to get the final candidate fault list sets. i.e., if the *set1_can_flts* set contains 'a' s-a-0(stuck-at-0), 'b' s-a-1(stuck-at-1) and 'c' s-a-1, the final *set1_can_flts* candidate fault list will have 'a' s-a-0, 'a' s-a-1, 'b' s-a-0, 'b' s-a-1, 'c' s-a-0 and 'c' s-a-1. A similar procedure is followed for *set2_can_flts* set.

To reinforce the point that opposite polarity of the suspected faults should be included to cover fault masking, let us take a look at some basic examples of fault masking.

Two faults of a Boolean circuit are called equivalent if and only if they transform the circuit such that the two faulty circuits have identical output functions. They are also called indistinguishable and have exactly the same set of tests [6]. When multiple faults are equivalent, there is no way to distinguish those faults, because the tests for the equivalent

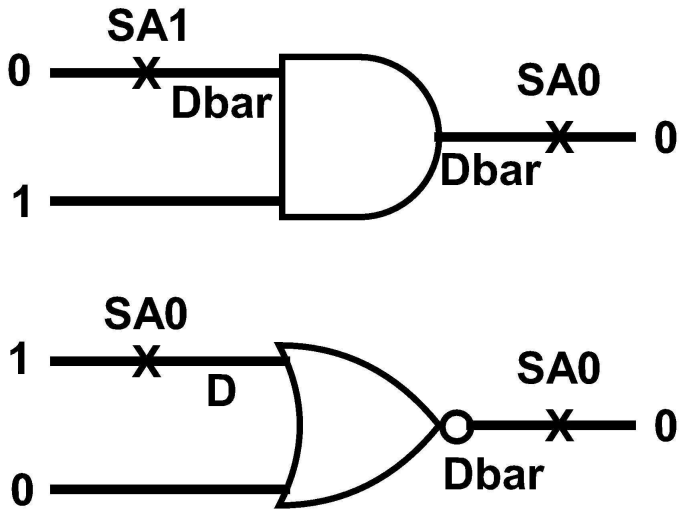


Figure 4.4: Opposite polarity fault masking.

faults are the same. Hence they will react as one single fault. To take this into account, the first part of Phase 4 of the algorithm adds all equivalent faults of the detected faults. This way, even if the detected fault is not the actual fault, but one of its equivalent faults is the real fault, we will be able to diagnose it. For an AND gate, all stuck-at-0 faults are equivalent. In Figure 4.4, input1 of an AND gate is stuck-at-1 and the output is stuck-at-0 . To activate the fault on the input1 line, a '0' must be supplied to input1 and a '1' must be supplied to input2, to propagate it. This will produce a Dbar (which denotes the good circuit value to be '0' and bad circuit value to be '1') on input1 and it will be propagated to the output of the AND gate. But it will be stopped from being propagated to the primary output because of the output stuck-at-0 fault masking it. Since the output fault is masking the input fault, the diagnosis procedure will come up with the output stuck-at-0 as a suspected fault. Because of this, stuck-at-0s on both the inputs are also included as suspects. But here, the case is that the input1 is stuck-at-1. Similar is the case with the NOR gate shown in Figure 4.4, where input1 and output are stuck-at-0. This will be the same for any such case of opposite polarity fault masking. So the second part of Phase 4, where including the opposite polarity of equivalent faults is performed, will cover for these cases of masking in a circuit.

Even though the fault diagnosis procedure will produce surrogate faults that will be able to accommodate most cases of fault masking, it will fail to represent the masked fault when it is situated far away from the masking fault. To try and cover this aspect of fault masking, will result in very poor and unacceptable diagnostic resolution. But fault masking is a very rare case. With circuit sizes getting larger, the probability of fault masking reduces. Even if a pessimistic approach is taken and fault masking is assumed in 50% of the failing circuit cases, out of which 25% are cases where the masking faults and masked faults are far apart, the diagnostic procedure will come up with the surrogate faults that represent 75% of the failing circuit cases.

Theorem 1. *If there is only a single stuck-at-fault present in the circuit under diagnosis (CUD), the diagnosis algorithm will always diagnose the fault, irrespective of the detection or diagnostic coverage of the test pattern set.*

Proof. Let us assume that there is a single stuck-at-fault in the CUD. It causes $N - k$ test patterns out of an N-pattern test set to fail, where k is the number of passing patterns in the test set. Considering the fact that a fault free circuit will not have any failing test patterns, the failing test patterns on the CUD are due to the presence of failure s. In other words, a test pattern can only fail because a fault that it detects is present. Hence all $N - k$ patterns detect the fault s and the remaining k patterns do not detect the fault s. If all $N - k$ patterns detect some fault present in the circuit, it has to be the same fault that all the $N - k$ patterns detect, because there is no more than one fault present in the circuit according to our assumption in the beginning. Moving forward with this revelation, Phase 3 will always come up with the actual fault, i.e., the intersection of the faults detected by all the failing patterns will be the actual single stuck-at-fault present in the circuit. □

Chapter 5

Analysis of the Algorithm

In a short story, `The Murders in the Rue Morgue`, the author Edgar Allan Poe, says "*The analytical power should not be confounded with simple ingenuity; for while the analyst is necessarily ingenious, the ingenious man is often remarkably incapable of analysis.*" The story revolves around a murder case of two ladies which puzzles everyone. But the murderer actually turns out to be an orangutan.

The diagnosis problem here can also be compared to a murder case investigation. Let us assume the actual fault is the murderer and the test patterns in the test vector set are the witnesses. The failing test patterns are supposed to have some knowledge about the crime scene and the passing test patterns are supposed to have no knowledge about the crime. Hence the failing patterns give information about faults which are the suspects, i.e., the failing pattern witnesses state that they saw the suspects at the crime scene. The passing test patterns give information on which faults are not the suspects, i.e., the passing pattern witnesses state that they saw the suspect somewhere else other than the crime scene, during the time frame the crime took place. They basically give an alibi for the suspect. Based on this alibi the diagnosis procedure, absolves the suspect of the crime and the next suspect is considered. Fault masking and Fault interference can be compared to cases where the witnesses are either lying or do not know the truth.

The Figure 5.1 shows the comparison of simulation effort between the proposed diagnosis procedure and traditional fault dictionary diagnosis method. It was plotted for a multiple (two) stuck-at fault failure case in C432 ISCAS85 benchmark ciruit. Circuit C432 has a total of 1078 single stuck-at faults in the fault list. The test vector set with 100% diagnostic coverage for this circuit contains 462 test vectors (with output selection implemented). The

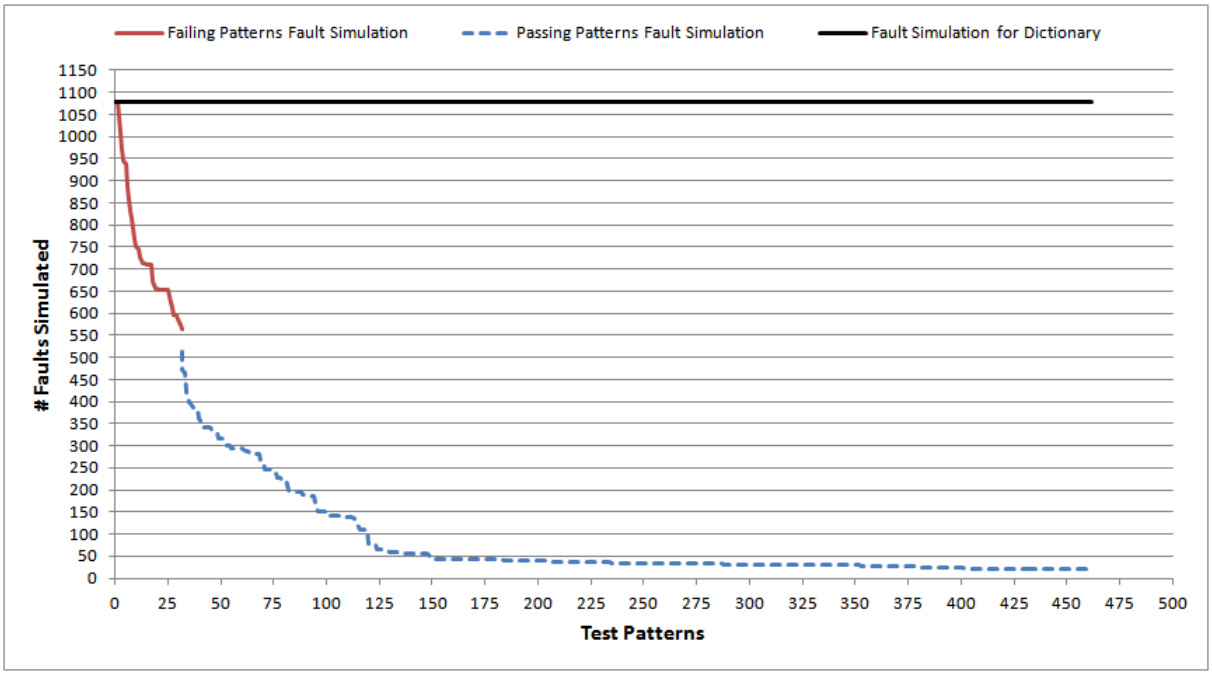


Figure 5.1: Simulation effort comparison with dictionary method applied to c432.

Table 5.1: A fault dictionary.

	t_0	t_1	t_2	t_3	t_4
F1:	1	1	1	0	0
F2:	1	0	1	0	0

dictionary method involves simulating all the faults for all the test vectors. Hence the entire area under the straight black line denotes the simulation effort of the fault dictionary method. The considered failure case produced 31 failing vectors (431 passing vectors). The proposed fault diagnosis procedure performs fault simulation with the failing test vectors first, which is denoted by the solid red line. This line drops down steep because, as and when the faults are detected, they are dropped. Hence we get to work with fewer faults as we proceed with the simulation. Next, the fault simulation of passing patterns is performed, which is denoted by the dotted blue line. It is to be noted that the faults that were detected and dropped during failing pattern simulation are the ones that are used for passing pattern simulation. In this case too, faults are dropped as and when they are detected by the passing patterns, which explains the drop in the line. Hence once again the number of faults to be simulated

keeps reducing throughout the simulation. Beyond a certain point, not many of the faults remaining are detected by the passing patterns, which makes the curve almost steady and flat. After simulating all the passing patterns, the faults that are remaining become the suspects and the surrogates. The area enclosed by both the lines (solid red and dotted blue) denote the simulation effort of the proposed diagnostic procedure which is far less than the traditional fault dictionary method.

For analyzing the effectiveness of each phase in the proposed diagnostic procedure, consider the dictionary entries or syndromes shown in Table 5.1.

The test pattern set in this example consists of 5 patterns (t_0, t_1, t_2, t_3 and t_4). The faults possible in the circuits are F1 and F2. As explained previously in chapter 3, the syndromes or signatures for the faults are formed by finding out which test pattern will be able to detect which of the faults. For example, if a test pattern is able to detect single fault F1, then a '1' is placed in the corresponding column for the fault F1. If test pattern is not able to detect single fault F1, then a '0' is placed in the corresponding column for the fault F1. For the purpose of simplifying the analysis let us assume there are no equivalent faults in the circuit.

Case 1: single fault F1

Let us consider that single fault F1 is the actual fault present in the circuit. So the output response of the faulty circuit will match the syndrome of F1 (11100), as in the dictionary. Hence the test vectors t_0, t_1 and t_2 are the failing vectors and test vectors t_3 and t_4 are the passing vectors. So Phase 1 of the proposed algorithm will produce the union of faults detected by all failing patterns. Since test patterns t_0 and t_2 detect both F1 and F2, the result of Phase 1 will be F1 and F2 in *sus_flts* set. Phase 2 of the algorithm will produce the union of the faults detected by the passing patterns and will remove these faults from the *sus_flts* set. In this case, Phase 2 will yield no faults. Phase 3 will produce the intersection of faults detected by all failing patterns and store it in *set1_can_flts* set. So here, it will come up with fault F1. The remaining faults from *sus_flts* set will be moved to *set2_can_flts* set,

i.e., fault F2 will be moved to *set2_can_flts* set. Phase 4 will add the equivalent and opposite polarity faults for faults present in each set. But here, since there are no equivalent faults, only opposite polarity of the faults F1 and F2 will be added to the corresponding fault sets. Hence the correct diagnosis is achieved in this case.

Case 2: single fault F2

The actual fault in this case is single fault F2. Hence the syndrome entry of the dictionary for fault F2 (10100) will match the output response of the faulty circuit. Phase 1 will produce F1 and F2 faults. Phase 2 will remove fault F1 from *sus_flts* set because the test vector t1 passes and it can detect fault F1. Now Phase 3 will produce F2, because there is only one fault in the *sus_flts* list. It is to be noted that if Phase 3 was performed ahead of Phase 1, it would have resulted in producing faults F1 and F2. That would have resulted in poor diagnostic resolution. Phase 4 will add the opposite polarity fault of F2. Hence we achieved perfect diagnosis in this case too.

From the above two cases, it can be inferred that Theorem1 is true. For a single fault situation, *set1_can_flts* set will always include the actual fault.

Case 3: multiple faults F1 and F2 (No Masking)

Consider faults F1 and F2 are present and they do not mask each other. The circuit will fail on all the patterns, on all the outputs where F1 or F2 can be detected. Hence the output response of the faulty circuit will be the OR-operation of the syndromes of both the faults F1 and F2, i.e., 11100. Now Phase 1 produces F1 and F2. Phase 2 produces no fault. So F1 and F2 are retained in *sus_flts* set. Phase 3 will produce fault F1. Hence the *set1_can_flts* set will have fault F1 and *set2_can_flts* set will have fault F2. Phase 4 will add the opposite polarity faults to the corresponding sets. Once again we achieve perfect diagnosis with the surrogate faults referring to the actual multiple faults. It is to be noted that fault F1 is reported as a prime suspect and F2 as a surrogate fault. So both the actual faults are detected.

Case 4: multiple faults F1 masking F2

If faults F1 and F2 are present and F1 masks F2, i.e., presence of fault F1 does not allow the effect of fault F2 to propagate to the output, and hence preventing it from being identified. This is because during fault propagation the effect of fault F2 will be overwritten by fault F1. Hence we end up with the syndrome of fault F1, 11100, which is the same as in case3. So *set1_can_flts* set will have fault F1 and *set2_can_flts* set will have fault F2, thus helping us to achieve perfect diagnosis.

Case 5: multiple faults F2 masking F1

When faults F2 and F1 are present and F2 masks F1, the syndrome will be 10100, which is the same as in case 2. But the result of this will be that only fault F2 will be identified and fault F1 will be removed from suspicion. In this case, we do not achieve a perfect multiple fault diagnosis the first time the algorithm is run. But once the single fault F2 is identified, it can be electron beam probed and rectified when found faulty. Now the case will be that the single fault F1 alone will be present in the circuit, which can be easily identified by the algorithm. The shortcoming of this is that the testing of the faulty chip and the algorithm have to be run more than once to detect both the faults in such cases.

Case 6: multiple faults F1 interfering with F2 (0 to 1)

Consider both faults F1 and F2 are present and F1 interferes with fault F2, i.e., the presence of F1 causes a passing test pattern of fault F2 to fail or a failing test pattern of F2 to pass. Thus changing a '0' in the syndrome of fault F2 to '1' or vice-versa. Let us assume that the passing test pattern t_3 for fault F2, fails, due to the presence of fault F1. Now the output response of the failing circuit will be 11110. Working with this information, the Phase 1 will produce, faults F1 and F2. Phase 2 will produce no faults, thereby *sus_flts* set retaining F1 and F2. Phase 3 will also produce no results, since there is no fault that will be detected commonly by all the failing test patterns. Now *set1_can_flts* will contain no faults and *set2_can_flts* will contain both faults F1 and F2, thereby aiding to achieve perfect diagnosis.

Since, from Theorem1, we already know that, for a single fault situation, *set1_can_flts* will always contain the actual single fault, we can be sure that any case which produces no faults in *set1_can_flts* will be a multiple fault situation.

Case 7: multiple faults F2 interfering with F1 (1 to 0)

If presence of fault F2 interferes with the syndrome of fault F1 and causes the failing test pattern t_0 of fault F1 to pass, then the resulting fault circuit response will be 11100, which is the same as in case1. Hence the diagnosis procedure will come up with fault F1 and its opposite polarity fault in *set1_can_flts* set and fault F2 and its opposite polarity fault in *set2_can_flts* set. Here too, we achieve perfect diagnosis.

The seven cases above include all possibilities of two-fault combinations, including masking and interference. The very same analysis can be extended to any number of multiple fault combinations and the diagnosis procedure will be able to come up with the actual faults or surrogate faults that might represent the actual faulty behavior. The reason for considering only 2 fault combinations is explained in chapter 6.

To understand the working of each phase of the diagnosis procedure let us consider a couple of failure cases in the C17 benchmark circuit. The C17 circuit has a total of only 22 single stuck-at faults and hence we can get 100% detection coverage with just 5 test vectors - 01111, 10101, 10011, 10000 and 01000. These vectors are each duplicated into two vectors for output selection. Hence the test vector set will contain 0111110 (t_0), 0111101 (t_1), 1010110 (t_2), 1010101 (t_3), 1001110 (t_4), 1001101 (t_5), 1000010 (t_6), 1000001 (t_7), 0100010 (t_8) and 0100001 (t_9).

The fault dictionary with collapsed fault list for circuit C17 is shown in table 5.2. The first column shows collapsed faults with their polarities. For example, '1 /NAND2_1/OUT' represents the output of gate 'Nand2_1' stuck-at-0.

Consider the single stuck-at fault shown in figure 5.2. The output of gate 'Nand2_1' is stuck-at-1. The observed syndrome for this defective circuit is `0010000000`. This says that only test pattern t_2 produces a failing response. In phase 1 of the diagnosis procedure, we

Table 5.2: C17 fault dictionary.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
1 /NAND2_1/OUT:	0	0	1	0	0	0	0	0	0	0
1 /N1 :	1	0	0	0	0	0	0	0	0	0
1 /NAND2_3/OUT:	0	0	0	0	0	0	0	0	1	1
1 /N2 :	0	0	0	0	1	0	1	1	0	0
0 /N3 :	1	1	1	0	0	0	0	0	0	0
1 /N3 :	0	0	0	0	1	1	1	0	0	0
1 /NAND2_2/OUT:	1	1	0	0	0	0	0	0	0	0
1 /N6 :	0	0	1	0	0	0	0	0	0	0
1 /NAND2_4/OUT:	0	0	0	1	0	1	0	0	0	0
1 /N7 :	0	0	0	0	0	0	0	1	0	0
1 /AND2_7/IN0 :	1	0	0	0	1	0	1	0	0	0
1 /NAND2_1/IN1:	0	0	0	0	1	0	1	0	0	0
0 /NAND2_2/OUT:	0	0	0	1	0	1	0	0	1	1
1 /NAND2_2/IN0:	0	0	0	0	0	1	0	0	0	0
1 /AND2_8/IN0 :	0	1	0	0	0	0	0	1	0	0
1 /NAND2_4/IN0:	0	1	0	0	0	0	0	0	0	0
0 /NAND2_3/OUT:	1	1	0	0	1	0	1	1	0	0
1 /NAND2_3/IN1:	1	1	0	0	0	0	0	0	0	0
0 /AND2_8/IN0 :	0	0	0	1	0	1	0	0	0	1
1 /NAND2_6/IN0:	0	0	0	0	0	0	0	0	0	1
0 /AND2_7/IN0 :	0	0	1	0	0	0	0	0	1	0
1 /NAND2_5/IN1:	0	0	0	0	0	0	0	0	1	0

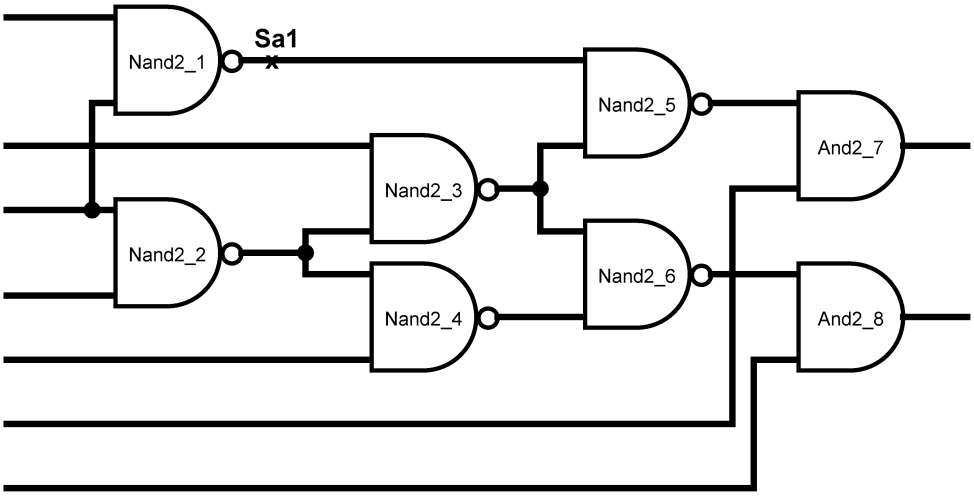


Figure 5.2: Single stuck-at fault case.

perform fault simulation of failing pattern t_2 (1010110) and put all the faults that it can detect in `sus_flts` set. In phase 2, we perform fault simulation of all the passing patterns and

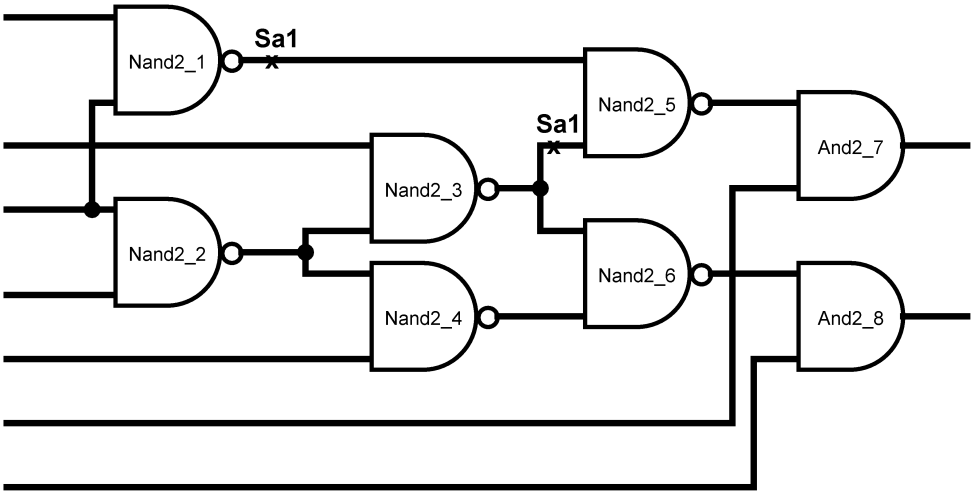


Figure 5.3: Multiple stuck-at fault case where both faults are diagnosed.

remove any faults that can be detected, from the `sus_flts` set. There is no need for phase 3, since there is only one failing pattern. Hence all the faults remaining in the `sus_flts` set is moved to `set1_can_flts` set and are called prime suspects. There are no `set2_can_flts` in this case. Phase 4 adds the equivalent faults and opposite polarity faults. It is noticed that only test pattern t2 can detect the actual fault. Rest of the nine test patterns in the test set are not able to detect the fault. If any other test pattern was able to detect the fault, it could not have been a passing test pattern. Hence the actual fault is diagnosed.

Now let us consider the multiple (two) stuck-at faults case shown in figure 5.3. The output of gate 'Nand2_1' is stuck-at-1 and the second input of gate 'Nand2_5' is stuck-at-1. The observed syndrome for this defective circuit is `0010000010`. This says that only test patterns t_2 and t_8 produce a failing response. In phase 1 of the diagnosis procedure, we perform fault simulation of failing patterns t_2 (1010110) and t_8 (0100010) and put all the faults that they can detect in `sus_flts` set. In phase 2, we perform fault simulation of all the passing patterns and remove any faults that can be detected, from the `sus_flts` set. In phase 3, we perform fault simulation of failing patterns t_2 and t_8 and get the faults (0 /AND2_7/IN0) that are commonly detected. These common faults are moved to `set1_can_flts` set and are called prime suspects. The remaining faults in `sus_flts` set moved to `set2_can_flts`

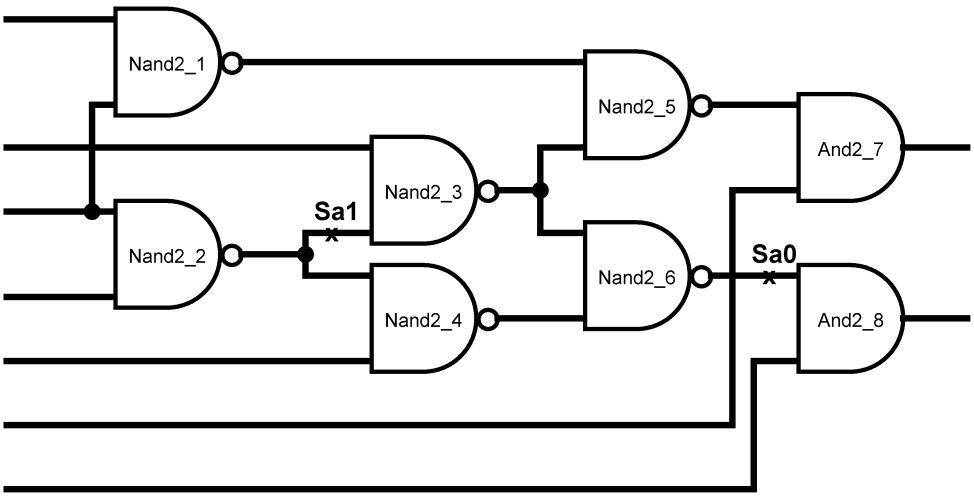


Figure 5.4: Multiple stuck-at fault case where both faults are diagnosed.

and are called surrogate faults. Phase 4 adds the equivalent faults and opposite polarity faults. Here both the actual faults are reported as surrogates and hence the actual fault is diagnosed.

One other multiple (two) stuck-at faults case is shown in figure 5.4. The second input of gate 'Nand2_3' is stuck-at-1 and the first input of gate 'And2_8' is stuck-at-0. The observed syndrome for this defective circuit is `1001010001`. This says that test patterns t_0, t_3, t_5 and t_9 produce a failing response. In phase 1 of the diagnosis procedure, we perform fault simulation of failing patterns t_0(0111110), t_3(1010101), t_5(1001101) and t_9(0100001) and put all the faults that they can detect in `sus_flts` set. In phase 2, we perform fault simulation of all the passing patterns and remove any faults that can be detected, from the `sus_flts` set. In phase 3, we perform fault simulation of all the failing patterns and find that there are no faults that are commonly detected. Hence `set1_can_flts` set is left empty and there are no prime suspects. All the faults in `sus_flts` set are moved to `set2_can_flts` and are called surrogate faults. Phase 4 adds the equivalent faults and opposite polarity faults. Here only one of the actual faults (0 AND2_8/IN0) is reported by surrogates and the other fault or its surrogates are not reported in the final candidate list. This is because the fault (1 /NAND2_3/IN1) is detected by tests t_0 and t_1. But the syndrome of the defective circuit

says t_1 is a passing pattern and hence this fault is removed from suspicion. This is because of the presence of fault masking (similar to case 5). Here we achieve only partial diagnosis in this case.

5.1 Fault Ranking

There is a very small probability that the diagnosis procedure will come up with no results, i.e., will have zero faults in both SET1 and SET2. That will happen during the case where multiple faults with masking and interference are present in the circuit and they together, produce a faulty output response which will allow a few of the test patterns which will detect them to pass and other test patterns which will detect them to fail or vice-versa. But this is a very rare phenomenon. In such cases a ranking procedure is followed to come up with the surrogate faults.

After Phase 1 of the diagnostic procedure is performed, we have the short listed faults in *sus_flts*. But while performing Phase 1, we also keep track of the number of failing patterns that detect every fault. This number for each fault is added to the weight of the corresponding fault, i.e., if a fault F1 is detected by three failing patterns, then the weight of fault F1 at the end of Phase 1 is assigned to be three. Similarly, while performing Phase 2 of the diagnostic procedure, we keep track of the number of passing patterns that detect every fault. This number for each fault is subtracted from the weight of the fault, which was originally obtained at the end of Phase 1. At the end of Phase 2 we get the final weight of every fault. The faults with the highest weight are reported to be prime suspect (SET1) faults and the faults with the second highest weight are reported to be the surrogate (SET2) faults.

It is to be noted that the final weights can also be negative. This will happen when a fault causes more passing patterns and fewer failing patterns. Even in this case, the top two highest weights are considered to be the fault suspects. Also, there can be cases where

the final weight is zero. This will happen when the fault is detected by the same number of passing and failing patterns.

The diagnostic procedure returning with zero faults is a very rare possibility. Out of 550 tests performed on the benchmark circuits, the algorithm came up with zero results only twice. But this fault ranking system guards the algorithm even against such cases as well.

Chapter 6

Experimental Results

The algorithm was tested by performing experiments on ISCAS'85 benchmark circuits using various test pattern sets. The circuit modeling and the entire algorithm was implemented in Python programming language [2] which internally invokes ATPG and Fault Simulator of Mentor Graphics FASTSCAN [1]. VBA Macros [27] were used to duplicate the test patterns for output selection. The tests were conducted on a personal computer (PC) with Intel Core-2 duo 3.06GHz processor and 4GB memory.

Results for every circuit were obtained by calculating the average values obtained from two separate runs of experiments, each containing 50 different random failure cases (except for circuit C17, which has only 22 faults).

The results of single fault diagnosis using a 1-detect pattern set are shown in Table 6.1. The first column states the circuit name, the second column contains the number of primary outputs the circuit has. The third column shows the number of patterns (with output selection implemented) used for diagnosis. So the actual number of patterns in the 1-detect pattern set for any circuit will be the number of patterns shown in the third column divided by the number of primary outputs (shown in column 2) of the table.

Diagnostic Coverage of the test pattern set based on single stuck-at-faults, excluding redundant faults, is stated in column 4. From [43], diagnostic coverage (DC) is defined as

$$DC = \frac{Number\ of\ detected\ fault\ groups}{Total\ number\ of\ faults} = \frac{n}{N} \tag{6.1}$$

Column 5 shows the percentage of cases the single fault was diagnosed. For single stuck-at-faults, the algorithm always comes up with the actual fault (100% diagnosis), even if the diagnostic coverage of the pattern set is not as high. Simulation time in seconds is stated

Table 6.1: Single fault diagnosis with 1-detect tests.

Circuit name	No. of outputs	No. of patterns	DC %	Diagnosis %	CPU* s	Fault ratio	
						SET1	SET2
C17	2	10	95.454	100	0.067	1.100	1.780
C432	7	462	94.038	100	0.189	1.025	6.675
C499	32	2080	98.000	100	0.588	1.029	16.722
C880	26	1664	94.161	100	0.503	1.069	2.248
C1908	25	3625	85.187	100	1.294	1.379	28.290
C2670	140	13300	85.437	100	6.455	1.320	8.207
C3540	22	3520	89.091	100	1.333	1.229	5.200
C5315	123	13899	91.192	100	6.847	1.054	4.204
C6288	32	1056	85.616	100	0.764	1.138	8.255
C7552	108	17064	86.507	100	10.123	1.281	10.765

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

Table 6.2: Single fault diagnosis with 2-detect tests.

Circuit name	No. of outputs	No. of patterns	DC %	Diagnosis %	CPU* s	Fault ratio	
						SET1	SET2
C499	32	3872	98.400	100	1.025	1.029	7.970
C1908	25	6425	86.203	100	2.242	1.379	14.798
C7552	108	27756	86.750	100	16.076	1.281	8.023

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

in column 6. The ratio of number of candidate faults in SET1 and SET2 are reported in columns 7 and 8, respectively. This ratio is nothing but the ratio of the total number of faults reported in each set to the number of faults expected in that set. The expected number of faults to be reported for an actual fault includes the actual fault, its equivalent faults and the opposite polarity faults for all equivalent faults, including the actual fault. This ratio denotes the diagnostic resolution of the procedure. The closer the fault ratio is to 1.0, the better the resolution. For single stuck-at-faults, the ratio of SET1 faults is almost 1.0 in all the cases. Hence, when the faults suggested in SET1 are probed (by electron beam or other procedures), one would identify the actual fault and it will not be necessary to probe the faults suggested in SET2. But since we do not know whether the actual fault is a single stuck-at fault or a multiple stuck-at fault, the SET2 surrogate faults cannot be completely neglected.

Table 6.3: Multiple fault diagnosis with 1-detect tests.

Circuit name	No. of Patterns	DC %	Both faults diagnosed (%)	One fault not diagnosed (%)	Both faults not diagnosed (%)	CPU* s	Fault ratio	
							SET1	SET2
C17	10	95.454	80.950	19.040	0.000	0.067	0.500	2.091
C432	462	94.038	90.566	7.547	1.886	0.135	0.563	3.516
C499	2080	98.000	49.056	20.754	30.188	0.613	0.371	17.589
C880	1664	94.161	86.792	9.433	3.773	0.502	0.900	3.205
C1908	3625	85.187	90.566	0.000	9.433	0.928	0.488	12.764
C2670	13300	85.437	88.679	3.773	7.547	4.720	0.564	7.046
C3540	3520	89.091	86.792	3.773	9.433	1.547	0.488	5.177
C5315	13899	91.192	98.113	1.886	0.000	7.065	0.422	3.886
C6288	1056	85.616	83.018	0.000	16.981	0.888	0.589	5.536
C7552	17064	86.507	96.226	1.886	1.886	7.539	0.358	7.104

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

For circuits C499, C1908 and C7552, the ratio of faults in SET2 is high. This is due to the fact that the diagnostic coverage of the test pattern set is not enough. To prove that improving diagnostic coverage of the test pattern set will improve the diagnostic resolution, 2-detect test patterns were used to diagnose the above mentioned three circuits. The results are shown in Table 6.2.

It can be noticed that the use of 2-detect patterns for diagnosis has increased the diagnostic coverage of the patterns by 1.016% maximum, for circuit C1908. But the resolution has improved almost by 50%. So for patterns with even higher diagnostic coverage, the resolution will be improved more. So the utmost efficiency of the diagnosis algorithm can be obtained by using a higher diagnostic capability test pattern set than just the detection test pattern set.

To test the relevance of reported surrogate faults to the actual non-classical faults, similar tests were conducted for multiple stuck-at-faults by introducing two stuck-at-faults simultaneously and considering 50 such different failure cases for each circuit. The two stuck-at-faults were chosen in such a way that, they are close to each other in the circuit. Also, the reason for considering only two faults to be present in the circuit simultaneously is that the probability of fault masking is maximum only when there are just two faults in

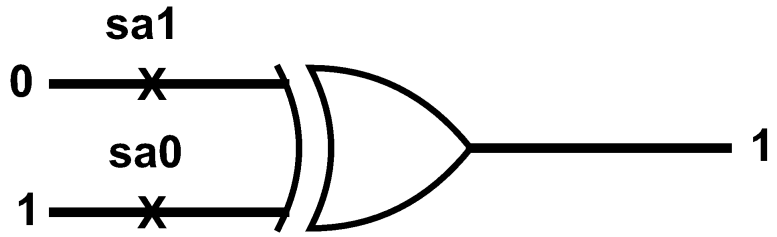


Figure 6.1: Fault masking (interference) in XOR gate.

the circuit and the probability keeps reducing when the number of faults present, increases. This is done to increase the chances of fault masking, to create a pessimistic environment for the algorithm to perform. The experiment was run twice and the average of the results was taken. Table 6.3 shows the multiple fault diagnosis performed with 1-detect test pattern set.

Since we are using the same 1-detect test pattern set as the one used for single fault diagnosis, there is no change in the number of patterns in the test pattern set or the diagnostic coverage. Column 4 of Table 6.3 shows the percentage of cases where both faults were diagnosed. Column 5 shows the percentage of cases where only one of the actual faults present was diagnosed. The sum of these two percentages subtracted from 100% gives the percentage of cases where both faults were not diagnosed, as shown in column 6. As the results in the table indicate, except for the circuit C499, all other circuits have, at least in 80% cases, a perfect diagnosis of both faults. A point to be noted is that the proposed diagnosis procedure does not assume that fault masking is not present and the reported percentage of diagnosis includes the possible fault masking and interference cases.

The reason for Circuit C499 (32-bit Single-Error-Correcting circuit) producing poor multiple fault diagnosis (resulting in irrelevant surrogate faults) even with a test pattern set having very high diagnostic coverage (based on single stuck-at-faults) must be examined. We found the presence of circular fault masking in many of the fault cases considered. The circuit has an XOR tree consisting of 104 two-input XOR gates. XOR logic gates are not considered

to be elementary logic gates since they are generally constructed from multiple Boolean gates, such that the set of faults depends on its construction. Yet, all four test patterns are needed to completely test a 2-input XOR gate, regardless of its construction [36]. Consider the XOR gate shown in Figure 6.1. The input 1 has a stuck-at-1 fault and input 2 has a stuck-at-0 fault. To propagate a single fault through XOR gate, the other input must be unchanged. But since this is a multiple fault situation, two faults are trying to propagate through the same XOR gate at the same time. So a '0' on input 1 is required to activate the stuck-at-1 fault and a '1' on input 2 is required to activate the stuck-at-0 fault. But since both the inputs are changed, the faults mask each other. This phenomenon is called circular masking. Hence the output is '1' which is the same as the good circuit output. Due to this circular masking, the algorithm will not be able to produce the relevant surrogate faults as the actual faults are dropped, since the pattern which should have failed, passes. This is not only for the case where both faults are present on the inputs of the XOR gate, but also for any case where both the actual faults are being propagated through the same XOR gate. The situation will improve while considering more than two faults to be present in the circuit because the probability of a complete circular masking decreases with the increase in the number of faults. In circuit C499, the presence of this huge XOR tree increases the circular masking and thereby deteriorates the performance of the algorithm. But as discussed before, the algorithm is able to produce such results even in a highly pessimistic environment, where choices (close neighborhood faults selected) are made in such a way that the probability of masking is high. One other ISCAS'85 benchmark circuit, which has (2-input) XOR gates present, is circuit C432. But it has only 18 XOR gates, which do not form a tree and hence the diagnostic percentage is not hurt significantly.

The ratio of faults present in SET1 is less than 1 because in most cases, faults reported in SET1 include one of the actual faults, its equivalent faults and the opposite polarity faults. The other actual fault, its equivalent faults and opposite polarity faults are present in SET2.

Table 6.4: Multiple (two) fault diagnosis with 2-detect tests.

Circuit name	No. of patterns	DC (%)	Faults diagnosed (%)			CPU* s	Fault ratio	
			Both	One	None		SET1	SET2
C499	3872	98.400	49.056	20.754	30.188	0.696	0.371	11.555
C1908	6425	86.203	90.566	0.000	9.433	2.314	0.488	7.232
C7552	27756	86.750	96.226	1.886	1.886	17.291	0.358	5.905

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

Table 6.5: Single fault diagnosis with diagnostic patterns.

Circuit name	No. of outputs	No. of patterns	DC %	Diagnosis %	CPU* s	Fault ratio	
						SET1	SET2
C17	2	12	100	100	0.067	1.000	1.780

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

Hence, the resolution of SET1 faults is always closer to 0.5 than being 1 while considering two faults.

The same three circuits show a comparatively poorer SET2 resolution. Hence, 2-detect patterns are used to show that the diagnostic resolution improves on improving the diagnostic coverage of the test pattern set. The results of this experiment are shown in Table 6.4.

Once again it is seen that, for small increase in diagnostic coverage of the patterns by 1.016% (maximum) for circuit C1908 the resolution is improved by almost 40%. Other two circuits show a similar trend.

The last experiment was to try the diagnosis procedure on a 100% diagnostic test pattern set. The circuit C17 reports 95.454% of diagnostic coverage (DC) with as few as 5 patterns that have 100% detection coverage. The total number of faults in the circuit is 22. There was only one fault pair that was not distinguished. Adding one more pattern that distinguishes the fault pair yielded 100% diagnostic coverage as expected. The diagnostic algorithm was then run using this test pattern set to yield the results shown in Tables 6.5 and 6.6.

The single fault diagnosis with 100% diagnostic coverage vector set, produced perfect diagnostic resolution '1.0' as expected in SET1, and a slightly improved resolution in SET2.

Table 6.6: Multiple fault diagnosis with diagnostic patterns.

Circuit name	No. of patterns	DC %	Both faults diagnosed (%)	One fault not diagnosed (%)	Both faults not diagnosed (%)	CPU* (s)	Fault ratio	
							SET1	SET2
C17	12	100	80.952	19.047	0.000	0.067	0.489	2.102

$*$ PC with Intel Core-2 duo 3.06GHz processor and 4GB memory

The multiple fault diagnosis with this test pattern set improved the resolution in SET1 by a very small amount and decreased the resolution of SET2 by the very same amount.

Also, the diagnostic coverage was improved by a very small percentage. Since the 1-Detect test pattern set already had a diagnostic coverage of 95.454, there was very little left to improve.

To sum up, the proposed diagnostic procedure, given a failing vector, if the cause is a single stuck-at-fault, will always come up with the actual fault, irrespective of the detection or diagnostic coverage of the test pattern set. If the detection coverage of the test pattern is higher, higher will be the resolution of the faults reported. Provided with 100% diagnostic coverage, the maximum resolution can be achieved. If the actual fault is a multiple stuck-at-fault without circular fault masking, the diagnostic procedure will come up with surrogate faults that represent the actual faults or the behavior of the actual faults, with higher resolution as the diagnostic coverage of the pattern set increases.

Chapter 7

Conclusion

Physical defects are not always classical single stuck-at faults. Even though most of the physical defects can be modeled by single stuck-at faults, it is becoming more difficult due to the shrinking of device features. Hence this work puts forward a diagnostic algorithm to diagnose non-classical faults with the single stuck-at fault simulation information.

The proposed diagnostic algorithm is of lower complexity and works efficiently even when fault masking or fault interference is present. It has higher diagnosability and resolution for single stuck-at-faults, even if provided with a test pattern set having nominal detection coverage. The same trend will be exhibited for the surrogate faults produced to represent multiple stuck-at-faults without circularly masking each other, when the diagnostic coverage of the test pattern set is increased. The algorithm is memory efficient, since it does not require a dictionary and also has reduced diagnostic effort (CPU time), since it works on relatively smaller number of fault suspects and does not require re-running simulations after frequently moving faults to and from the suspected fault list based on heuristics.

The cases where the diagnostic procedure came up with candidate faults or surrogates which were no way related to the actual fault, need to be inspected closely. This happened because the reported faults or surrogates were able to produce the same function as the actual faults present. Hence this is valuable information and must be investigated.

Considering that fault simulation tools will always be limited to a few fault models (e.g., single stuck-at or transition faults), we should explore the relationships between non-classical faults (bridging, stuck-open, coupling, delay, etc.) and the corresponding surrogate classical representatives. For example, some non-classical faults like stuck-open or bridging require an initialization pattern to precede a stuck-at test pattern. Thus, the test result for the

non-classical fault agrees with a single stuck-at fault only on a subset of patterns. Further analysis can establish better correlation between actual faults and their surrogates.

Bibliography

[1] *ATPG and Failure Diagnosis Tools*. Mentor Graphics Corp., Wilsonville, OR, 2009.

[2] *Python Tutorial Release 2.6.3. docs@python.org*. Python Software Foundation, 2009.

[3] M. Abramovici and M. A. Breuer, "Fault Diagnosis Based on Effect-Cause Analysis: An Introduction," in *Proc. 17th Design Automation Conf.*, June 1980, pp. 69–76.

[4] M. Abramovici and M. A. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," *IEEE Transactions on Computers*, vol. C-29, no. 6, pp. 451–460, June 1980.

[5] V. D. Agrawal, D. H. Baik, Y. C. Kim, and K. K. Saluja, "Exclusive Test and Its Applications to Fault Diagnosis," in *Proc. 16th International Conf. VLSI Design*, Jan. 2003, pp. 143–148.

[6] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre, "Fault Collapsing Via Functional Dominance," in *Proc. International Test Conf.*, Oct. 2003, pp. 274–280.

[7] C. Alagappan and V. D. Agrawal, "Dictionary-Less Defect Diagnosis as Real or Surrogate Single Stuck-At Faults," in *Proc. International Test Conf.*, 2013. *Submitted*.

[8] C. Alagappan and V. D. Agrawal, "Dictionary-Less Defect Diagnosis as Surrogate Single Stuck-At Faults," in *Proc. 22nd North Atlantic Test Workshop*, 2013.

[9] T. Bartenstein, D. Heaberlin, L. Huisman, and D. Sliwinski, "Diagnosing Combinational Logic Designs Using the Single Location at-a-Time (SLAT) Paradigm," in *Proc. International Test Conf.*, 2001, pp. 287–296.

[10] M. Beckler and R. D. (Shawn) Blanton, "On-Chip Diagnosis for Early-Life and Wear-Out Failures," in *Proc. International Test Conf.*, Nov. 2012, pp. 1–10.

[11] V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and P. Bollineni, "Multiple Error Diagnosis Based on Xlists," in *Proc. 36th Design Automation Conference*, 1999, pp. 660–665.

[12] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Boston: Springer, 2000.

[13] C.-Y. Cao, C.-H. Liao, and C. H.-P. Wen, "Diagnosing Multiple Byzantine Open-Segment Defects Using Integer Linear Programming," *J. Electronic Testing: Theory and Applications*, vol. 27, no. 6, pp. 723–739, Dec. 2011.

[14] H. Y. Chang, E. Manning, and G. Metze, *Fault Diagnosis of Digital Systems*. Florida: R. E. Krieger Publishing Company, 1974.

[15] H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 7, pp. 813–833, July 1988.

[16] A. L. D'Souza and M. S. Hsiao, "Error Diagnosis of Sequential Circuits Using Region-Based Model," in *Proc. 14th International Conference on VLSI Design*, 2001, pp. 103–108.

[17] M. R. Grimaila, S. Lee, J. Dworak, K. M. Butler, B. Stewart, B. Houchins, V. Mathur, J. Park, L.-C. Wang, and M. R. Mercer, "REDO - Random Excitation and Deterministic Observation - First Commercial Experiment," in *Proc. 17th IEEE VLSI Test Symp.*, Apr. 1999, pp. 268–274.

[18] S. Holst and H.-J. Wunderlich, "Adaptive Debug and Diagnosis Without Fault Dictionaries," *J. Electronic Testing: Theory and Applications*, vol. 25, no. 4, pp. 259–268, Aug. 2009.

[19] J. L. A. Hughes, "Multiple Fault Detection Using Single Fault Test Sets," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 100–108, Jan. 1988.

[20] L. M. Huisman, *Data Mining and Diagnosing IC Fails*. NewYork: Springer, 2005.

[21] R. C. Jaeger and T. N. Blalock, *Microelectronic Circuit Design, 4th Edition*. Ohio: McGraw Hill Higher, 2010.

[22] S. Jo, T. Matsumoto, and M. Fujita, "SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions," in *Proc. 21st IEEE Asian Test Symposium*, Nov. 2012, pp. 19–24.

[23] Y. Karkouri, E. M. Aboulhamid, E. Cerny, and A. Verreault, "Use of Fault Dropping for Multiple Fault Analysis," *IEEE Transactions on Computers*, vol. 43, no. 1, pp. 98–103, Jan. 1994.

[24] Y. C. Kim, V. D. Agrawal, and K. K. Saluja, "Multiple Faults: Modeling, Simulation and Test," in *Proc. 15th International Conf. VLSI Design and Proc. 7th Asia and South Pacific Design Automation Conf.*, Jan. 2002, pp. 592–597.

[25] B. Klenke, "Test Technology Overview Module 43." `http://ares.cedcc.psu.edu/ee497i/rassp_43/sld010.htm`, 1998. [Online; accessed 12-March-2013].

[26] D. Knebel, P. Sanda, M. McManus, J. A. Kash, J. C. Tsang, D. Vallett, L. Huisman, P. Nigh, R. Rizzolo, P. Song, and F. Motika, "Diagnosis and Characterization of Timing-Related Defects by Time-Dependent Light Emission," in *Proc. International Test Conference*, Oct. 1998, pp. 733–739.

[27] M. Kofler, *Definitive Guide to Excel VBA*. New York: Apress, 2000.

[28] S. D. Millman, E. J. McCluskey, and J. M. Acken, "Diagnosing CMOS Bridging Faults With Stuck-At Fault Dictionaries," in *Proc. International Test Conf.*, Sept. 1990, pp. 860–870.

[29] I. Pomeranz and S. M. Reddy, "On the Generation of Small Dictionaries for Fault Location," in *Proc. IEEE/ACM International Conference on Computer-Aided Design*, Nov. 1992, pp. 272–279.

[30] I. Pomeranz and S. M. Reddy, "On Correction of Multiple Design Errors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 2, pp. 255–264, Feb. 1995.

[31] S. M. Reddy, I. Pomeranz, and S. Kajihara, "On the Effects of Test Compaction on Defect Coverage," in *Proc. 14th IEEE VLSI Test Symp.*, Apr. 1996, pp. 430–435.

[32] E. M. Rudnick, W. K. Fuchs, and J. H. Patel, "Diagnostic Fault Simulation of Sequential Circuits," in *Proc. International Test Conference*, Sept. 1992, pp. 178–186.

[33] P. G. Ryan, S. Rawat, and W. K. Fuchs, "Two-Stage Fault Location," in *Proc. International Test Conference*, Oct. 1991, pp. 963–.

[34] J. W. Sheppard and W. R. Simpson, *Research Perspectives and Case Studies in System Test and Diagnosis*. Boston: Springer, 1998.

[35] N. Sridhar and M. S. Hsiao, "On Efficient Error Diagnosis of Digital Circuits," in *Proc. International Test Conference*, 2001, pp. 678–687.

[36] C. E. Stroud, *A Designer's Guide to Built-in Self-Test*. Boston: Springer, 2002.

[37] H. Takahashi, K. O. Boateng, K. K. Saluja, and Y. Takamatsu, "On Diagnosing Multiple Stuck-At Faults Using Multiple and Single Fault Simulation in Combinational Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 3, pp. 362–368, Mar. 2002.

[38] R. Ubar, S. Kostin, and J. Raik, "Multiple Stuck-at Fault Detection Theorem," in *Proc. IEEE 15th International Symp. Design and Diagnostics of Electronic Circuits and Systems*, Apr. 2012, pp. 236–241.

[39] A. Veneris, "Fault Diagnosis and Logic Debugging Using Boolean Satisfiability," in *Proc. 4th International Workshop on Microprocessor Test and Verification*, May 2003, pp. 60–65.

[40] S. Venkataraman and S. B. Drummonds, "POIROT: A Logic Fault Diagnosis Tool and Its Applications," in *Proc. International Test Conf.*, 2000, pp. 253–262.

[41] L. C. Wang, T. W. Williams, and M. R. Mercer, "On Efficiently and Reliably Achieving Low Defective Part Levels," in *Proc. International Test Conf.*, Oct. 1995, pp. 616–625.

[42] Y. Zhang and V. D. Agrawal, "A Diagnostic Test Generation System," in *Proc. International Test Conf.*, Nov. 2010. Paper 12.3.

[43] Y. Zhang and V. D. Agrawal, "An Algorithm for Diagnostic Fault Simulation," in *Proc. 11th Latin-American Test Workshop (LATW)*, Mar. 2010, pp. 1–5.

[44] Y. Zhang and V. D. Agrawal, "Reduced Complexity Test Generation Algorithms for Transition Fault Diagnosis," in *Proc. International Conf. Computer Design*, Oct. 2011, pp. 96–101.

[45] L. Zhao and V. D. Agrawal, "Net Diagnosis Using Stuck-At and Transition Fault Models," in *Proc. 30th IEEE VLSI Test Symp.*, Apr. 2012, pp. 221–226.