A Design Methodology for Physical Design for Testability

Salahuddin A. Almajdoub

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

APPROVED:

Scott F. Midkiff, Chairman

James R. Armstrong

Aicha A. Elshabini-Riad

Hanif D. Sherali

Joseph G. Tront

July, 1996

Blacksburg, Virginia

Keywords: Physical Design for Testability, Bridging Faults, I_{DDQ} Testing

A Design Methodology for Physical Design for Testability

Salahuddin A. Almajdoub

Committee Chairman: Scott F. Midkiff

The Bradly Department of Electrical Engineering

(ABSTRACT)

Physical design for testability (PDFT) is a strategy to design circuits in a way to avoid or reduce realistic physical faults. The goal of this work is to define and establish a specific methodology for PDFT. The proposed design methodology includes techniques to reduce potential bridging faults in complementary metal-oxide-semiconductor (CMOS) circuits. To compare faults, the design process utilizes a new parameter called the fault index. The fault index for a particular fault is the probability of occurrence of the fault divided by the testability of the fault. Faults with the highest fault indices are considered the worst faults and are targeted by the PDFT design process to eliminate them or reduce their probability of occurrence.

An implementation of the PDFT design process is constructed using several new tools in addition to other "off-the-shelf" tools. The first tool developed in this work is a testability measure tool for bridging faults. Two other tools are developed to eliminate or reduce the probability of occurrence of bridging faults with high fault indices. The row enhancer targets faults inside the logic elements of the circuit, while the channel enhancer targets faults inside the routing part of the circuit.

To demonstrate the capabilities and test the effectiveness of the PDFT design process, this work conducts an experiment which includes designing three CMOS circuits from the ISCAS 1985 benchmark circuits. Several layouts are generated for every circuit. Every layout, except the first one, utilizes information from the previous layout to minimize the probability of occurrence for faults with high fault indices. Experimental results show that the PDFT design process successfully achieves two goals of PDFT, providing layouts with fewer faults and minimizing the probability of occurrence of hard-to-test faults. Improvement in the total fault index was about 40 percent in some cases, while improvement in total critical area was about 30 percent in some cases. However, virtually all the improvements came from using the row enhancer; the channel enhancer provided only marginal improvements.

ACKNOWLEDGEMENTS

All gratitude are due to *Allah*, God the Almighty, who created me, who guides me, who provides me with sustenance and whenever I fall ill, it is He who heals me.

I wish to express my sincere thanks to Dr. Scott F. Midkiff, my advisor and committee chairman for his continuous support and assistance. Without his patience and cooperation throughout this research, it would not have been possible to finish this project. I will always be grateful to Dr. Midkiff who taught me to always look at the "big picture."

I would also like to thank Dr. James R. Armstrong, Dr. Aicha A. Elshabini-Riad, Dr. Hanif D. Sherali, and Dr. Joseph G. Tront for serving on my committee and for their help. Special thanks go to Dr. Sherali for his help on the optimization theory. Dr. Dong S. Ha served in the original committee before his sabbatical and provided several corrections to the dissertation proposal.

I am grateful to Dr. F. J. Ferguson and Alvin Jee on the west coast for providing me with latest version of CARAFE and for their prompt answering my questions by email. Also I would like to thank K. Kozminski for providing the ISCAS 1985 benchmark circuits layout.

I also would like to thank my colleagues Mike Montgomery, David Lee, Rhett Hudson who helped me demystify the UNIX operating system, and Todd Flemming for his help on C++ programming.

I dedicate this work to my wife and my mother. I am in debt to their unlimited support and patience throughout the course of this work.

Finally, I thank all my friends in Blacksburg and back home who supported me with their love, care and prayers.

This work was supported in part by the University of Bahrain.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Physical design for testability (PDFT) is a strategy to design circuits in a way to avoid or reduce realistic physical faults. Physical faults include bridging faults, break (open) faults, transistor stuck-on and transistor stuck-off. Compared to traditional gate-level stuck-at faults, physical faults more closely represent realistic faults appearing at the gate level and switch level. Physical design for testability is still a new and emerging area.

Unfortunately, little work is published about physical design for testability. Most of the published work is limited to a specific part of PDFT and does not present a comprehensive approach to the PDFT concept. The only work that gives an overview of the whole area is by Ferguson [25]. Ferguson defined the concept and stated the goals of PDFT. However, Ferguson presented the goals in an abstract manner and did not state how to achieve these goals.

The goal of this work is to define and establish a specific methodology for PDFT, thus contributing to the overall concept of physical design for testability and providing a systematic means to accomplish the goals of PDFT. A new methodology to design complementary metal-oxide-semiconductor (CMOS) circuits is developed. This methodology includes techniques to reduce potential bridging faults, a common type of physical fault in CMOS. The design process targets bridging faults with high probability of occurrence and low testability. The new process is intended to produce reliable, easily testable CMOS circuit designs.

Three CMOS circuits are designed using this new design process to study the behavior of the process and to demonstrate its capabilities. The initial design of each circuit is generated first, then, using feedback information, the design process generates new designs. Every new design should be

more testable that the previous, but with a paid price in terms of area and delay. The experiment shows that the proposed design process allows the designer to navigate in the design space and explore alternative designs with different attributes.

An implementation of the new PDFT design process is constructed using several new tools in addition to other off-the-shelf tools. This implementation is based on MCNC's computer-aided design (CAD) system, Open Architecture Silicon Implementation Software (OASIS) [65]. Several tools are attached to OASIS to produce the desired design process. Some of the attached tools were developed in prior research, like the bridging fault extractor and the bridging fault test generator. Other tools are developed for this work, like the testability measure tool for bridging faults. This work also developed two other tools that try to eliminate or reduce the probability of occurrence of bridging faults with high probability of occurrence and low testability. The two tools differ in the type of faults they target. The first tool, the row enhancer, targets faults inside the logic elements of the circuit. The second tool, the channel enhancer, targets faults inside the routing part of the circuit.

A missing block in previous PDFT work is a testability measure for bridging faults. This work defines a testability measure for bridging faults by extending traditional testability measures defined for stuck-at faults to bridging faults. Testability is usually defined in terms of controllability and observability, however, supply current, or I_{DDQ}, testing grants observability. Therefore, the testability measure defined here is based only on controllability. This work defines the derived testability measure for bridging faults as an extension of the traditional controllability measures defined for stuck-at faults. Three of the traditional methods are extended to bridging faults and then compared to select the one most suitable for bridging faults. A new tool that calculates the testability of potential bridging faults is used in the new PDFT design process to identify bridging faults with low testability.

PDFT is divided into two parts: logic and interconnection. The logic physical design for testability deals with faults occurring inside logic elements like standard cells and logic arrays. The second category deals with faults occurring in the routing part of the circuit or due to placement. Two tools are developed in this work, one to handle the logic bridging faults and the other to handle the interconnection bridging faults. CMOS circuits studied in this work consist of several rows of horizontally abutted standard cells. Routing channels, located above and below each row,

connect each row to the rows above and below. This arrangement suggests naming the tool that handles logic bridging faults as the row enhancer and the tool that handles the channel bridging faults as the channel enhancer.

The goal of the channel enhancer is to minimize the probability of occurrence of bridging faults with high probability of occurrence and low testability that occur inside the routing channel. The channel enhancer consists of a channel router and a post-processor. The router uses an extended version of the net merging channel routing algorithm that is extended to handle cyclic cases. The router utilizes a new track assignment scheme that minimizes the probability of occurrence of bridging faults with low testability between horizontal wires in the channel. The channel enhancer also includes a via shifting post-processor that minimizes critical area.

The row enhancer is the feedback element in the PDFT design process. The row enhancer reads the list of bridging faults and tries to eliminate or reduce the probability of occurrence of faults with high probability of occurrence and low testability. The row enhancer only deals with faults occurring inside the logic part of the circuit. The row enhancer minimizes the probability of occurrence of some logic faults by replacing the fault-related gates with defect-tolerant versions. To facilitate the elimination or reduction of the critical area of bridging faults inside the cells, a defect-tolerant cell library is designed. The defect-tolerant cell library is derived from a standard cell library by expanding the area of the cells and then redesigning the cells to reduce the critical area. In most cases the defect-tolerant version minimizes or totally eliminates the critical area located inside the normal version. The row enhancer produces a list of gates that should use the defect-tolerant version in the next circuit layout.

This work contributes to PDFT by proposing a new design process that incorporates the handling of realistic faults into the design process. The new design process provides a systematic means to accomplish the goals of PDFT. Moreover, this work proposes several new tools that are integrated into a design suite to implement the new design process. Experiments to study the behavior and the capabilities of the new design process provide answers about benefits and limitations of PDFT.

The following chapter discusses bridging faults, I_{DDQ} testing, design for testability, and physical design for testability, and surveys prior research. Chapter 3 discusses the goals and scope of this work and then presents the new PDFT design process. An experiment that evaluates the

effectiveness and demonstrates the capabilities of the PDFT design process is proposed in Chapter 4. Chapter 5 describes a new testability measure for bridging faults. The channel enhancer tool is discussed in Chapter 6. Chapter 7 describes the defect-tolerant cell library used by the row enhancer. Chapter 8 presents the row enhancer program. Chapter 9 presents the experimental results. Conclusions and suggestions for future research are provided in Chapter 10.

Chapter 2

Background and Motivation

This chapter presents background on physical design for testability and discusses the problems that motivate this research. The chapter is divided into three parts.

The first part of the chapter describes bridging faults together with other CMOS fault models. Different methods to model bridging faults are discussed in Section 2.1. Section 2.1.3 presents several classifications for bridging faults. Section 2.1.4 introduces several realistic fault extraction programs. Then the I_{DDQ} testing method for bridging faults testing is presented in Section 2.1.5.

The second part of this chapter introduces design for testability. First, traditional testability measures, controllability and observability are presented together with several evaluation methods. Then, Section 2.2.2 discusses traditional gate-level design for testability methods. Physical design for testability is presented in Section 2.2.3.

The final part of this chapter summarizes the status of physical design for testability. This summary identifies open problems associated with this concept and research needed to more fully establish the concept of physical design for testability.

2.1 Bridging Faults

Bridging faults are shorts between normally unconnected signal lines resulting from spot defects that occur during fabrication [76]. Bridging faults are a common type of failure in CMOS circuits. One study shows that bridging faults account for thirty to fifty percent of all faults [76]. Testing for bridging faults was originally done using logic testing at the gate level. However, this method

has been proven to be ineffective [76]. An alternative approach that is gaining popularity is supply current, or I_{DDQ}, testing.

2.1.1 Fault Models for CMOS Circuits

For the past three decades, researchers have used the stuck-at fault model to represent faults in digital circuits. The stuck-at fault model has been used satisfactory to model faults in transistor-transistor logic (TTL) and n-channel metal-oxide-semiconductor (nMOS) circuits. As CMOS emerged as the dominant very large scale integration (VLSI) technology, researchers realized that the stuck-at fault model cannot accurately model physical faults in CMOS [30, 62]. For example, a bridging fault in a CMOS circuit my cause some nodes, including some output nodes, to be logically indeterminate. Unlike other technologies, CMOS does not provide a logical value dominance. The need for more accurate fault models to represent physical faults in CMOS became evident. The following switch-level fault types are used to represent physical faults in CMOS [76].

1. Node stuck-at-0: A node is said to be stuck-at-0 when its logic value is permanently stuck at logic "0."

2. Node stuck-at-1: A node is said to be stuck-at-1 when its logic value is permanently stuck at logic "1."

3. Transistor stuck-off: A transistor stuck-off model represents a transistor with a permanently nonconducting channel.

4. Transistor stuck-on: This fault type models a permanently conducting transistor, i.e., its source and drain are explicitly shorted together.

5. Break fault: A break or open fault is a break in the connection between two nodes such that two nodes that are supposed to be connected become unconnected.

6. Bridging fault: A bridging or a short fault connects two different nodes that are not supposed to be connected.

This list can actually be reduced to two types of faults: bridging faults and break faults. A node stuck-at-0 can be modeled as a bridge between the node and *GND*, while a node stuck-at-1 can be

modeled as a bridging fault between the node and V_{DD}. A transistor stuck-on can be modeled as a bridging fault between the transistor's source and the drain. A transistor stuck-off is a break fault in the channel of the transistor or a bridging fault between the gate and the drain.

2.1.2 Modeling Bridging Faults

Most early research dealt with bridging faults at the gate level. The work modeled the bridging fault as a wired-AND or wired-OR. There are several problems with this model. First, a bridging fault is not restricted to the gate level; bridging faults occur at switch-level as well. Another problem with this model is that wired-AND and wired-OR do not represent the behavior of bridging faults in CMOS. Although wired-AND and wired-OR have been used successfully to model bridging faults in other technologies, e.g., TTL and nMOS, CMOS has no dominant voltage so two nodes shorted by a bridging fault may not end up having the voltage of the dominant voltage level. In most cases the resulting voltage will be an intermediate voltage based on the structure of the voltage divider created by the bridging fault. This situation may create indeterminate voltage values.

In this work, CMOS circuits will be modeled at the gate level and the switch level, depending on the class of the bridging fault encountered. A bridging fault will be modeled as a short circuit between switch-level or gate-level nodes.

2.1.3 Classification of Bridging Faults

When testing for bridging faults is done at the gate level, researchers use a wired-AND or wired-OR fault model. Through this simple model, bridging faults are classified into two types: feedback and non-feedback bridging faults. Since bridging faults also occur at the switch level, the gate-level model is inadequate. The following discusses some previously published switch-level classifications for bridging faults.

Malaiya and Jayasumana [54] report a detailed study of bridging faults and suggest a switch-level classification. They divide bridging faults into three classes:

1. bridging faults within a logic element, where a logic element is a simple or complex gate,
2. bridging faults between gate-level nodes without feedback, and
3. bridging faults between gate-level nodes with feedback.

Although they restricted their study to the above three types, they acknowledge the existence of bridging faults between switch-level nodes located in different logic elements.

Sousa, *et al.* [78] extend Malaiya and Jayasumana's work. They include bridging faults between switch-level nodes in different logic elements and consider primary input and output nodes as a separate node class. In later work, Saraiva, *et al.* [75] enhance this classification by considering whether or not a bridging fault produces feedback.

Midkiff and Bollinger [58] suggest that bridging faults should be classified according to three factors.

1. Extent of the fault: whether the fault affects a single gate or two gates, i.e., whether it is an intra-gate or inter-gate fault.

2. Type of affected nodes: the node types considered were V_{DD}, GND, logic gate input, logic gate output, p-type internal gate and n-type internal gate.

3. Presence of feedback.

In later work, Midkiff and Bollinger [59] present a modified and simplified bridging fault classification for I_{DDQ} testing. Figure 2.1 describes this classification. The new classification is oriented to I_{DDQ} testing. It does not include feedback since I_{DDQ} testing is capable of detecting feedback bridging faults without treating them as a special case [10]. They also report the relative percentage of occurrence for each class based on experimental results. This work uses this classification.

2.1.4 Extraction of Bridging Faults

If all possible bridging faults between any two nodes are considered, then the number of bridging faults will be very high [17]. In practice, only the adjacent or overlapping wires in the circuit layout need to be considered. This idea led to the need for a bridging fault extractor. The main job of a bridging fault extractor is to generate a list of potential bridging faults in the layout. Also, a bridging fault extractor can provide information about the likelihood of occurrence of each fault. This section describes several bridging fault extraction methods and the available implementations.

Inductive Fault Analysis (IFA) is the most prominent physical faults extraction method for CMOS circuits [76]. IFA consists of three major steps.

- **1. Gate Level**
  - 1.1. Wire Stuck-At
  - 1.2. Wire to Wire
    - 1.2.1. Same Gate
      - 1.2.1.1. Input to Input
      - 1.2.1.2. Input to Output
    - 1.2.2. Different Gates
- **2. Switch Level**
  - 2.1. Internal Node Stuck-At
  - 2.2. Transistor Stuck-On
  - 2.3. Wire to Internal Node
    - 2.3.1. Same Gate
      - 2.3.1.1. Input to Node
      - 2.3.1.2. Output to Node
    - 2.3.2. Different Gates
  - 2.4. Internal Node to Internal Node
    - 2.4.1. Same Gate
    - 2.4.2. Different Gates

Figure 2.1: Bridging fault classification [59].

1. Defect generation and analysis: A list of point defects is generated using statistical methods, then the electrically significant defects are kept while the insignificant ones are discarded.

2. Defect to fault translation: Each of the significant defects is analyzed and significant defects that contribute to faulty circuit behavior are transformed to circuit-level faults.

3. Fault classification and ranking: Faults are classified into circuit fault types and the relative likelihood of occurrence of each fault is estimated to produce a ranked fault list.

This method, like all extractors mentioned here, is not limited to bridging faults. IFA supports line stuck-at, transistor stuck-on, transistor stuck-off, floating line faults, bridging faults and some other faults.

Ferguson and Shen [27] developed the FXT program to extract realistic faults in CMOS circuits. FXT is an implementation of the inductive fault analysis method.

Acknowledging that FXT was a mere research tool, Jee and Ferguson [39] developed CARAFE. Similar to FXT, CARAFE is an implementation of the inductive fault analysis method. CARAFE is still in its "alpha" version, yet many researchers are using it successfully to extract bridging

faults. Unfortunately, the bridging faults extracted by CARAFE are not classified into any bridging fault classification. Also, CARAFE generates some untestable bridging faults that involve isolated, unused circuit nodes. Moreover, many of the bridging faults generated by CARAFE are equivalent from a circuit-level point of view. Bollinger developed CARP (CARAFE Post-processor) to solve these problems [9]. CARP removes untestable faults, combines equivalent faults and classifies the result into the classification proposed by Midkiff and Bollinger [59] presented in the previous section.

Long computation time is a major disadvantage of the inductive fault analysis method. Jacomet and Guggenbuhl [36] developed a fault extractor that uses fast arithmetic techniques to extract the potential physical defects from the circuit. Instead of using time consuming simulations, this method uses a set of simple formulas to extract physical defects. These defects subsequently are transformed into circuit faults. The likelihood of occurrence is calculated for each fault based on the statistical data of physical defects. Fault collapsing is performed by adding the probabilities of individual faults. This method supports bridging faults, break faults and stuck-at faults. Bridging faults are classified into two types: logical and current increase. Logical bridging faults can be detected by applying test vectors at the circuit inputs and observing the circuit outputs. Current increase bridging faults produce indeterminate logic values in the circuit and must be tested using I_{DDQ} testing.

Another physical fault extractor that avoids time consuming simulation was developed by Teixeira, *et al.* [78, 82]. This method defines a set of physical failure modes, *PhFM*, caused by defects. Open metal-1/diffusion contacts and shorts between adjacent metal-2 lines are two examples of the defined failure modes. For every activated failure mode, the extractor scans the circuit layout generating a list of all potential faults caused by this failure mode. Physical defects are translated to break, bridging, transistor stuck-off, or transistor stuck-on fault types. The extractor then assigns to each fault a weight, w_j, representing the likelihood of occurrence of the fault. This weight is calculated based on the relative likelihood of occurrence of the failure mode causing the fault and the geometrical characteristics of the fault. A post-processor performs fault collapsing and updates the weights by adding the weights of the different faults that are collapsed into a single fault.

Large VLSI circuits impose a major challenge for fault extraction tools due to the computational resources required to handle large circuits. Two of the newly developed extractor tools, CREST [61] and EYE [4], take advantage of the available hierarchy in the circuit layout to speed-

up computation and minimize memory requirement. Both tools successfully extracts large circuits while requiring considerably less execution time compared with other non-hierarchical extractors. Although CREST and EYE are mainly developed for yield prediction, minor modification to these tools or the use of a post-processor similar to CARP [9] should provide a fault list that include fault likelihood information.

2.1.5 I_{DDQ} Test Generation for Bridging Faults

I_{DDQ} testing can be used in static CMOS circuits since the steady-state supply current is very small. I_{DDQ} testing detects bridging faults in CMOS circuits by setting the two potentially shorted nodes to opposite values. If the bridge exists, the supply current is large which signals the existence of the bridging fault. Many researchers have proven the effectiveness of I_{DDQ} testing compared to traditional stuck-at fault testing [2, 55, 80]. I_{DDQ} testing is simple and effective, yet there are several practical problems. I_{DDQ} testing requires longer periods of time for each test vector because the circuit needs more time to stabilize when a high supply current is created by the fault. The QTAG standard has been proposed to address this problem [6]. Fault masking due to large leakage current in large circuits is another problem [53]. Also, I_{DDQ} testing does not imply a method to generate test patterns. The following is a presentation of some of the previous results on I_{DDQ} testing.

Malaiya and Su [55] performed the first systematic study of current testable faults in CMOS circuits. They present several motivations for the use of I_{DDQ} testing.

1. Power supply pins provide additional accessibility making fault propagation unnecessary.

2. I_{DDQ} testing can provide transistor-level testing as opposed to gate-level testing in the conventional stuck-at fault model.

3. The large current resulting from a bridging fault between two nodes with opposite values is generally a few orders of magnitude greater than normal leakage current. Therefore, excessive leakage current is easily identifiable.

4. Minimal stuck-at test sets have a close relationship with minimal I_{DDQ} test sets such that a minimal stuck-at test set can be used as a starting point to generate a minimal I_{DDQ} test set.

Acken [2] demonstrated that I_{DDQ} testing can effectively detect bridging faults. He also pointed out two problems with I_{DDQ} testing: test time and test vector generation. I_{DDQ} testing takes more time compared with logic testing because of the time needed for the supply current to stabilize. However, Acken did not suggest any methodology to generate the test vectors. Acken realized the importance of considering the circuit layout when he specified that testing should be done only for nodes that are adjacent in the layout.

Storey and Maly [80] compared stuck-at fault testing and I_{DDQ} testing when applied to CMOS bridging faults. Gate-level stuck-at fault test patterns were generated using MCNC's Computer-Aided Design (CAD) system, Open Architecture Silicon Implementation Software (OASIS) [65]. I_{DDQ} test patterns were generated using a test generation program developed by Nigh and Maly [63]. A layout-level extractor produced a list of potential bridging faults. Then, a switch-level simulation program determined the effectiveness of both sets of test patterns in detecting bridging faults. When the stuck-at test patterns were applied to the circuit, the detection of the fault was verified by monitoring the primary output nodes of the circuit, i.e., fault propagation to the output is needed. When the I_{DDQ} test pattern was used, the detection of the fault was verified by monitoring the supply current. This study concluded that for bridging faults, I_{DDQ} testing is superior to stuck-at fault testing in terms of accuracy and fault coverage. It should be noted here that this study was performed only on the smallest ISCAS 1985 benchmark circuits [13] since the I_{DDQ} test pattern generation program required excessive computer resources for large circuits.

Bollinger and Midkiff [10] present a test generation methodology that automatically generates I_{DDQ} tests for unrestricted bridging faults in CMOS circuits. A modular, hierarchical approach is used to accurately represent the structure of CMOS design styles. The bridging fault model used in this work can model the effect of arbitrary, unrestricted bridging faults. Unlike the work of Storey and Maly [80], this test pattern generation program successfully generated test patterns for all of the ISCAS 1985 benchmark circuits.

Functionally equivalent nodes in a circuit may cause some bridging faults to be undetectable by I_{DDQ} testing. Isren and Figueras [35] present an efficient method to identify classes of functionally equivalent nodes in combinational circuits. Isren and Figueras suggest two benefits for their work. First, knowledge about functionally equivalent nodes should be provided to the test generation tool to prevent it from wasting time in trying to generate tests for undetectable bridging faults. Second,

functionally equivalent nodes should not be placed near one another in the layout to minimize the probability of undetectable bridging faults. Isren and Figueras only consider gate-level bridging faults.

Chakravarty and Thadikaran [15] present three algorithms for selecting a small subset of test vectors to perform I_{DDQ} testing. The small subset is selected from a stuck-at test set generated by available stuck-at test generation programs or random test generators. Selection algorithms are based on a new fault simulation technique that simulates gate-level and switch-level bridging faults. Both combinational and sequential circuits are considered.

Reddy, *et al.* [71] propose a procedure to generate compact test sets to detect bridging faults using I_{DDQ} testing. First, random test vectors are generated as long as they are effective in detecting large numbers of bridging faults. Then, deterministic test vectors are generated to produce tests that detect the largest possible number of yet undetected bridging faults. This method generated smaller test sets with better fault coverages than Chakravarty and Thadikaran [15] described above.

I_{DDQ} testing is not limited to the detection of only bridging faults, but can also detect transistor stuck-open faults [74], break (open) faults [56], and delay faults [33]. I_{DDQ} testing is gaining popularity in the industry as a reliable testing methodology [45, 64]. When used as a supplement to traditional gate-level testing, I_{DDQ} testing provides an easy yet effective way to increase fault coverage [45, 64, 73].

2.2 Design for Testability

Design for testability (DFT) implies that design techniques are used to make the testing of a circuit more effective. Traditionally, DFT is done at the gate level with the goal of increasing the controllability and observability of the circuit. As researchers realize the importance of realistic physical faults, the need for a new level of DFT has become evident. This new level, physical design for testability (PDFT), deals with physical switch-level and layout-level faults. Physical design for testability is still an emerging area. The following sections first present traditional gate-level testability measures and traditional gate-level DFT methods. Then, previous work in the area of physical design for testability is presented.

2.2.1 Testability Measures

Testability analysis evaluates the relative degree of difficulty of testing circuit nodes for line stuck-at faults [28]. Testability analysis depends on two properties: controllability and observability. Controllability is a measure of the ease or difficulty of setting a node to a desired logic value, while observability is a measure of the ease or difficulty of propagating a node's value to a primary output. A node is said to be easily testable if it is highly controllable and observable. One of the goals of testability measures is to guide the designer in enhancing the testability of the circuit. However, many designers do not accept this technique because no accurate relationship between circuit characteristics and testability has been demonstrated [28, 69]. Moreover, testability measures do a poor job in predicting detectability of individual faults [3]. Testability measures are also successfully used to guide test generation algorithms [16, 20]. Other researchers have used testability measures to estimate fault coverage as an alternative to fault simulation [12, 14, 37]. This section presents some of the widely used gate-level testability methods. Little work has been done to define testability measures for fault models other than the line stuck-at fault model. Work on non-stuck-at testability measures is also presented in this section.

Goldstein [32] presents one of the first successful testability programs, SCOAP (Sandia Controllability Observability Analysis Program). SCOAP characterizes circuit nodes as sequential or combinational. A combinational node is defined to be a primary input or a combinational standard cell output, while a sequential node is an output node of a sequential standard cell. SCOAP represents the testability of each node N by a vector having six elements:

1. $CC^0(N)$ combinational 0-controllability,
2. $CC^1(N)$ combinational 1-controllability,
3. $SC^0(N)$ sequential 0-controllability,
4. $SC^1(N)$ sequential 1-controllability,
5. $CO(N)$ combinational observability, and
6. $SO(N)$ sequential observability.

$CC^0(N)$ and $CC^1(N)$ are the minimum number of combinational node assignments required to set node N to "0" or "1," respectively. Analogously, $SC^0(N)$ and $SC^1(N)$ are the minimum number

of sequential node assignment required to set node N to "0" or "1," respectively. $CO(N)$ is the number of combinational standard cells between node N and a primary output plus the minimum number of combinational node assignments required to propagate the logical value on node N to a primary output of the circuit. Similarly, $SO(N)$ is the number of sequential standard cells between node N and a primary output plus the minimum number of sequential standard cells that must be controlled to propagate the logical value on node N to a primary circuit output. Note that these measures are exact integers that depend on circuit topology.

Another testability measure program is CAMELOT (Computer Aided Measure for Logic Testability) reported by Bennetts, *et al.* [8]. CAMELOT assigns a single controllability value, $CY \in [0, 1]$, to every gate-level line in the circuit. The maximum value 1 indicates a node, such as a primary input, where forcing a "1" is as easy as forcing a "0." At the other extreme, a CY of 0 indicates a node that cannot be set to "1" or "0". In practice, the majority of the nodes in a circuit have CY node values between these two limits. Similarly, CAMELOT assigns a value OY to each node indicating the observability of that node. OY is defined to be a measure of the ease of observing a state of a node at the circuit's primary outputs. The controllability and observability values calculated by CAMELOT are heuristic figures of merit having no mathematical foundation. Clearly, with just one controllability value and one observability value for each node, CAMELOT requires less computation than SCOAP. CAMELOT provides a testability measure for each node, TY_{node},

$$TY_{node} = CY_{node} \times OY_{node} \tag{2.1}$$

and a testability measure for the whole circuit, $TY_{circuit}$,

$$TY_{circuit} = \frac{\sum TY_{node}}{\text{number of nodes}} \tag{2.2}$$

VICTOR, by Ratiu, *et al.* [70], is different from the other testability programs in its emphasis on detecting redundant stuck-at faults. Also, VICTOR is restricted to combinational circuits. VICTOR defines three controllability and three observability parameters for every node in the circuit: label, weight, and size. A label is a sufficient test for controlling or observing a node. The weight is the weighted sum of the constrained primary inputs and fan-out branches associated with a certain label. Finally, the size is the number of all possible tests for controlling or observing a node. In addition to calculating testability measures for every node and identifying redundant faults, VICTOR generates test vectors for some of the faults in the circuit. Unfortunately, it is

typical in VICTOR to identify many nodes as being potentially redundant even if no redundant faults are present [69].

While the previous methods use heuristics to derive the testability measures, other methods used probability theory to develop testability measures, based on the work of Parker and McCluskey [67].

Brglez [12, 14] introduced COP (Controllability/Observability Program) that uses probability theory to calculate testability measures. COP assigns three probability values to each line.

1. $C^1(l)$: the 1-controllability of a line l is defined as the probability that line l takes value "1."

2. $C^0(l)$: the 0-controllability is the probability that line l takes value "0."

3. $O(l)$: the observability of line l is the probability of observing the value of line l at a primary output.

COP is a simple direct application of probability theory for the calculation of testability.

STAFAN [37] uses probability theory in a way similar to COP. STAFAN, however, assigns two observability values to each node: 1-observability and 0-observability. These observabilities are defined similarly to COP's observability, with the added initial condition that line l is set to "1" or "0," respectively. STAFAN calculates the controllabilities based on an experiment that uses random vectors, while COP calculates them based on the Boolean equations of each node. This makes the testability measures calculated by COP dependent only on the circuit, while STAFAN produces testability measures that depend on both the circuit and the test vectors used in the experiment. Hence, STAFAN produces different values for testability measures based on the test vectors used in the experiment, while COP values are always consistent.

The work presented above targets only gate-level stuck-at faults. Two published works, summarized below, target bridging faults and break faults.

Kapur, *et al.* [41] presented the first testability measures for bridging faults. They define controllability as the number of input vectors that create a difference in the node values at the fault site divided by the total number of input vectors. Observability is defined as the number of input vectors that allow the difference in node values at the fault site to propagate to at least one observable circuit output divided by the total number of input vectors. Both of these definitions are restricted to combinational circuits. This work uses the wired-AND and wired-OR models to model bridging faults at the gate-level. Calculating controllability and observability for bridging

faults by this method requires first calculating the Boolean function realized at every node in the circuit. One advantage of this method is that it provides accurate results even when the circuit has reconvergent fan-out. However, its complexity in terms of memory and computation time is high. The high computing resource requirements and the limited bridging fault model are two major disadvantages of this work.

Sousa, *et al.* [78] report a different approach to measure fault testability. They consider bridging faults and break faults, and suggest that the testability of a certain fault is related to the type of fault and its location. Their method performs fault classification of bridging and break faults. A level of fault hardness is assigned to each fault class. A fault is assumed to be a hard fault if it is difficult to detect, or undetectable, by logic testing. Fault hardness depends on fault type and topological characteristics. Using extensive simulations and experience, Sousa, *et al.* suggest a level of fault hardness for fault classes. However, they do not provide a hardness level for many of the bridging fault classes. In later work, Saraiva, *et al.* [75] enhance the bridging fault classification presented in [78] by adding classifications based on whether or not a bridging fault introduces feedback. They also suggest fault hardness levels for all bridging fault classes. This method is significantly different than previous ones in the sense that it does not assign a testability measure to each fault, but rather to each fault class. The only work needed to decide on the testability of a certain fault is to determine the fault class to which the fault belongs. Clearly, the amount of computation in this method is minimal once a fault hardness level for each fault class is developed.

Dalpasso, *et al.* [21] define a testability measure for bridging faults called the random detection probability. Random detection probability is defined as the expected random test length required to detect a fault. The less number of random test vectors needed to detect a bridging fault the more testable is the fault. The calculation of the random detection probability requires time consuming accurate switch level fault simulations. However, for standard cell design, the use of a database of standard cells testability characteristics makes the calculation of the bridging faults random detection probability more efficient.

2.2.2 Gate-Level Design for Testability

Gate-level design for testability (DFT) techniques are usually divided into two classes: *ad hoc* techniques and structured scan techniques. This section presents a summary of prior for both

types of techniques.

Ad hoc techniques are techniques which are applied to a particular circuit, but cannot be used as a general method to solve the testability problem [84]. In general, the area penalty for *ad hoc* techniques is lower than that for structured scan techniques. However, test generation and fault simulation are more complicated processes when *ad hoc* techniques are used. Some of the known *ad hoc* techniques are as follows [1, 69, 84].

1. Add initialization circuitry when the circuit has an unknown initial state.
2. If the circuit contains an internal clock, add circuitry to disconnect the internal clock and connect the tester's clock instead.
3. Add circuitry to permit the tester to break feedback loops.
4. Avoid "wired" logic.
5. Add test points to access fan-out and fan-in points.
6. Partition large counters, shift registers and large combinational circuits.
7. Avoid redundant logic.

Scan techniques are structured techniques that give a general solution for the DFT problem. Scan techniques enhance controllability and observability by using a scan register [1]. Scan registers have both shift and parallel load capabilities. Scan techniques are based on the concept that if the value in all latches in a given circuit can be set to any specified value and if they can be observed with a straightforward operation, then the circuit can be thought of as purely combination [84]. Tests can then be applied to the primary inputs and shift register outputs. Scan techniques permit access to the internal nodes of a circuit without requiring a separate external connection for each node accessed. Few (up to four) inputs are added to the circuit to facilitate access to many internal nodes. Test generation for circuits that use scan techniques is a straightforward process with a low computational cost. However, a penalty is paid by adding complex flip-flops or latches to the circuit that increases the complexity of the circuit, the area, and the propagation delay [69]. Some of the most popular scan techniques are described below.

The first scan technique, called scan path, was presented by Williams [85]. Scan path uses a special D flip-flop called a raceless D flip-flop. Circuits that incorporate scan path have two modes of operation: normal functional mode and test mode [69]. During test mode, flip-flops are connected to form a shift register and a test pattern is shifted into the flip-flops. By applying the rest of the test pattern to the primary inputs and then running the circuit at normal mode for one clock period, the combinational circuitry takes the flip-flop contents and the primary inputs as input. Part of the combinational output is taken from the primary outputs while the other part, stored in the flip-flops, is shifted out of the flip-flops using the test mode. Originally, scan path used a multiplexed data flip-flop which has the ability to choose between two inputs [85]. A better implementation of scan path using a two-port flip-flop was developed at NEC [29].

Eichelberger and Williams [23, 24] present another scan design method called level-sensitive scan design (LSSD). LSSD replaces each circuit flip-flop by a two-port latch and a second single-port latch. Clearly, this adds to the complexity of the circuit due to the complex latch structure used [84]. The name LSSD stems from the fact that its operation does not depend on signal rise and fall time or on internal delays, hence, races and hazards do not affect the steady-state response of a circuit that incorporates LSSD [1]. LSSD's ability to solve some of the problems in manufacturing and maintaining VLSI systems was proven to the extent that some manufacturers, IBM for example, have adapted it as their standard design technique [69].

Instead of using the functional shift register Stewart's [79] scan technique adds another shift register whose sole purpose is to shift the test data in and out. This technique is called scan-set. Scan-set has higher hardware overhead than scan path or LSSD. A major advantage of scan-set is that the scan function can occur during normal system operation such that a snapshot of system status can be obtained without any degradation in system performance [84]. Another advantage of scan-set is its ability to scan circuit nodes other than latch outputs which provides better observability [69].

Ando [5] introduced random access scan which was developed at Futjitsu. This method allows each latch to be uniquely selected so that it can be either controlled or observed. The similarity of this addressing mechanism to that of random memory access gives the method its name [84]. Random access scan has the ability to scan out the latches during normal operation. It requires a complex latch structure, but incurs a small pin overhead.

2.2.3 Physical Design for Testability

Physical design for testability means to design testable circuits in terms of physically realistic faults, in particular, bridging and break faults. This section presents prior work done in this area. Prior research can be divided into two categories. The first category deals with physical design for testability for logic elements. The second category deals with interconnections, placement and routing. Given the relative newness of this area, prior work is limited.

Levitt and Abraham [48] did several experiments that apply layout-level design for testability to improve the stuck-open coverage of stuck-at test sets. They designed a modified subset of the CMOS3 library [34] using local transformations. The purpose of a local transformation is to modify a cell such that the new design is less prone to break faults. Having designed the modified cell library, they synthesized several benchmark circuits using the original CMOS3 library cells and their modified cells. The conclusion of their experiments is that for an increase of 15 percent in area, the size of the fault list can be reduced by 35 percent, while fault coverage for break faults using stuck-at test vectors would increase by 35 percent.

Ferguson [25] establishes several principles of physical design for testability. He suggests that testability can be enhanced in three ways:

1. design the circuit to have fewer faults,
2. make difficult-to-test faults easier to detect, and
3. make difficult-to-detect faults unlikely.

Ferguson also suggests that the problem be divided into two parts: PDFT for logic blocks and PDFT for interconnections. Logic blocks, like standard cells and gate arrays, are usually reusable in many designs. By modifying the layout of these blocks to make them more testable, a circuit using these logic blocks should be more testable. Ferguson defined the difficulty of testing a certain fault f in a logic block by the following equation.

$$Diff_f = \frac{\text{number of tests that detect } f}{\text{number of possible tests}} \tag{2.3}$$

Clearly, using this equation to calculate the difficulty of testing all possible physical faults in a circuit is not usually practical. As for creating interconnections in the layout, which includes placement, routing and logic selection, Ferguson suggested the following ideas.

1. Placement: place the gates of the same depth together to reduce the possibility of feedback bridging faults.

2. Routing: route in a way that will prevent potential feedback bridging faults.

3. Logic selection: for some faults using a cell with more drive strength can prevent oscillation, making some faults easier to detect.

Although Ferguson [25] provides one of the best definitions of the objectives and potential strategies for the PDFT process, the work provides only an outline of how research should proceed. No experiments to support or validate the work were conducted. Also, the suggested gate-level test generation program, Nemesis [26], was not designed to handle bridging faults. While several modifications were made to Nemesis to make it capable of handling bridging faults, it cannot detect all types of bridging faults [26]. Most of the suggestions made by Ferguson to make layouts more testable are based on the way Nemesis detects bridging faults. For example, feedback bridging faults are avoided as much as possible since Nemesis has difficulty detecting them. This suggests that making a bridging fault, or any other fault, easier to detect depends on the complete testing methodology, i.e., making a certain layout more testable for Nemesis is different than making it more testable using I_{DDQ} testing.

Teixeira and colleagues have also investigated PDFT [75, 78, 83]. They use a circuit and fault extractor called Lift. By scanning the layout, Lift generates a fault list of layout level faults in addition to the netlist. A weighting factor, p_j, is assigned to each fault which represents the probability of occurrence of that fault. Lift supports four types of faults: LOP (line open), BRI (bridging), TSOP (transistor stuck-open) and TSON (transistor stuck-on). Each of the four fault types is subdivided into fault classes based on the fault topology. Each fault class has a level of hardness, i.e., a fault is assumed to be a hard fault if it is difficult to detect by logic testing using a stuck-at test set. The level of hardness assigned to each class is based on extensive simulations with different designs and layout styles. Teixeira, *et al.* present a set of guidelines for the avoidance of hard break faults in [83]. Saraiva, *et al.* [75] designed a cell library based on these guidelines and designed two layouts for the c432 ISCAS 1985 benchmark circuit [13] using this cell library. Those two layouts had a better fault coverage compared with two other layouts for the same circuit, one provided by a European manufacturer and the other designed using a commercial CAD package.

The two layouts that use the new cell library not only have better fault coverage, but also use less area than the other two. This raises questions about the adequacy of the comparison made. No guidelines to avoid bridging faults were presented in [83].

Saraiva, *et al.* [75] state the need for a new design philosophy that influences design methodology and CAD tool development. They also suggest that PDFT should involve the following features.

1. Cell versioning which implies that two versions of the same cell are available, should be used. The normal version, designed for performance, should be used most often while the more testable version, designed with more stringent design rules, should be used at critical areas within the circuit layout.

2. The use of custom non-CMOS logic blocks should be restricted since they are usually less testable than full CMOS.

3. New routers that take testability into consideration should be designed.

4. A "decompactor" should be designed that can identify unused chip regions and selectively "decompact" interconnection lines.

Lee and Breuer [47] present the following design and test rules for CMOS circuits to facilitate I_{DDQ} testing for bridging faults.

1. The gate and drain (or source) nodes of a transistor cannot be in the same transistor group.

2. During steady-state operation, there must be no conducting path from V_{DD} to GND.

3. During steady-state operation each output of transistor group must be connected to V_{DD} or GND through a path of conducting transistors.

4. There must be no control loops among transistor groups.

5. The bulk (or well) of an n-type (p-type) transistor must be connected to GND (V_{DD}).

6. During testing, each primary input must be controlled by a strong power source whose current is also monitored.

A transistor group is specific circuit partition. Lee and Breuer point out that this set of rules is minimal in the sense that if any rule is removed, circuits exist for which I_{DDQ} cannot give correct results. They also prove the sufficiency of the rules, i.e., if all of the proposed rules are satisfied then each irredundant single or multiple fault can be detected using a single test vector. An irredundant multiple bridging fault is one which affects the logic function of the circuit. Various strategies are suggested to make I_{DDQ} testing still possible for CMOS circuits that do not satisfy all of the above rules. Fully complementary primary and complex gates satisfy the above rules.

Pitaksanonkul, *et al.* [68] developed a channel router program, DTR, that minimizes the total area and simultaneously reduces critical areas between channel wires, thus reducing the probability of bridging faults. The critical area is the area in which the center of a defect must fall to cause the fault to occur. For inter-layer bridging faults the critical area falls between adjacent wires while for intra-layer bridging faults the critical area falls around and including the overlap area between different layers wires [39]. DTR uses a new algorithm based on the channel router algorithm proposed by Yoshimura and Kuh [86]. DTR produces at least 11 percent less critical area when compared to Yoshimura and Kuh's router, while using a comparable total area. Unfortunately, DTR minimizes critical area between horizontal wires only and does not allow "dog-legging," which is breaking a horizontal wire into several segments and allowing each segment to reside on a different horizontal track.

Balachandran, *et al.* [7] propose a channel router post-processor to reduce the critical area in the channel without increasing the total area. This post-processor reduces the critical areas between horizontal wires and the critical areas between vertical wires by dog-legging the wires to fit into unused gaps within the channel.

Recently Kuo [44] developed YOR (Yield Optimizing Router), a channel router which includes a post-processor. Similar to DTR, YOR uses a modified version of the method of Yoshimura and Kuh [86] to minimize the critical area in addition to minimizing the total channel area. Experimental results have shown that the contribution of the routing algorithm to critical area reduction is insignificant. Instead the major contribution to the critical area reduction is provided by YOR's post-processor. The post-processor simultaneously minimizes critical area and vias using four techniques:

1. Net burying: Net burying moves a wire segment on the top routing layer to the lower routing

layer. This is possible only if there is no wire underneath the moved wire.

2. Net floating: Net floating is the opposite of net burying in that it moves a lower layer wire to the top layer.

3. Net bumping: Net bumping bends part of a net segment on a routing track to another track to reduce the critical area caused by it. Net bumping is a general case of net dog-legging.

4. Shifting vias: vias can be shifted or eliminated using net burying and net floating.

McGowen and Ferguson [57] developed a channel router that reduces the probability of undetectable bridging faults between horizontal wires in the channel. This router reduced the critical area of undetectable bridging faults between horizontal wires by 40 percent without increasing the number of routing tracks. McGowen and Ferguson limited their work to non-feedback bridging faults because the testing tool they used, Nemesis [26], cannot handle feedback bridging faults accurately.

WENTR [43] is another channel router that enhances testability. WENTR uses simulated annealing [72] algorithm for channel routing and incorporates the enhancement of testability into the objective function of the simulated annealing process. Testability is measured using a function called the testability penalty function. The testability penalty function is defined as the sum of probability of occurrence times the undetectability for all potential bridging faults. By adding the minimization of the testability penalty function to the objectives of the simulated annealing process, WENTR enhances testability while minimizing area at the same time. WENTR produces channels that have 30 percent reduction in the testability penalty function at an expense of 10 percent larger area.

2.3 Summary

With the emergence of CMOS as the dominant technology for VLSI, more attention is being paid to physical faults since traditional gate-level stuck-at faults are ineffective in representing physically realistic faults in CMOS technology. As a result, the idea of physical design for testability has emerged as a solution to provide more reliable and testable CMOS circuits. Unfortunately, little work on physical design for testability has been done to date. In particular, bridging faults have

received little attention even though they are common in CMOS circuits. Several good ideas have been discussed in the literature, but no experiments on bridging faults have been conducted. More work is needed in the area of PDFT, including theory, design methodology, and experimental work. Testability measures for physical faults also need attention.

Chapter 3

Statement of the Problem

This chapter states the need for a new design methodology that supports physical design for testability. The previous chapter clarified that PDFT is still an emerging concept and that little experimental work has been done to establish it. Previous work defined the main goal of PDFT, to design a circuit that has fewer physical faults, or more specifically, to remove faults that are hard to detect.

This chapter is divided into three sections. The first section presents the specific goals of this work. The second section deals with establishing the concept of physical design for testability presenting what is done and what is still undefined. The third section explains the scope of this work.

3.1 Research Goals

Four specific goals are proposed.

1. Provide a clear definition of physical design for testability.

2. Propose a new design methodology that incorporates PDFT.

3. Realize a suite of CAD tools needed for the proposed design methodology. This includes the development of new tools for measuring the controllability of bridging faults and other tools to avoid or reduce the probability of occurrence of bridging faults. The new tools will be integrated with existing CAD tools.

4. Demonstrate the feasibility and benefits of using the proposed design methodology.

3.2 The Concept of Physical Design for Testability

Given the status of physical design for testability, it is clear that there is still much work needed, even in the basic foundations of this concept. This section discusses the goals of PDFT and suggests a new design process that supports PDFT. The new design process needs several new CAD tools. Those CAD tools are also discussed in this section.

3.2.1 Definition of Physical Design for Testability

Since no formal definition for PDFT is given in prior work, we provide the following definition.

Physical Design for Testability is a design strategy and a set of techniques to avoid or reduce the likelihood of realistic physical faults, and to ease the testability of imminent faults.

For faults that cannot be avoided, PDFT should reduce the probability of occurrence of these faults and also should ensure that these faults are easily testable. PDFT deals with physical faults which include bridging faults, break (open) faults, transistor stuck-on and transistor stuck-off. Compared to traditional gate-level stuck-at faults, physical faults more closely represent realistic faults appearing at the gate level and switch level.

Note that PDFT is concerned with faults not defects. Defects are more related to the manufacturing process and technology while PDFT is related to the design and testing stages. A single physical fault can be the result of the accumulative effect of several defects occurring at several different locations in the layout. When dealing with physically realistic faults, PDFT works at layout level, switch level and gate level. Although PDFT utilizes information about defect statistics, PDFT does not deal with altering the manufacturing process or technology.

As will be shown later, PDFT overlaps with gate-level design for testability (DFT) in testability measures: controllability and observability. Also, gate-level stuck-at faults, traditionally handled using DFT, are also modeled as physical realistic faults and handled by PDFT as well.

3.2.2 Goals of Physical Design for Testability

The goal of much of the prior work was to make the circuit designed more "testable." Unfortunately, "testable" is not always used in a clear and precise manner. The set of goals suggested by Ferguson in [25] is the best published definition of the goals of physical design for testability. This work uses the same list of goals suggested by Ferguson with more specific interpretation.

1. *Design the circuit to have fewer faults.* The design process should recognize physical faults, including which situation led to the occurrence of these faults and which design methods can eliminate some of these faults. The design process must have access to the process defect statistics that contribute to fault occurrence. If a circuit has fewer faults, the time for fault simulation, time for test generation, and number of test vectors will be reduced. A price is paid, however, in a more complicated design process. For standard cell design, the cells should be designed in such a way to have fewer faults. By using a fault extractor, a cell could be redesigned several times until satisfactory results are achieved. Typically, the result should be a balance between a reduction in physical faults and an increase in area. At the routing stage, the router should have a mechanism to reduce the critical area in the channel with no, or minimal, increase in channel area.

2. *Make difficult faults easier to detect.* Ferguson [25] suggests that adding control and observation points to nodes that are difficult to control and observe can make difficult faults easier to detect. This requires identification of which faults are easy to test and which ones are difficult to test, which in turn requires that fault types and the testing methodology be defined. Then, the testability of faults, which depends on the design methodology, can be evaluated. As will be shown later, the difficulty of detecting a bridging fault, using I_{DDQ} testing, is a function of the controllability of the gate-level nodes shorted by the fault. Designing the circuit to have highly controllable gate-level nodes should cause many bridging faults to be more testable. Hence, PDFT starts at the design of the gate-level implementation of the circuit. Any gate-level structure with low controllability or low observability should be avoided. Increasing the controllability and observability of gate-level nodes by redesigning the gate-level implementation is a better approach than adding controllability and observability points since this requires adding more pins the chip.

3. *Make difficult to detect faults unlikely.* There are two ways to make difficult to detect faults unlikely to occur: on a fault type basis and on a single fault basis. Using a fault type basis means that the average testability of different types and classes of physical faults should be evaluated. Then, the design strategy should avoid the occurrence of these difficult to detect types. This starts from designing the first layout. This method is similar to the work of Saraiva [75]. The other method is to evaluate the testability of each fault on individual basis and to design the hard to detect faults out of the layout. The design process should also consider the probability of occurrence of each fault. Obviously the worst faults would be the ones with high probability of occurrence and low testability.

3.2.3 Proposed Design Methodology

This work proposes the design process shown in Figure 3.1. Instead of designing the circuit and then generating tests, this process incorporates the evaluation of the realistic faults into the design process. This design methodology assumes standard cell design.

At the top of Figure 3.1, the design process starts with a gate-level netlist or a Boolean equation description of the circuit. Also, a standard cell library must be available. This work uses a multiple-version standard cell library. A multiple-version standard cell library includes two or more versions of each cell. Different versions of the layouts are designed with different goals. This work uses two versions for each cell, normal and defect-tolerant. Other cell versions could be added to optimize other parameters like delay. The normal versions are designed in a traditional manner to provide reliable cells with optimal area and lowest achievable delay. The defect-tolerant versions are designed to reduce the probability of occurrence of physical faults. The first circuit layout generated includes normal versions of the cell only. Starting from the second iteration, a mixture of normal and defect-tolerant versions are used. A minimization and mapping program produces a gate-level netlist that is mapped to the cell library. This program optimizes the gate-level netlist for area. A placement and routing program constructs the circuit layout. In the first iteration, the placement and routing program uses the normal cell version. Then, the circuit layout is evaluated using several tools. A fault extraction tool coupled to fault testability evaluation tool provides a list of potential physical faults together with their likeliness of occurrence and testability. A critical path delay extraction tool provides the delay. This information, plus the area, is provided to the

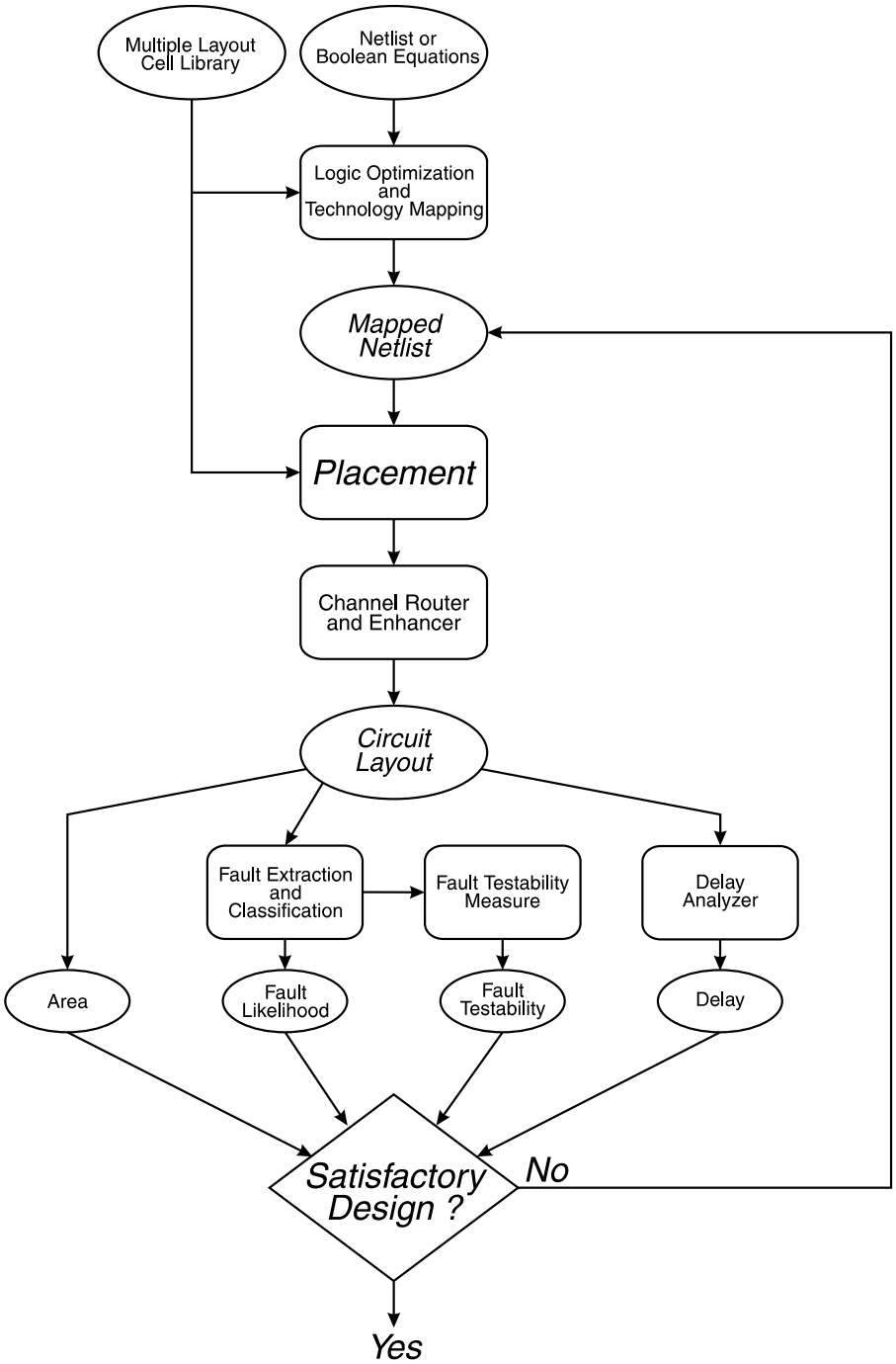

Figure 3.1: The proposed physical design for testability process.

designer who decides if the design is acceptable or not. If not, another design iteration is used to enhance the layout. Using feedback from the previous iteration, a new layout for the circuit is created.

The following information is provided to the designer for each circuit layout.

1. Area
2. Critical path delay
3. Physical fault list that includes the probability of occurrence for each fault
4. Testability of each physical fault

We define a parameter called the fault index. The fault index for a particular fault is the probability of occurrence of the fault divided by the testability of the fault.

$$\text{fault index} = \frac{\text{probability of occurrence}}{\text{testability}} \tag{3.1}$$

The probability of occurrence of a fault dependents on the type and the topology of the fault. This value is usually provided by a fault extractor program which extracts the list of physical faults from the layout. The testability of the fault is a measure of difficulty of testing the fault (see Chapter 5). Faults with a high probability of occurrence and low testability have a high index. These faults are targeted in the next design iteration.

In the next iteration the designer tries to enhance the implemented layout using the same gate-level netlist. The enhancement of the layout is done in two ways:

1. row enhancement using the defect-tolerant layout for some cells,
2. channel enhancement by reducing the critical area for some bridging faults with low testability.

CMOS circuits studied in this work assume standard cell design and consist of several rows of horizontally abutted standard cells. Routing channels, located above and below each row, connect each row to the above and below rows. This arrangement suggests naming the tool that handles logic bridging faults as the row enhancer and the tool that handles the channel bridging faults as the channel enhancer. Note that PDFT is divided into two parts: logic and interconnection [25]. The logic physical design for testability deals with faults occurring inside logic elements like standard

cells and logic arrays. The second category deals with faults occurring in the routing part of the circuit or due to a certain placement situation. Hence, the row enhancer deals with logic PDFT and the channel enhancer deals with interconnection PDFT.

Row Enhancement

Row enhancement decides which cells in the netlist should be replaced by the defect-tolerant version. A new circuit layout is constructed to avoid some of the physical faults. The testability of the physical faults is not enhanced since version change does not change the gate-level nor the switch-level implementation. Some of the physical faults are removed, however. The row enhancement program (row enhancer) scans the list of physical faults selecting the ones with high fault indices that can be removed using row enhancement. The designer has the ability to set the fault index threshold used by the row enhancer. Only faults with a fault index higher than this parameter are considered by the row enhancement program. Also, the designer can instruct the row enhancer to target a fixed number of faults starting with the highest index fault.

Channel Enhancement

The channel enhancement program (channel enhancer) is actually a channel router with a post-processor. The post-processor modifies the channel to limit the probability of physical faults. The channel router function works in two modes: normal and testability consideration mode. The testability consideration mode tries to reduce the probability of some of the most undesirable faults in the channel, i.e., faults with high fault indices.

Section 8.4, later in this work, includes a discussion of the relation between the channel enhancer and the row enhancer. A particular point of interest is what fault types are affected by the row enhancer and what fault types are affected by the channel enhancer. There are also fault types affected by both tools and fault types not affected by either.

3.2.4 CAD Tools

The design process discussed above depends on the availability of certain CAD tools. This section describes the CAD tools needed to implement the PDFT methodology.

1. Gate-level synthesis tool: Many gate-level synthesis tools are available with optimization and technology mapping features. Since the testability of bridging faults is a function of the surrounding gate-level nodes (see Chapter 5), adding the ability to enhance the testability of gate-level nodes should enhance the testability of physical faults.

2. Physical fault extractor: Fault extractors are perhaps the only mature CAD tools in the PDFT process. Several fault extractors have been developed as discussed in Chapter 2. CARAFE [39] and LOCAL45 [52] are two commonly used physical fault extractors. More work is needed to integrate these extractors into the PDFT process. All of the extractors available today were designed mainly for test generation or yield prediction, rather than to redesign layouts to avoid some faults.

3. Placement program: Work is needed to investigate whether placement programs could incorporate PDFT.

4. Routing program: Chapter 2 presented several routing programs that were designed to avoid bridging faults.

5. Decompactor: Saraiva [75] suggested using a decompactor, but no implementation is available.

6. Multiple-version cell library: Strictly speaking, a cell library is not a CAD tool. It is mentioned here since it is an important element in the proposed PDFT process.

3.3 Scope of this Work

Applying all of PDFT concepts suggested in the previous section is infeasible, hence, the scope of this work will be limited to the following.

1. Out of the overall CMOS circuit design process, this work considers the logic design stage to the layout implementation stage.

2. Only bridging faults are considered in this work. They are the most common type of physical fault in CMOS circuits [76] and no previous PDFT experiments for bridging faults have been conducted. Other physical faults, such as break faults will not be considered in this work.

3. The work considers only I_{DDQ} testing since I_{DDQ} testing is suitable for detecting bridging faults. Unlike logic testing, I_{DDQ} testing does not require fault propagation.

The tools examined will not include logic optimization, placement tools, and decompactors. The addition of extra controllability or observability nodes will also not be considered.

Chapter 4

Methodology

This chapter presents a practical implementation of the PDFT design process presented in Chapter 3. Some, but not all, of the CAD tools used in the PDFT design process already exist, and no thorough experiments have been conducted to date. In particular, no one has conducted experiments that incorporate bridging faults.

4.1 Design Flow Chart

The proposed design flow is shown in Figure 4.1. This design flow is an implementation of the process shown in Figure 3.1 in the previous chapter. The design flow in Figure 4.1 has the following characteristics.

1. It uses hierarchy at the gate level, switch level and layout level.
2. It uses off-the-shelf CAD tools as much as possible, but in an integrated manner.
3. It is based on standard cell design.

The following presents details of the design process.

4.1.1 A Multiple-Layout Cell library

For demonstration and experimental purposes, the multiple-layout standard cell library will have at least five to seven standard cells with at least two layouts for each cell. Normal layouts are taken

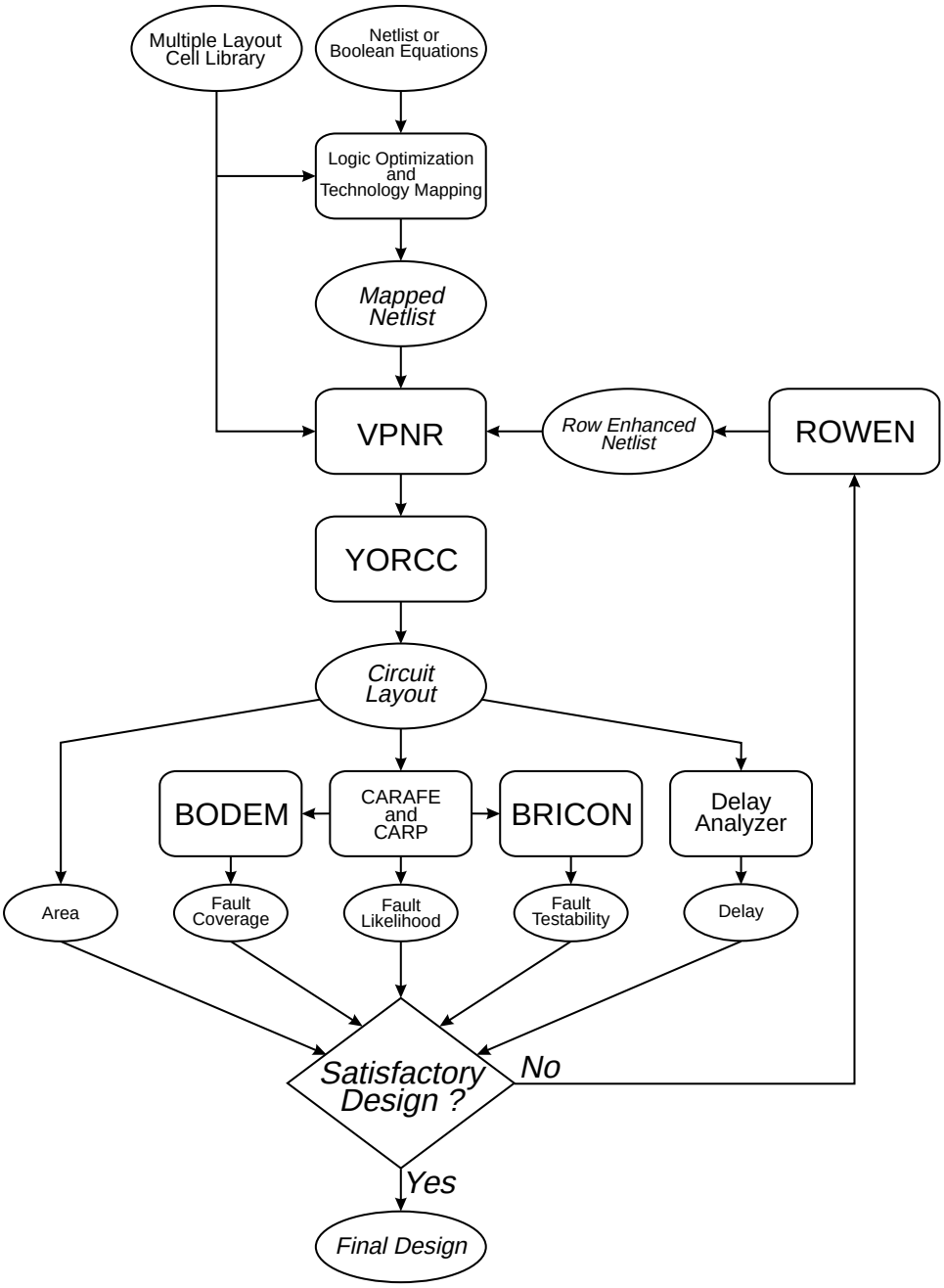

Figure 4.1: Implementation of the PDFT process.

from the MCNC cell library. Defect-tolerant layouts will be developed using MAGIC [66] and CARAFE [39]. The defect-tolerant layouts will be designed to reduce critical area while keeping the increase in area below a specified limit. A maximum increase of 20 percent in the area of each cell is proposed for demonstration purposes.

4.1.2 Logic Synthesis

In this implementation the logic synthesis stage consists only of the technology mapping program Decaf [49] from the OASIS package [65]. Decaf reads a netlist in the ISCAS 1985 [13] format and maps the network into a netlist consisting of standard cells. The optimization performed by Decaf can be targeted towards obtaining a circuit with minimum transistor count or minimum delay. This work will use the transistor count minimization mode to minimize the area. Note that if the different layouts of each cell in the multiple-layout cell library use the same switch-level design, then the netlist generated for the first layout applies to all the other layouts.

4.1.3 Placement and Routing

The placement and routing program VPNR, part of the OASIS package, will perform the placement of standard cells and global routing. Channel routing, however, will be done by a new router developed specifically for this work. The new router, YORCC, is based on YOR [44], but will extend YOR to consider testability information while doing the routing process. In addition of being a channel router, YORCC is also a channel enhancer that minimizes the critical area of bridging faults in the channel.

4.1.4 Layout Evaluation

The following metrics will be evaluated for each circuit layout.

1. The total area of the layout in square lambdas. A square lambda is the smallest unit of area considered in designing circuit layouts and its specific value depends on the implementation technology.

2. Critical path delay which is the longest delay between the input and output signals.

3. The likelihood of occurrence for each bridging fault represented by the critical area of the fault.

4. Testability of each bridging fault as evaluated by BRICON, a tool developed specifically for this work (see Chapter 5).

5. Weighted fault coverage. In a traditional design environment, fault coverage evaluation would not be part of the iterative layout design process. In this work fault coverage will be evaluated for every layout generated to monitor any improvement.

4.1.5 Row Enhancement

The row enhancer program, ROWEN, is the feedback element in the design process. ROWEN reads the fault list that includes the probability of occurrence of and the testability of each fault. ROWEN targets the faults with high fault indices and replaces their related gates with the defect tolerant version to reduce the critical area and hence reduce the fault index.

4.2 Experimental Procedure

Experiments with benchmark circuits will be used to both demonstrate and evaluate the PDFT design process. The experiments will show that the new design process allows the designer to find different solutions with different attributes. The goal of the experiments is to demonstrate the capabilities of the design process rather than to derive an efficient design. Three of the large ISCAS 1985 circuits [13] will be used; c7552, c5315 and c3540. Circuit c6822 will not be used due to problems in evaluating controllability values for the bridging faults in the circuit (see Section 5.5).

The experiment starts with a circuit netlist. The logic minimization and mapping technology program generates a netlist that is mapped to the standard cell library. The netlist generated minimizes the number of transistors. This netlist applies for all layout versions since different cell layouts share the same switch-level diagram. Placement and routing programs generate the first circuit layout without any information about possible bridging faults. The second iteration generates the second layout for the circuit using feedback information from the first iteration. The second iteration uses row enhancement to avoid some of the bridging faults. Channel enhancement is not based on feedback information and, thus, channel enhancement is applied to all layouts. In

this work the design process will target the 100 faults with the highest fault indices. These faults are taken from the list of bridging faults of the previous layout. Several layouts will be generated for the circuit. As we generate the layouts, we measure the variations in evaluation metrics and note any changes.

The reason for picking the 100 faults with highest fault indexes is demonstrated in Figure 4.2 and Figure 4.3. Figure 4.2 and Figure 4.3 are the result of applying CARAFE and BRICON (see Chapter 5) on circuit c432 of the ISCAS 1985 benchmark circuits. CARAFE generates a list of potential bridging faults together with the critical area of each fault. BRICON reads the output of CARAFE and generates a controllability parameter for each bridging fault. Since the probability of occurrence of each fault is proportional to the critical area, we substitute the critical area in place of probability of occurrence when calculating the fault index using Equation (4). Also, the controllability value generated by BRICON is used instead of testability since, for I_{DDQ} testing, testability depends only on controllability.

Figure 4.2 shows critical area and fault index for the 500 faults with the highest critical area in circuit c432. Figure 4.3 reorders the faults relative to the fault index. The faults with the highest fault indices contribute most of the area under the fault index curve. For example, the top 100 faults contribute 40 percent of the area under the curve. Since this circuit has 2,844 faults generated by CARAFE, around 3 percent of the faults contribute 40 percent of the area under the fault index curve. Moreover, picking a small number of faults will make the row enhancement computationally simpler.

4.3 Expected Results

After the experiment is finished, there will be several layouts for each circuit. The first layout includes only normal cell layouts and no row enhancement. The second layout includes some defect-tolerant cell layouts. The third layout includes more defect-tolerant cells, and so on. Channel enhancement is applied to all layouts. It is expected that the area and the delay will increase with each new layout, while the area under the fault index curve will decrease and the fault coverage will increase. In other words, the testability of the layouts should increase, but at the expense of increased area. It is important to remember that the goal of this experiment is to evaluate the

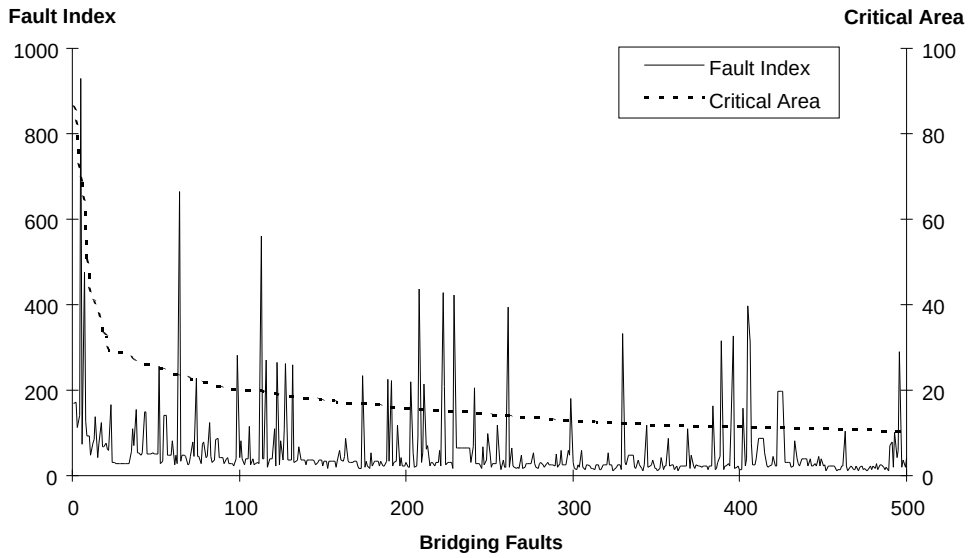

Figure 4.2: Critical area and fault indexes for the top 500 faults ordered by critical area.

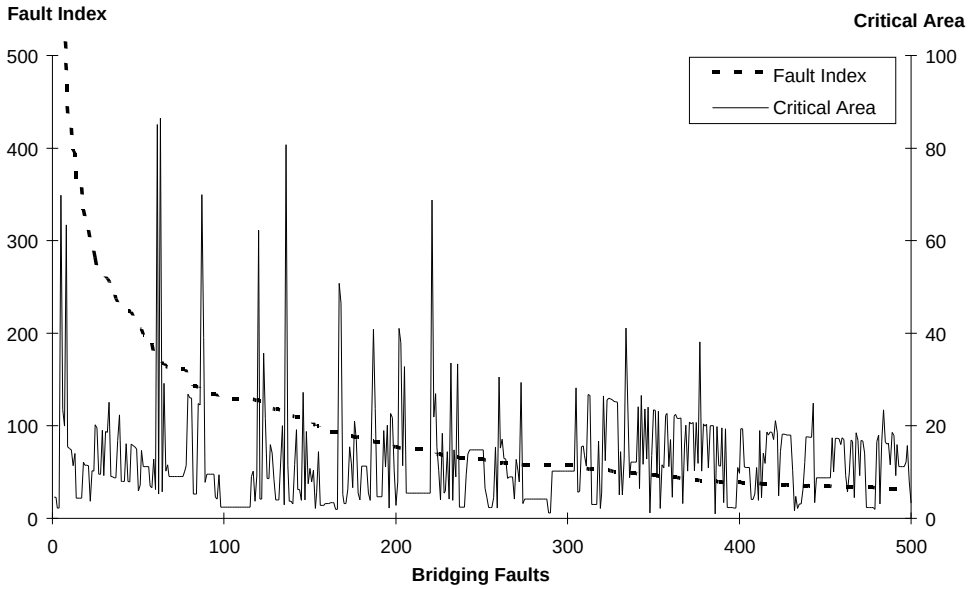

Figure 4.3: Critical area and fault indexes for the top 500 faults ordered by fault indices.

PDFT process. It is not the purpose of this experiment to design an optimum circuit.

Every layout can be characterized by four metrics: area, delay, testability, and weighted fault coverage. The design space for a given logic-level implementation consists of different layouts at different points in the four-dimensional space defined by these four metrics. It is expected that the iterations will show a progression through the design space, with each iteration showing improvement with respect to increased testability and increased fault coverage. However, it is also expected that the improvement will be at the expense of degradation with respect to increased delay and/or increased area. The proposed design process will be considered to be successful if it produces, for each circuit, layouts in different regions of the design space, thus allowing the designer to make trade-offs to meet his or her needs.

Chapter 5

Bridging Controllability—A Testability Measure for Bridging Faults

Defining a testability measure for bridging faults is an essential part of PDFT. The purpose of this chapter is to derive a testability measure for bridging faults. Testability is usually defined in terms of controllability and observability; however, I_{DDQ} testing grants observability. Therefore, the testability measure defined here is based only on controllability. This chapter defines the derived testability measure for bridging faults which is an extension of the traditional controllability measures defined for stuck-at faults. Three of the traditional methods are extended to bridging faults and then compared to select the one most suitable for bridging faults.

Section 5.1 extends the traditional controllability measures defined for stuck-at to bridging faults. Section 5.2 explains how bridging controllability is calculated for different classes of bridging faults. A procedure for calculating bridging controllability is outlined in Section 5.3. Section 5.4 presents BRICON a program that calculates bridging controllability. Section 5.5 conducts an experiment to investigate the relation between bridging controllability and detectability and also to find out which of the traditional controllability methods, when extended, is more suitable for bridging faults. A comparison to previous work in testability of bridging faults is presented in Section 5.6.

5.1 Extension of Controllability Measures to Bridging Faults

I_{DDQ} testing drives the two bridged nodes to opposite logic values and observes the supply current. The assumption made is that the easier it is to set these two nodes to opposite values, the easier it is to detect the fault. Traditional controllability methods are extended to handle two nodes by adding an imaginary EXOR gate B with the two bridged nodes as inputs and the node f as output, i.e., $f = x \oplus y$, as shown in Figure 5.1. When $f = 1$ the fault is detected, so the more difficult it is to drive f to 1, the more difficult it is to detect the fault. If two wires, x and y, are shorted by a bridging fault then the bridging fault controllability of this fault is

$$BC(x, y) = \mathit{Controllability}^1(x \oplus y) = \mathit{Controllability}^1(f) \tag{5.1}$$

where $\mathit{Controllability}^1(f)$ is the one-controllability of f. $\mathit{Controllability}^1(f)$ can be found by using one of the gate-level controllability methods, as discussed below.

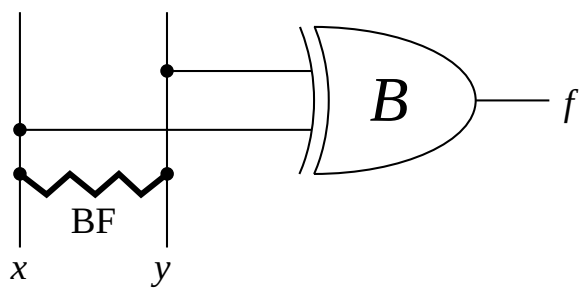

Figure 5.1: The imaginary EXOR gate B connected to the two wires shorted by a bridging fault.

5.1.1 CAMELOT

Let $BC_C(x, y)$ be the bridging fault controllability based on CAMELOT. The expression for $BC_C(x, y)$ is

$$BC_C(x, y) = CY(f) = CY(x \oplus y) = \frac{CY(x) + CY(y)}{2} \tag{5.2}$$

where $CY(x)$ is the controllability of node x according to CAMELOT. This equation is not applicable when one of the nodes, say x, is GND or V_{DD}. That is, no controllability value can be assigned to GND or V_{DD} that satisfies the general equation of CAMELOT.

5.1.2 SCOAP

Let $BC_S(x, y)$ be the bridging fault controllability based on SCOAP. The equation for $BC_S(x, y)$ is

$$BC_S(x, y) = CC^1(x \oplus y) - 1 = CC^1(f) - 1 \tag{5.3}$$

where $CC^1(x)$ and $CC^0(x)$ are the 1-controllability and 0-controllability of node x according to SCOAP. SCOAP defines the controllability of a node as the number of nodes assigned including the node itself. Since node f is imaginary, subtracting one provides the needed compensation. The 1-controllability for f is

$$CC^1(f) = \min\{CC^0(x) + CC^1(y),\ CC^1(x) + CC^0(y)\} + 1 \tag{5.4}$$

Hence, the expression for $BC_S(x, y)$ is

$$BC_S(x, y) = \min\{CC^0(x) + CC^1(y),\ CC^1(x) + CC^0(y)\} \tag{5.5}$$

For stuck-at faults, i.e., when either x or y is at GND or V_{DD}, the following values are defined.

$$CC^0(V_{DD}) = CC^1(GND) = 1 \tag{5.6}$$

$$CC^1(V_{DD}) = CC^0(GND) = 0 \tag{5.7}$$

For general bridging faults, SCOAP needs to handle internal switch-level nodes. The controllability figure is still the minimum number of combinational *gate-level* node assignments required to justify a "1" or "0" on that node, whether this node is a gate-level or switch-level node.

5.1.3 COP

Let $BC_P(x, y)$ be the bridging fault controllability based on COP. The expression for $BC_P(x, y)$ is

$$BC_P(x, y) = C^1(x \oplus y) = C^1(f) = C^0(x) \cdot C^1(y) + C^1(x) \cdot C^0(y) \tag{5.8}$$

$C^1(x)$ and $C^0(x)$ are the 1-controllability and 0-controllability of node x according to COP. COP satisfies the boundary conditions (node x set to V_{DD} or GND) with no extra definitions as shown in the following equations.

$$C^0(V_{DD}) = C^1(GND) = 0$$

$$C^1(V_{DD}) = C^0(GND) = 1$$

5.2 Bridging Controllability and Bridging Fault Classes

Equations 5.2, 5.5 and 5.8 cannot be used for general switch-level bridging faults since the controllability methods handle gate-level wires only. Also, x may be a function of y or vice versa, making the calculated bridging fault controllability inaccurate. These problems are solved by evaluating f in terms of independent gate-level nodes and then using the basic definition of controllability to calculate the 1-controllability of f. The following section indicates how this is done for different classes of gate-level and switch-level bridging faults as defined in [59].

5.2.1 Gate-Level Faults

Bridging faults at the gate-level exist between wires connecting gates or a wire and power or ground.

5.2.1.1 Wire Stuck-At

A wire stuck-at bridging fault shorts a wire x to either power or ground. For I_{DDQ} testing, we need to set the wire x to the opposite logic value. The general equations for bridging controllability can be used for SCOAP and COP. For a short to ground, SCOAP gives

$$BC_S(x, GND) = \min\{CC^0(x) + CC^1(GND),\ CC^1(x) + CC^0(GND)\} = CC^1(x)$$

Note that we use the controllabilities of ground defined in Section 5.1.2. Similarly, for x shorted to V_{DD}

$$BC_S(x, V_{DD}) = CC^0(x)$$

For a short to ground, COP gives

$$BC_P(x, GND) = C^0(x) \cdot C^1(GND) + C^1(x) \cdot C^0(GND) = C^1(x)$$

Note that $C^1(GND) = 0$ while $C^0(GND) = 1$ from the basic definition of COP. Similarly, for x shorted to V_{DD}

$$BC_P(x, V_{DD}) = C^0(x)$$

For CAMELOT we solve for f since no controllabilities can be defined for GND and V_{DD}. If x is stuck-at-0 then

$$f = x \oplus 0 = x$$

and if x is stuck-at-1 then

$$f = x \oplus 1 = \overline{x}$$

Hence, the equations for BC using CAMELOT are

$$BC_C(x, GND) = CY(x)$$

$$BC_C(x, V_{DD}) = CY(x)$$

CAMELOT gives the same value for bridging controllability for x shorted to ground or power, which is a weakness of CAMELOT.

5.2.1.2 Wire to Wire—Same Gate—Input to Input

For a fault shorting two wires x and y that are inputs of a gate G, the general equations can be used assuming that x and y are independent.

5.2.1.3 Wire to Wire—Same Gate—Input to Output

For a bridge between a gate input and output, the gate output is a function of the input to the gate, as shown in Figure 5.2. This is a reconvergent fan-out situation, created in the faulty circuit, that affects the calculation of the bridging controllability. Since we know the relation between the two wires, we can solve for f in terms of the gate input nodes and use the basic equations for controllability to find $BC(x, y)$.

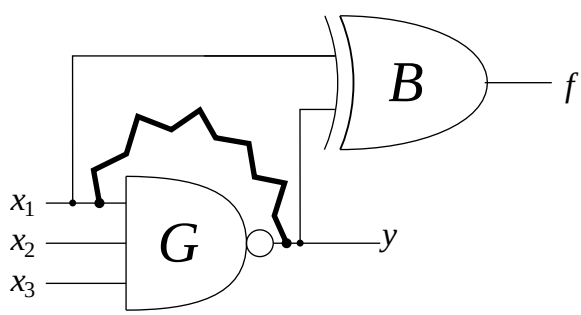

Figure 5.2: Input node shorted to output node.

For example, assume an n-input NAND gate with inputs $x_1, x_2, \ldots, x_n$ and output y and that x_1 is bridged to y. Solving for f in terms of $x_1, x_2, \ldots, x_n$ yields

$$f = x_1 \oplus y = x_1 \oplus \overline{(x_1 x_2 \ldots x_n)} = \overline{x}_1 + x_2 \ldots x_n$$

For simplicity, assume $n = 3$, i.e., $f = \overline{x}_1 + x_2 x_3$. CAMELOT gives

$$CY(f) = CTF \cdot \frac{CY(x_1) + CY(x_2) + CY(x_3)}{3}$$

where

$$CTF = 1 - \frac{|N(0) - N(1)|}{|N(0) + N(1)|} = 1 - \frac{|3-5|}{|3+5|} = \frac{3}{4}$$

and

$$BC_C(x_1, y) = CY(f) = \frac{CY(x_1) + CY(x_2) + CY(x_3)}{4}$$

For SCOAP,

$$CC^1(f) = \min\{CC^0(x_1),\ CC^1(x_2) + CC^1(x_3)\} + 1$$

$$BC_S(x_1, y) = CC^1(f) - 1 = \min\{CC^0(x_1),\ CC^1(x_2) + CC^1(x_3)\}$$

For COP, f is expanded as a function of minterms.

$$f = x_1 \oplus y = x_1 \oplus \overline{x_1 x_2 x_3} = \overline{x}_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 x_2 x_3$$

From the expression for f we can find the bridging controllability.

$$\begin{aligned} BC_P(x_1, y) \quad = \quad & C^0(x_1)C^0(x_2)C^0(x_3) + C^0(x_1)C^0(x_2)C^1(x_3) + C^0(x_1)C^1(x_2)C^0(x_3) \\ & + C^0(x_1)C^1(x_2)C^1(x_3) + C^1(x_1)C^1(x_2)C^1(x_3) \end{aligned}$$

5.2.1.4 Wire to Wire—Different Gates

For faults that bridge two wires x and y that are inputs or outputs of two different gates, G_1 and G_2, the controllabilities are the same as for gate-level bridged inputs faults described above.

5.2.2 Switch-Level Faults

Stuck-at controllability methods were designed to work at the gate-level, not at the switch-level. Bridging faults, however, occur at both the gate-level and the switch-level. The major difference

between gate-level wires and internal nodes at the switch-level is that many internal nodes can be tri-stated, i.e., not driven to either "1" or "0". This situation makes the basic bridging controllability equations unsuitable for most internal nodes. Gate-level methods are extended to the switch-level by treating the internal node under consideration as an output of another logic gate that exists inside the original one, as shown in Figure 5.3. Hence, we introduce an imaginary gate I residing inside the actual gate G. I has the same inputs as G, with the internal node n as its output. Gate I is used to express n as a function of the inputs of G, thus permitting the calculation of the controllability of the internal node.

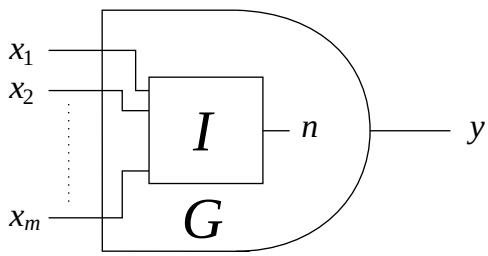

Figure 5.3: The imaginary logic block I whose output node represents the internal node n.

For internal node n, there are three Boolean functions, n_0, n_1 and n_z, derived from the gate structure. n_0 is true when $n = 0$, n_1 is true when $n = 1$, and n_z is true when n is tri-stated. Function f is defined as shown in Table 5.1. A bridging fault between nodes a and b is detected if function f is true. f is defined to be "0" if one or both of the bridged nodes is tri-stated since the fault cannot be detected. Table 5.1 defines function $f = a \oplus b$.

Table 5.1: Function $f = a \oplus b$

		b		
		0	1	z
a	0	0	1	0
	1	1	0	0
	z	0	0	0

5.2.2.1 Internal Node stuck-At

Controllability for an internal node stuck-at fault is similar to the gate-level external wire stuck-at fault. The expression for function f, is

$$f = n \oplus 0 = n = n_1$$

for n stuck-at-0 and

$$f = n \oplus 1 = \overline{n} = n_0$$

for n stuck-at-1 BC is then the 1-controllability of f. For example, consider the three-input NAND gate shown in Figure 5.4. This gate has input nodes a, b and c, output node y, and internal nodes n and m. Assume that n is shorted to ground, i.e., $f = n_1$. The three Boolean functions of n are

$$n_0 = bc = abc + \overline{a}bc$$

$$n_1 = a\overline{b} + a\overline{c} = a\overline{b}c + a\overline{b}\overline{c} + ab\overline{c}$$

$$n_z = \overline{a}\overline{b} + \overline{ac}$$

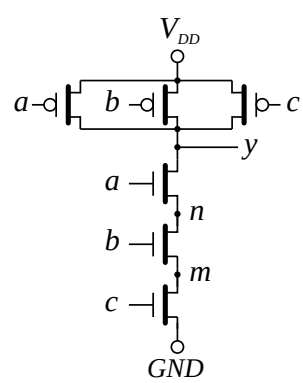

Figure 5.4: Three-input NAND gate.

Using CAMELOT, $N(0) = 2$ (the number of minterms in n_0) and $N(1) = 3$ (the number of minterms in n_1), so

$$CTF = 1 - \frac{2-3}{2+3} = \frac{4}{5}$$

$$CY(f) = CTF \cdot \frac{CY(a) + CY(b) + CY(c)}{3} = \frac{4}{15}[CY(a) + CY(b) + CY(c)]$$

$$BC(n, GND) = CY(f) = \frac{4}{15}[CY(a) + CY(b) + CY(c)]$$

For SCOAP,

$$CC^1(f) = \min\{CC^1(a) + CC^0(b),\ CC^1(a) + CC^0(c)\} + 1$$

$$BC_S(n, GND) = CC^1(f) - 1 = \min\{CC^1(a) + CC^0(b),\ CC^1(a) + CC^0(c)\}$$

For COP, the 1-controllability of f is found using the minterm form of n_1,

$$C(f) = C^1(a)C^0(b)C^0(c) + C^1(a)C^0(b)C^1(c) + C^1(a)C^1(b)C^0(c)$$

$$BC_P(n, GND) = C^1(f) = C^1(a)C^0(b)C^0(c) + C^1(a)C^0(b)C^1(c) + C^1(a)C^1(b)C^0(c)$$

In a similar manner we can calculate the bridging controllability for n shorted to V_{DD}.

5.2.2.2 Transistor Stuck-ON

To detect a stuck-on transistor, the drain node is set to "1" and the source node is set to "0", and the gate is set such that the transistor should be off. In static CMOS, the drain node is "1" and the source node is "0", only if the gate node turns the transistor off. For pass transistors, the drain node d and the source node s can be set to opposite values only if the gate node turns the transistor off. Hence, there is no need to consider the gate. For a pass transistor

$$f = d \oplus s = d_1 \cdot s_0 + d_0 \cdot s_1 \tag{5.9}$$

For a transistor in a gate the equation simplifies to

$$F = d_1 \cdot s_0 \tag{5.10}$$

The pass transistor equation is the general equation that includes the other case. Once f is found as a function of the gate-level input nodes, controllability is determined as for previous faults.

5.2.2.3 Wire to Internal node—Same Gate—Input to Node

Given the equations for the internal node n, the controllability for an input wire to internal node bridge is similar to that for the gate-level input to output bridge above. If input node x is shorted to internal node n, the expression for f is

$$f = n \oplus x = n_1 \cdot \overline{x} + n_0 \cdot x \tag{5.11}$$

5.2.2.4 Wire to Internal node—Same Gate—Output to Node

Controllability for an output wire to internal node is similar to the previous case, with internal node n shorted to the node y instead of an input node. The expression for f is

$$f = n \oplus y = n_1 \cdot \overline{y} + n_0 \cdot y \tag{5.12}$$

5.2.2.5 Wire to internal Node—Different Gates

If an internal node n is shorted to an external wire x that belongs to a different gate, the expression for f is

$$f = n \oplus x \tag{5.13}$$

In this case, different gates are involved, so it is simpler to calculate the 1-controllability and the 0-controllability of the internal node and then apply the general equation for bridging controllability, rather than solving for f.

5.2.2.6 Internal Node to Internal Node—Same Gate

If an internal node n is shorted to an internal node m, and both n and m belong to the same gate, the expression for f is

$$F = n \oplus m = n_0 \oplus m_1 + n_1 \oplus m_0 \tag{5.14}$$

We solve for f in terms of the input nodes of G and then find the 1-controllability for f which is BC.

5.2.2.7 Internal Node to Internal Node—Different Gates

For internal nodes n and m, each in different gates G_1 and G_2, respectively,

$$F = n \oplus m \tag{5.15}$$

As for the wire to internal node fault above, it is simpler to find the 0-controllability and 1-controllability for each node and then treat them as gate-level nodes by applying the general equation for bridging controllability.

5.3 Procedure for Calculating Bridging Controllability

The following procedure calculates $BC(x, y)$, the bridging controllability between two wires, x and y, using either SCOAP or COP. This procedure is modified to handle CAMELOT, as discussed below.

1. Find the gate-level 1-controllability and 0-controllability for all gate-level nodes.
2. If x and y are gate-level nodes with no relation between them, find $BC(x, y)$ using the general bridging fault equations 5.5 or 5.8.
3. For faults where different gates are involved, find the 1- and 0-controllability of any internal node involved, and then apply the general bridging fault equation.
4. For the remaining faults that include two nodes in the same gate, solve for f in terms of the input nodes of the gate. Then, from the 1-controllability of f, find BC.

If CAMELOT is used, only a single controllability value is calculated for each node. Also, since no controllabilities can be defined for *GND* or V_{DD} that satisfy CAMELOT's general equation, Equation 5.2, wire and internal node stuck-at faults should be handled as in step 4.

5.4 BRICON: A Bridging Controllability Program

This section presents BRICON, a program that calculates bridging controllability. BRICON calculates BC based on SCOAP and COP. CAMELOT was not implemented due to its lack of a strong mathematical foundation. In particular, no controllability values can be defined for *GND* or V_{DD} that satisfy CAMELOT's general equation. BRICON is implemented as shown in Figure 5.5. We start with a description of the CMOS circuit layout in MAGIC [66] format. CARAFE [39] provides the bridging fault list, while MAGIC extracts the circuit. CARP [59] reads these two files and generates a gate-level netlist for the circuit and a modified fault list that compensates for some problems in the original fault list as mentioned in Chapter 2. BRICON reads the modified fault list and the circuit netlist file and generates another fault list that has all the information in the modified fault list plus two bridging controllability figures, one based on SCOAP and the other

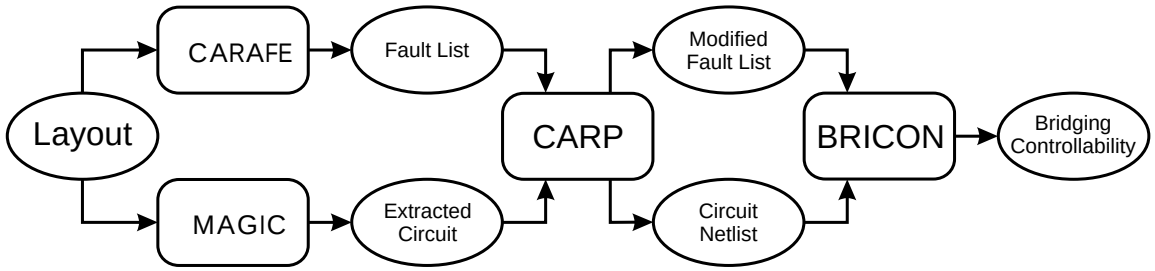

Figure 5.5: Flow diagram showing inputs and output of BRICON.

based on COP. BRICON has a library of all CMOS standard cells used in the layouts giving the Boolean logic equations for the output node and internal nodes of each cell.

BRICON was tested using the MCNC layouts[1] of the ISCAS 1985 benchmark circuits [13]. BRICON successfully calculated the bridging controllability values for all types of bridging faults and successfully recognized bridging faults that are always detectable with I_{DDQ} testing, e.g., a bridging fault between the input and output of an inverter.

5.5 BRICON Experimental Results

This section describes an experiment conducted to assess the benefit of bridging controllability. The experiment determines the correlation between bridging controllability and an estimate of detectability, where detectability d for a certain bridging fault is defined as

$$d = \frac{\text{number of input test vectors that detect the fault}}{\text{all possible input test vectors}} \tag{5.16}$$

Finding an exact value for detectability is infeasible due to the large number of test vectors involved. An estimate of d can be found by using a random sample of test vectors. The number of random vectors, n, is determined by

$$n = \frac{(z_c)^2 \cdot p \cdot q}{E^2} \tag{5.17}$$

where z_c is the confidence coefficient, E is the error, p is proportion of success and $q = p - 1$ is the proportion of failure [40]. For 99 percent confidence ($z_c = 2.58$) and an error of 0.03, Equation (14) produces n = 1,850 test vectors. Parameters q and p are both set to 0.25 which is the worst case

[1]Layouts generated at MCNC and provided by K. Kozminski.

since the product of p and q is maximum when both are set to 0.25. The experiment is conducted as shown in Figure 5.6. For every circuit in the set of ISCAS 1985 benchmark circuits, 200 bridging faults were chosen randomly from the modified fault list by PICKF. The 200 bridging faults and 1,850 random vectors were fed to SIM, a bridging fault simulator written specifically for the experiment. SIM simulates every bridging fault for every test vector to estimate the detectability of the bridging faults. BRICON calculates the bridging controllability for the 200 faults. CORR combines the two lists and finds the correlation. The results of the experiment are summarized in Table 5.2.

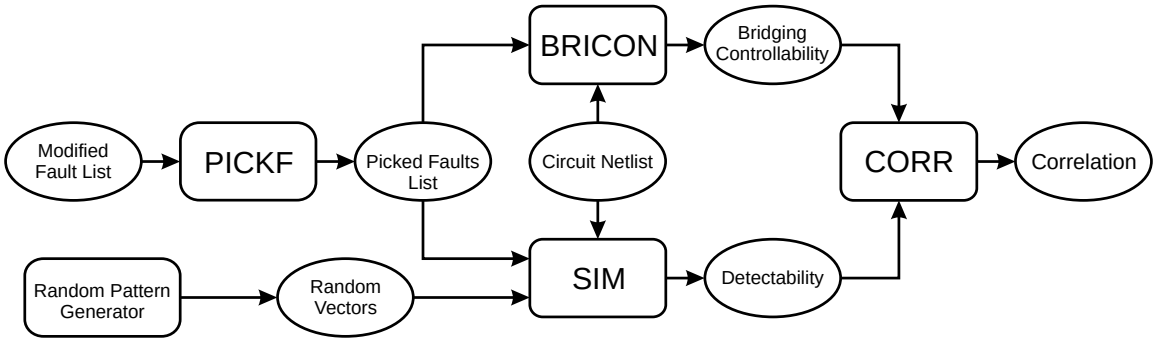

Figure 5.6: Flow diagram showing the experiment.

As indicated in Table 5.2, the correlation between bridging controllability according to COP and detectability is high, more than 0.9 in all cases except circuit c6822 which has a correlation of 0.78. Bridging controllability based on SCOAP exhibits lower correlation. This is due to the fact that the definition of COP and the definition of detectability are similar and are both based on probability theory, while SCOAP is defined based on the number of nodes rather than probabilities.

Circuit c6822 is an interesting case to consider. This circuit has the lowest correlation with COP. It is also a very deep circuit having 119 levels compared to 12 to 34 levels for all other circuits. While conducting the experiment on c6822, numerical problems occurred in calculating the COP controllabilities for the last few levels of the circuit. It is suspected that as the controllabilities propagated through the circuit, truncation errors accumulated and led to a loss of accuracy in calculating the controllabilities. This numerical error plus the large number of levels in c6822 led to the decrease in the correlation. It should be noted that SCOAP results are integers, hence, the

Table 5.2: BRICON Experimental Results for Benchmark Circuits

Circuit	**c432**	**c499**	**c880**	**c1355**	**c1908**	**c2670**	**c3540**	**c5315**	**c6288**	**c7552**
Nodes	174	221	272	407	341	694	772	1323	1880	1681
Gates	138	180	212	366	308	537	722	1,145	1848	1474
Depth	24	12	17	23	20	25	34	23	119	21
Correlation (COP)	0.91	0.99	0.98	0.94	0.98	0.91	0.94	0.96	0.78	0.92
Correlation (SCOAP)	-0.46	-0.61	-0.41	-0.40	-0.4	-0.16	-0.14	-0.37	-0.15	-0.34

depth of the circuit does not harm the accuracy of SCOAP results. The programs were implemented on a DECstation 5000 which has a word length of 32 bits. Using a 64-bit machine in the future should solve the numerical problem associated with COP. The experiment do demonstrate that bridging controllability according to COP is generally a good estimate of detectability.

5.6 Comparison with Previous Work

In this section, a previous bridging fault controllability measure presented by Kapur, *et al.* [41] is discussed and the differences between their work and this work are highlighted. Also, the average controllability for each fault classification is computed and compared to previous work.

Kaput, *et al.* presented the first testability measures for bridging faults [41]. They define controllability as the number of the input vectors that create a difference in the node values at the fault site divided by the total number of input vectors. Observability is defined in [41] as the number of input vectors that allow the difference in node values at the fault site to propagate to at least one observable circuit output divided by the total number of input vectors. Both of these definitions are restricted to combinational circuits. Wired-AND and Wired-OR models are used to model bridging faults. Calculating controllability and obsevability for bridging faults by the method in [41] requires first calculating the Boolean function realized at every node in the circuit. One advantage of this method is that it does not suffer from the reconvergent fan-out problem, however, a high price in terms of memory and computation time is paid.

This work is different than Kapur's work in several aspects. This work does not provide a new definition for controllability, rather it extends the stuck-at controllability measures to accommodate bridging faults. Also, unlike Kapur's work, this work is not restricted to gate-level bridging faults. All three controllability methods discussed in this work produce inaccurate results when renconvergent fan-out is encountered. This remains a problem when these methods are extended to handle bridging faults. Finally, the complexity of BRICON, in terms of memory and CPU time, is less than the method in [41].

Saraiva, *et al.* [75] do not present a method to calculate the detectability of bridging faults, rather they present a detectability level for each bridging fault class. This detectability level is based on experience and extensive simulations rather than a specific methodology. To compare

this work with Saraiva's, the mean and standard deviation for each of the bridging fault classes were calculated for a commutative list of bridging faults generated by BRICON for all the ISCAS 1985 benchmark circuits. The results are summarized in Table 5.3. It is interesting to see that no contradictory results were obtained; in fact the results agree for four classifications even though Saraiva based his results on logic testing instead of I_{DDQ}.

5.7 Summary

This chapter defines a testability measure for bridging faults. Testability is usually defined in terms of controllability and observability. Since I_{DDQ} testing grants observability, testability measure defined here is based only on controllability. Three of the traditional controllability measures defined for stuck-at faults are extended to bridging faults. Comparing the three controllability methods discussed, CAMELOT has the weakest mathematical foundation. Being only a figure of merit, CAMELOT is not suitable for calculating switch-level bridging controllability. One case that illustrates the weaknesses of CAMELOT is that it will give the same bridging controllability value when a node is shorted to GND or V_{DD}. It should be emphasized that this conclusion applies to using CAMELOT for bridging faults and not stuck-at faults. COP is preferred over SCOAP since it has a solid mathematical foundation and is easier to implement. Boundary conditions are satisfied for COP without extra definitions, while SCOAP needs extra definitions as given in Section 5.1.2. The experiments conducted demonstrate that bridging controllability provides a good estimate of detectability, yet requires relatively low execution time.

BRICON a program that calculates bridging controllability based on SCOAP and COP is presented. BRICON's output gives the probability of occurrence of bridging faults and the controllability of each fault. The probability of occurrence is generated by CARAFE and preserved in BRICON's fault list. These two pieces of information are expected to help the circuit designer decide how to reduce the effects of bridging faults. Bridging controllability may also have an impact on I_{DDQ} testing tools if used to guide testing tools similar to the way the original controllability figures are used to guide gate-level ATPG tools.

In the following chapters The probability of occurrence and controllability, based on COP, of each fault are used to calculate the fault index. Through the fault index the PDFT design process

Table 5.3: Detectability Results of Bridging Fault Classification

Level	**Bridging Fault Class**	**Mean**	**Standard Deviation**	**Detectability (BRICON)**	**Detectability (Saraiva)**
Gate Level	Wire Stuck-at-1	0.491	0.221	varies	easy
	Wire Stuck-at-0	0.507	0.221	varies	easy
	Wire to Wire—Same Gate—Input to Input	0.456	0.112	moderate	moderate
	Wire to Wire—Same Gate—Input to Output	0.710	0.172	easy	easy
	Wire to Wire—Different Gates	0.498	0.074	moderate	easy/moderate
Switch Level	Internal Node stuck-at-1	0.257	0.186	varies yet difficult	difficult
	Internal Node stuck-at-0	0.246	0.173	varies yet difficult	difficult
	Transistor Stuck-ON	0.177	0.111	difficult	N/A
	Wire to Internal node—Same Gate—Input to Node	0.596	0.193	varies	difficult
	Wire to Internal node—Same Gate—Output to Node	0.354	0.234	varies	difficult
	Wire to internal Node—Different Gates	0.474	0.114	moderate	easy
	Internal Node to Internal Node—Same Gate	0.612	0.389	varies	difficult
	Internal Node to Internal Node—Different Gates	0.424	0.186	varies	easy

identifies bridging faults with low testability and high probability of occurrence. Then through other tools in the following chapters the PDFT design process targets these faults to reduce their probability of occurrence or eliminate them if possible.

Chapter 6

The Channel Enhancer

Channel routing is a standard problem in VLSI layout design. Traditionally, the objective of channel routing is to realize a set of wire connections utilizing as little area as possible. As the size and complexity of integrated circuits increase, the occurrence of physical faults like bridging faults and break faults becomes more probable. This has led some researchers to develop channel routers that, in addition to minimizing the area, try to reduce the likelihood of bridging faults. Reduction of the likelihood of bridging faults can be incorporated into the routing algorithm [57, 68] or into a post-processor [7, 44].

This chapter presents YORCC, a new channel router that supports physical design for testability (PDFT). The router reduces the critical area for bridging faults by incorporating critical area reduction into the routing algorithm and using a critical area reduction post-processor. Also, the new router compares the faults in terms of fault indices, thus making use of the critical area and the bridging controllability.

Section 6.1 introduces the net merging channel routing algorithm proposed by Yoshimura and Kuh [86]. Section 6.2 extends Yoshimura and Kuh's algorithm to handle cyclic conflicts. Section 6.3 presents a new track assignment scheme that minimizes the fault indices of bridging faults in the channel. A post-processor which minimizes the critical area and the number of vias is discussed in Section 6.4. Section 6.5 presents the YORCC program and experimental results.

6.1 The Yoshimura and Kuh Net Merging Algorithm

This section presents the net merging channel routing algorithm developed by Yoshimura and Kuh [86].

The channel routing problem can be understood by considering a rectangular channel with two rows of terminals along its top and bottom sides with N nets to be routed through the channel. Each terminal is assigned a number, i, where $0 \leq i \leq N$. Terminals with the same number, i, where $i \neq 0$, must be connected by net i. Terminals with number 0 designate unconnected terminals. Two layers are available for routing. We assume horizontal tracks on one layer and vertical tracks on the other. Traditionally vertical tracks are referred to as columns and horizontal tracks are referred to as tracks. Figure 6.1 shows the netlist representation for a channel routing problem, while Figure 6.2 shows a solution to this example.

$$\begin{bmatrix} 0 & 1 & 4 & 5 & 1 & 6 & 7 & 0 & 4 & 9 & 10 & 10 \\ 2 & 3 & 5 & 3 & 5 & 2 & 6 & 8 & 9 & 8 & 7 & 9 \end{bmatrix}$$

Figure 6.1: Netlist representation for a routing problem [86].

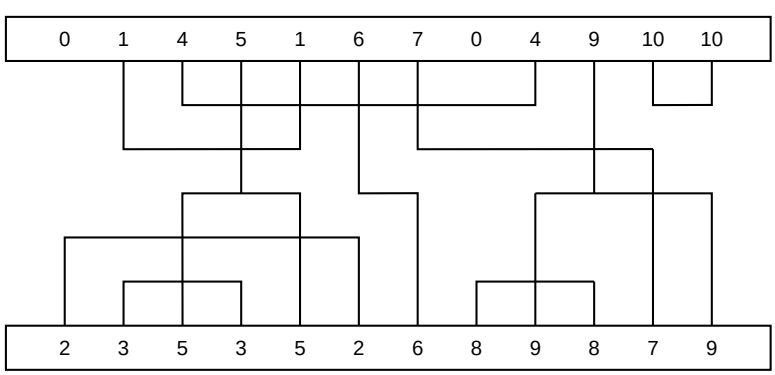

Figure 6.2: A solution for the example in Figure 6.1 [86].

The following two definitions are needed for the net merging algorithm.

Vertical Constraint Graph

The relation between nets can be represented by a directed graph G_v, where each node corresponds to a net and a directed edge from net a to net b means that net a must be placed above net b, i.e., the horizontal track containing a must be above the horizontal track containing b. Figure 6.3 shows the vertical constraint graph G_v for the netlist shown in Figure 6.1.

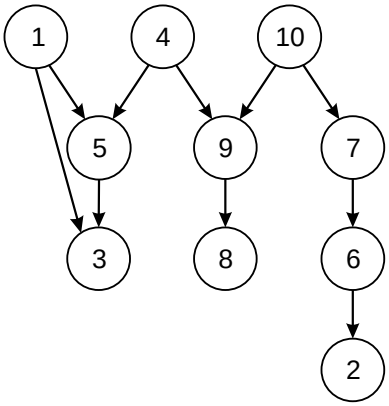

Figure 6.3: Vertical constraint graph G_v for the example in Figure 6.1 [86].

Zone Representation

If the horizontal segments of nets a and net b both intersect column i, then nets a and b cannot be assigned to the same track. Therefore, columns are divided into zones based on the horizontal segments of each net. Every zone designates a set of nets that cannot be placed on the same track and, hence, nodes a and b cannot be merged in the vertical constraint graph. The net merging algorithm tries to merge nets such that any two or more nets are assigned to a single track. Nets that share the same zone cannot be merged.

Table 6.1 provides the zone representation for the problem shown in Figures 6.1 and 6.2. Table 6.1 lists, for each column in the channel, the set of nets that intersect that column. Any two nets that intersect the same column cannot be assigned to the same track. For example, each of the nets $\{1, 2, 3, 4, 5\}$ has to be assigned to a different track since they all intersect column 3. Notice

that there is no need to consider columns 1 or 2 since both are subsets of column 3. Hence, columns 1 and 2 are in the same zone that includes column 3. A zone is any group of consecutive columns where all columns are subsets of one of the columns in the same group. The first zone covers columns 1 to 5 and includes the nets $\{1, 2, 3, 4, 5\}$ since all the columns in this zone are subsets of column 3 and, thus, cannot be assigned to the same track.

Table 6.1: Zone Representation Example [86]

Columns	**Nets**	**Zone**
1	2	1
2	1, 2, 3	
3	1, 2, 3, 4, 5	
4	1, 2, 3, 4, 5	
5	1, 2, 4, 5	
6	2, 4, 6	2
7	4, 6, 7	3
8	4, 7, 8	4
9	4, 7, 8, 9	
10	7, 8, 9	
11	7, 9, 10	5
12	9, 10	

The goal of the routing algorithm is to determine an optimum allocation of the nets such that

1. the vertical constraints expressed by G_v are satisfied, and
2. the number of tracks needed for realization is minimized.

The number of tracks needed is greater than or equal to the depth of the vertical constraint graph. Therefore, by minimizing the depth of the vertical constraint graph, we minimize the lower bound of the number of tracks. If i and j are nets for which

1. there exists no horizontal overlap, i.e., i and j are not in the same zone, and
2. there is no direct path between node i and node j in the vertical constraint graph,

then, net i and j can be placed on the same track. The net merging algorithm for a channel with m zones is shown in Figure 6.4. This algorithm assumes that no cycle exists in the initial vertical constraint graph.

```
    Algorithm NetMerge
    {
1.      L = {};
2.      for i = 1 to m − 1{
3.          L = L + {nets which terminate at zone Z_i};
4.          R = {nets which begin at zone Z_{i+1}};
5.          merge L and R so as to minimize the increase of the
                longest path length in the vertical constraint graph;
6.          L = L − {net merged at step 5};
        }
    }
```

Figure 6.4: Net merging algorithm for routing a channel with m zones.

6.2 Resolving Cyclic Conflicts

A disadvantage of the net merger algorithm is that it does not provide a solution when cyclic conflicts are present. A cyclic conflict occurs when a certain net a has to be above net b, while b also needs to be above a. An example of a cyclic conflict is shown in Figure 6.5. In this example, net a has to be above net b which has to be above net c which, at the same time, has to be above net a. Clearly, net a cannot be placed above c and below it at the same time. This section presents a new extension to Yoshimura and Kuh's algorithm to handle cyclic conflicts.

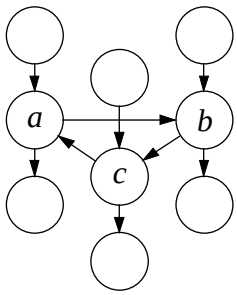

Figure 6.5: An example of a cyclic conflict.

6.2.1 Cycle Breaking

To remove the cyclic conflict, the cycle has to be broken. The method presented here breaks the cycle by splitting one of the cycle nodes into two nodes. The splitting of a node into two nodes should also minimize the increase in the number of horizontal tracks.

Two distance values u and d are calculated for each node, n. To calculate u and d, two "fictitious" nodes are added to the vertical constraint graph as shown in Figure 6.6 [86]. The node at the top is called the source, s, while the node at the bottom is called the sink, t. u is defined as the length of the longest path from s to node n [86]. Similarly, d is defined as the length of the longest path from node n to t.

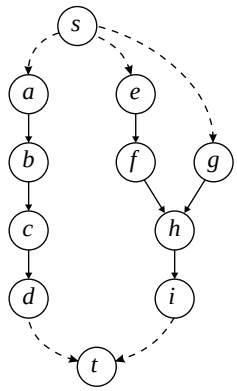

Figure 6.6: Imaginary nodes s and t used to calculate u and d.

When a node, say a, is split, it produces two nodes a_u and a_d. a_u is connected at the top of the broken cycle while a_d is connected at the bottom as shown in Figure 6.7. a_u gets all the edges directed from node a while a_d gets all the edges directed to a. This arrangement minimizes the increase in the number of horizontal tracks because it creates an overlap between the nodes above and below a. This overlap shortens the depth of the graph and, hence, the number of tracks.

For a cyclic path with n nodes, there are n possible ways to break the cycle. For every node in the cyclic path we calculate the depth of the graph if the cycle is to be broken at that node. The cycle is broken at the node that produces the least depth graph when broken. Assume a cycle of nodes $\{n_1, n_2, \ldots, n_n\}$. The following procedure determines which node n_i provides the least depth graph when broken.

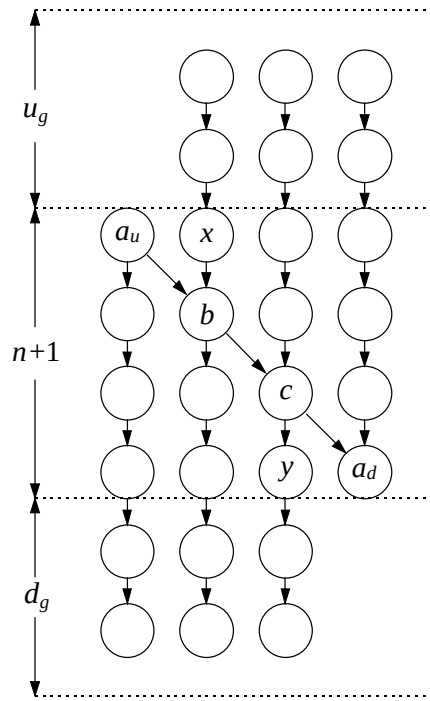

Figure 6.7: Breaking the cyclic conflict.

1. Calculate u and d for every node in the graph.

2. For every node in the cycle calculate the depth of the graph if the cycle is broken at this node. The following equations are used.

$$u_g(i) = \max(u_i - (n+1), u_{i-1} - n, u_{i-2} - (n-1), \ldots, u_{i-(n-1)} - 2) \quad (6.1)$$

$$d_g(i) = \max(d_i - (n+1), d_{i+1} - n, d_{i+2} - (n-1), \ldots, d_{i+(n-1)} - 2) \quad (6.2)$$

$$depth(i) = u_g(i) + (n+1) + d_g(i) \quad (6.3)$$

   Here, $1 \leq i \leq n$, n_{i+1} represents the node after node n_i in the cycle and n_{i-1} represents the node before node n_i in the cycle. $depth(i)$ is the depth of the graph if the cycle is broken at node n_i.

3. Break the cycle at the node that gives minimum *depth*.

The equations for $u_g(i)$, $d_g(i)$ and $depth(i)$ are easily derived from Figure 6.7.

6.2.2 Calculation of u and d

The definitions of u and d, as given in [86], do not hold when there is a cycle in the vertical constraint graph. This section extends the definitions of u and d to the cyclic conflict case.

For a node a that is part of a cycle, u is calculated based on the input edges to this node that are not related to the cycle. As for the nodes below a, u is calculated based on the worst case. The worst case is the largest u possible when the cycle is broken according to the technique described above. By looking at Figure 6.7, it is clear that node y, the node below the last node in the broken cycle, will have the largest u compared to all the nodes that are immediately below the cycle. Hence, for every node below a cycle, u is calculated assuming the above cyclic node is the last one after the split node in the broken cycle. d is calculated in a similar manner. By looking at Figure 6.7, it is clear that node x will have the largest d where node x is the node above the first node after the split node in the broken cycle. Hence, for every node above a cycle, d is calculated assuming the lower cyclic node is the first node after the split node.

One way to visualize how u and d are calculated in the presence of cyclic paths is to draw the cyclic path nodes at the same horizontal level as shown in Figure 6.8. The distance from the source to any node in loop is based on vertical edges only while the distance from the source to any node below the loop is calculated based on the greatest distance possible for the node if the loop is broken.

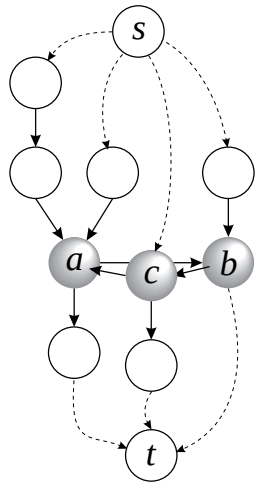

Figure 6.8: Visualization of u and d for cyclic paths.

If there is only one cyclic conflict in the graph, there is no need to calculate u for nodes below the cycle or d for nodes above the cycle. However, when there are two or more cycles in the graph and one cycle is above the other then we need the extended definitions of u and d. In this case, the first cycle considered is broken based on the worst case result of breaking the other cycles.

6.2.3 Connecting the Split Net

Since the two new nets, a_u and a_d, belong to a single net, they have to be reconnected at a some column in the channel layout. We search the channel for a column that can connect a_u and a_d without increasing the number of tracks. To facilitate this search we divide the columns of the channel into three regions, r_1, r_2 and r_3, as shown in Figure 6.9. Region r_1 designates the overlap between a_u and a_d, region r_2 designates the existence of a_u or a_d, but not both, and region r_3

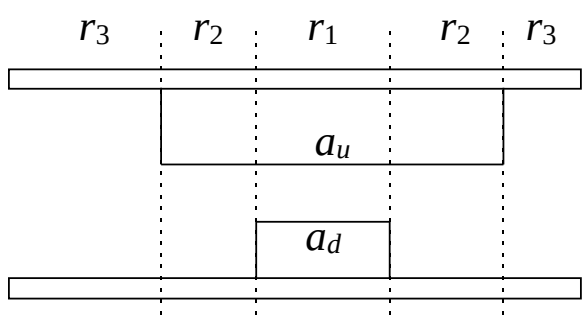

Figure 6.9: Dividing the channel into regions based on a_u and a_d.

designates the area beyond the limits of both a_u and a_d. The search starts in region r_1, if no suitable node is found, the search goes to region r_2, and if no solution is found then we search region r_3. In every region the search would be for the following types of columns in the given order.

1. (a_u, a_d): This is the best option since no extra column is used. This connection can exist only in region r_1.

2. $(a_u, 0)$ or $(0, a_d)$: This connection is always within region r_1 or r_2.

3. $(0, 0)$: This is sought in regions r_1, r_2 or r_3.

4. (a_u, q) or (p, a_d): This is always in region r_1 or r_2.

5. $(0, q)$ or $(p, 0)$: This is sought in regions r_1, r_2 or r_3.

6. (p, q): This is sought in regions r_1, r_2 or r_3.

The first three cases do not involve extra nodes, while the last three involve one or two extra nodes, q and/or p. p designates the net connected to the top of the reconnecting column and q designates the net connected to the bottom of the reconnecting column. Figure 6.10 shows an example of case 6 where a_u and a_d are connected in region r_3. The reconnecting of a_u and a_d under a terminal

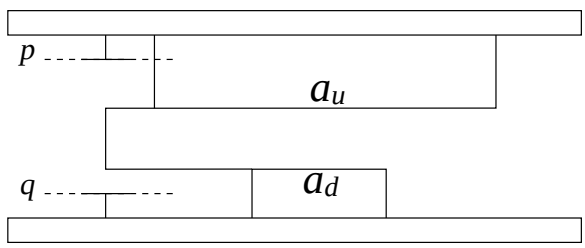

Figure 6.10: Example for connecting a net using extra nodes.

connected to net p and above a terminal connected to net q creates two new edges in the vertical constraint graph. The first edge is (p, a_u) while the second edge is (a_d, q) as shown in Figure 6.11. The goal of the search is to minimize the increase in the depth of the graph, thus, avoiding the increase in the lower bound of tracks. The search avoids the use of a column that is used by other nets, p and/or q, since the extra edges in the vertical constraint graph are likely to increase the depth of the graph. Two conditions are needed to ensure proper reconnecting of a_u and a_d. First, the new edges must not add new cyclic loops to the vertical constraint graph. Second, q and p should be chosen to minimize the increase in the number of tracks. A detailed discussion of restrictions on q and p is given in Section 6.2.4.

Table 6.2 shows the order of search through the three regions, r_1, r_2, and r_3, in the channel. If the search fails in all three regions, we ease the restrictions on p and q by allowing an increase in the depth of the graph.

6.2.4 Restrictions on p and q

Nodes p and q should be picked to minimize the number of horizontal tracks. If there is no other subgraph with a greater depth, the minimum number of tracks is the depth of the graph after

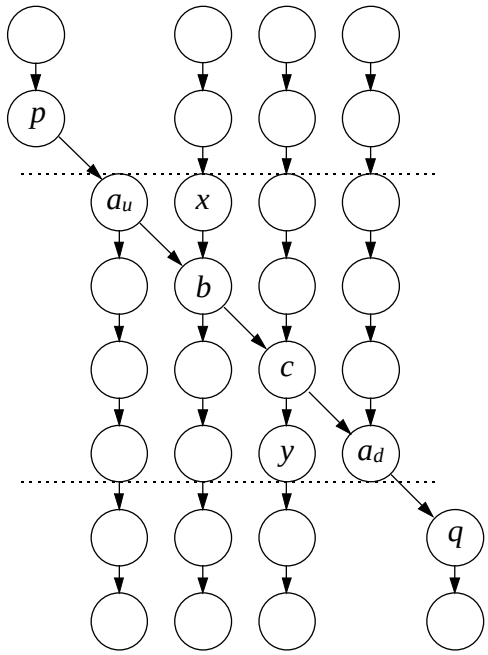

Figure 6.11: Connecting the broken cycle to p and q.

connecting p and q. For a graph with a single subgraph, if the cyclic path is broken at a certain node a, then the depth of the new graph is

$$depth = \max(u_g, u_p) + (n+1) + \max(d_g, d_q)$$

where u_g and d_g are calculated using node a from Equations 6.1 and 6.2, respectively. Also, u_p is the distance u for node p and d_q is the distance d for node q. Hence, to minimize the increase in the depth, p and q must adhere to the following conditions.

$$u_p \leq u_g$$

$$d_q \leq d_g$$

However, if there is another subgraph with depth d_s, as shown in Figure 6.12, such that

$$d_s > u_s + (n+1) + d_g$$

then, we look for p and q that satisfy the following

$$d_s \geq u_p + (n+1) + d_q \tag{6.4}$$

Table 6.2: Search Order for the Reconnecting Column

Rank	**Column**	**Region**
1	(a_u, a_d)	1
2	$(a_u, 0)$ or $(0, a_d)$	1
3	$(0, 0)$	1
4	(a_u, q) or (p, a_d)	1
5	$(0, q)$ or $(p, 0)$	1
6	(p, q)	1
7	$(a_u, 0)$ or $(0, a_d)$	2
8	$(0, 0)$	2
9	(a_u, q) or (p, a_d)	2
10	$(0, q)$ or $(p, 0)$	2
11	(p, q)	2
12	$(0, 0)$	3
13	$(0, q)$ or $(p, 0)$	3
14	(p, q)	3

If the reconnecting column contains an extra net, p, at the top and no net connected at the bottom, then the following equation is used

$$d_s \geq u_p + (n + 1) + d_g \tag{6.5}$$

and if the reconnecting column contains an extra net, q, at the bottom and no net connected at the top, then the following equation is used

$$d_s \geq u_g + (n + 1) + d_q \tag{6.6}$$

The above conditions are needed to minimize the increase in the depth of the graph. Another necessary condition is that the new edges, (p, a_u) and (a_d, q), must not introduce a new cycle to the graph. Hence, there should be no path from any of the cyclic path nodes to node p and there should be no path from q to any of the cyclic path nodes

6.2.5 Conflict Resolution Algorithm

The following summarizes the procedure used to resolve cyclic conflicts.

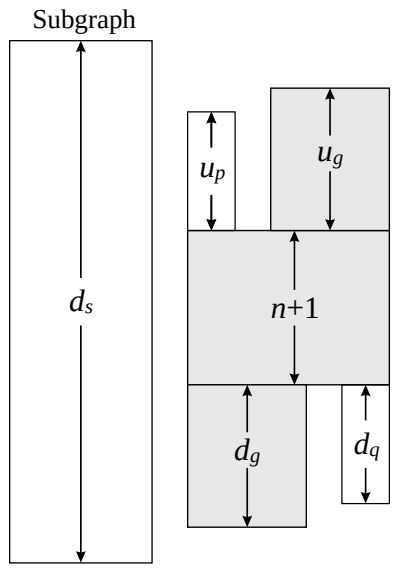

Figure 6.12: Effect of connecting p and q to the depth of the graph.

1. Identify all cycles in the vertical constraint graph, G_v. If there are no cycles then quit.
2. Calculate u and d for every node in G_v.
3. Pick a cycle to break. Identify the best location (node) to break this cycle.
4. Scan the channel for best location (column) to reconnect the broken net.
5. Update G_v and the zone representation, then go back to step 1.

6.3 Track Assignment

This section discuses the track assignment problem. The net merging channel routing algorithm proposed by Yoshimura and Kuh uses an arbitrary track assignment. This section presents an alternative track assignment scheme that makes the channel more testable.

The following are different ways to assign tracks.

1. Arbitrary track assignment (Yoshimura and Kuh [86]).
2. Track assignment that minimizes the critical area between nets [44, 68]. Kuo [44] reports

an insignificant contribution to the reduction in the critical area using this method, while Pitaksanonkul, *et al.* [68] report up to an 11 percent reduction in some cases.

3. Track assignment which avoids untestable bridging faults (McGowen and Ferguson [57]).

4. Track assignment which avoids bridging faults with low testability. For bridging faults, the testability can be calculated as bridging controllability. For example, if there are two nets, b and c, that can be assigned to the track below the track containing net a, select net b if $BC(a, b) \geq BC(a, c)$ and net c otherwise.

5. Track assignment which minimizes the sum of fault indices for faults inside the channel. The advantage of this method is it considers both the critical area and the bridging controllability.

This work uses the last method, minimization of the sum of fault indices. This method can be formulated as a traveling salesman problem [46] with an extra constraint. The constraint is called the precedence constraint which means that the salesman, in addition to minimizing the distance of the trip, must visit certain cities before others. The precedence constraint represents the constraints imposed by the vertical constraint graph G_v. Note that G_v used is the one produced after the end of the node merging step shown in Figure 6.4.

6.3.1 Formulation of the Track Assignment Problem

The track assignment problem requires the router to assign each node to a track. The vertical constraint graph adds a precedence constraint. For example, if node a precedes node b in G_v, then a must be assigned to a track that precedes the one to which node b is assigned. Minimizing the sum of fault indices requires a search for an optimal ordering for the nets within the tracks.

As stated above, this problem can be formulated as the traveling salesman problem [46] where the cities visited correspond to the nodes that need to be assigned to the tracks. The traveling salesman needs to visit all the cities in an order that minimizes the total journey distance. In this case, the distance between cities represents the fault index of a bridging fault that could occur between two nets assigned to consecutive tracks. The precedence constraints imposed by G_v are visualized as requiring that certain cities must be visited before certain other cities. For example, if there is an edge from node a to node b in G_v, this implies that city a must be visited before city

b. This problem is called the precedence constrained traveling salesman problem (PCTSP) [77, 31]. The following is a simple formulation of the PCTSP in terms of optimization theory followed by a more efficient, and also more complex, formulation.

6.3.1.1 Basic Formulation for PCTSP

Consider a set of nodes $0, 1, 2, \ldots, n$ where node 0 is a "dummy" node and nodes $1, \ldots, n$ need to be permuted. There is a cost of c_{ij} incurred if i immediately precedes j in this permutation, for $i, j = 1, \ldots, n$. Also, there exist certain precedence relationships among the nodes $1, \ldots, n$ where $i \rightarrow j$ implies that i should appear before j in a feasible permutation. We assume that a feasible permutation exists, i.e., the precedence structure is acyclic. This structure can be conceptualized as the precedence constrained traveling salesman problem via the use of the aforementioned dummy node 0, where the salesman begins at this "home city," node 0, traverses nodes $1, \ldots, n$ in some order that is feasible given the precedence constraints, and then returns to the home city, node 0, where $c_{0j} = c_{j0} = 0 \; \forall \; j = 1, \ldots, n$. To mathematically model this problem, define the following additional notation.

$$x_{ij} = \begin{cases} 1 & \text{if the salesman goes from city } i \text{ directly to city } j \\ 0 & \text{otherwise} \end{cases} \qquad i, j = 0, \ldots, n$$

$$u_j = \text{rank order in which city } i \text{ is visited} \qquad j = 0, \ldots, n$$

where $u_0 = 0$, and $1 \leq u_j \leq n$ for $j = 1, \ldots, n$.

The following presents a simple formulation of the traveling salesman problem with precedence constraint.

Minimize $\displaystyle\sum_i \sum_j c_{ij} x_{ij}$

subject to

$$\sum_{j \neq i} x_{ij} = 1 \qquad \forall \quad i = 0, \ldots, n$$

$$\sum_{i \neq j} x_{ij} = 1 \qquad \forall \quad j = 0, \ldots, n$$

Solution is a tour

Solution satisfies precedence constraints

x_{ij} binary

The first two constraints ensure that every city is visited exactly once. The third constraint is needed to ensure that the solution is a complete tour that covers all the cities rather than traversing several disconnected subtours. The precedence constraints are taken directly from the vertical constraint graph G_v. Finally, x must be a binary vector for the first two constraints to be meaningful.

The above formulation is not complete nor efficient from an optimization theory point of view [81]. The following presents a complete formulation that presents the constraints stated in words in a mathematical form and also tightens these constraints.

6.3.1.2 Tight MTZ–Based Formulation for PCTSP

Before presenting a formulation for the above constraints, we introduce the following definitions [77].

G_p: Precedence graph on nodes $1, \ldots, n$, where $(i, j) \in G_p \Leftrightarrow i$ precedes j. Note that this includes all transitions edges. An edge (i, k) is defined as a transition edge if there exists two other edges $i \rightarrow j$ and $j \rightarrow k$ in the same graph. Transition edges are redundant information and are eliminated using the following definition.

G_p^*: G_p - {transition edges}. Hence, G_p^* contains no edge (i, j) for which there exists an alternate path in G_p^* from i to j.

$F(i)$, $R(i)$: Set of forward related and reverse related nodes in G_p, respectively, for each node $i = 1, \ldots, n$. Node j is forward related to node i if the edge (i, j) exists in G_p. Similarly, node j is reverse related to node i if the edge (j, i) exists in G_p

$F^*(i)$, $R^*(i)$: Similarly defined with respect to G_p^*.

Figure 6.13 shows an example for precedence graphs G_p and G_p^* based on the vertical constraint graph, G_v, shown in Figure 6.3. Notice that G_p has more edges than G_v while G_p^* has fewer. In general, G_p will always have more edges than G_v, while G_p^* will have the same number or fewer edges than G_v. Also, both G_p and G_p^* should be constructed after node merging is performed in G_v. The example in Figure 6.13 is constructed before node merging is done for the sake of simplicity.

To construct an efficient formulation of the PCTSP, the constraints representing tours and precedence constrains must be developed as "tight" inequalities, i.e., as constraints that yield tighter or closer linear programming relaxations. The constraints that enforce the solution to be a

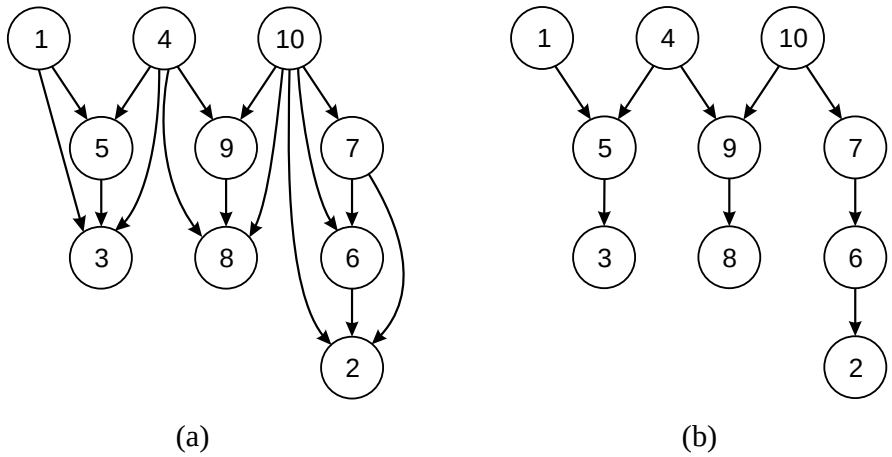

Figure 6.13: Precedence graph example: (a) G_p (b) G_p^*.

tour are called subtour elimination constraints. These constraints can be expressed using the Miller-Tucker-Zemlin (MTZ) method [60], for example. The MTZ method provides three equations based on the rank u_i of a node i. Precedence constraints are constructed as follows.

1. If a certain precedence constraint, $i \rightarrow j$, exists in G_p^*, then there should be no path from j to i.

2. If $i \rightarrow j$ and $j \rightarrow k$ precedences exist, then there can be no $i \rightarrow k$ (or $k \rightarrow i$) transition.

3. The rank of any node must be consistent with G_p^*. Hence, if i precedes j in G_p^*, then the rank of j must be greater by at least one than the rank of i.

4. Any node that has an input edge in G_p^* cannot be the first node visited. Likewise, any node that has an output edge in G_p^* cannot be the last node visited.

The following is a formulation for the PCTSP.

$$\text{Minimize} \quad \sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$\begin{aligned}
\sum_{j \neq i} x_{ij} &= 1 && \forall \quad i = 0, \ldots, n && (a) \\
\sum_{i \neq j} x_{ij} &= 1 && \forall \quad j = 0, \ldots, n && (b) \\
u_j &\geq (u_i + 1) - n(1 - x_{ij}) + (n-2)x_{ji} && \forall \quad i, j \geq 1,\ i \neq j,\ (i,j) \notin G_p - G_p^* && (c) \\
u_j &\geq 1 + (1 - x_{0j}) + (n-2)x_{j0} && \forall \quad j = 1, \ldots, n && (d) \\
u_j &\leq 1 + (n-2)(1 - x_{0j}) + x_{j0} && \forall \quad j = 1, \ldots, n && (e) \\
x_{ji} &= 0 && \forall \quad (i,j) \in G_p^* && (f) \\
x_{ij} &= x_{ji} = 0 && \forall \quad (i,j) \in G_p - G_p^* && (g) \\
u_i &\leq u_j - 1 && \forall \quad (i,j) \in G_p^* && (h) \\
x_{0j} &= 0 && \forall \quad j \ni R^*(j) \neq \phi && (i) \\
x_{j0} &= 0 && \forall \quad j \ni F^*(j) \neq \phi && (j) \\
u_0 &= 0 && && (k) \\
x_{ij} &= (0,1) && \forall \quad i, j = 0, \ldots, n && (l)
\end{aligned}$$

Constraints (c), (d), (e) and (k) are the subtour elimination constraints using Desrochers and Laporte's [22] tightening of the standard MTZ constraints [60]. Constraints (f)–(j) are precedence constraints derived from precedence graphs G_p and G_p^* as discussed above.

6.3.2 Track Assignment Algorithm

Due to the complexity of the above formulation of the track assignment problem, this work uses the following simple algorithm which uses heuristics to minimize the total fault index in the channel.

1. Start at the top track and set the current track to 1.

2. Find top nodes in the G_v, i.e., nodes with no input edges.

3. Assign the top node with the largest number of output edges to the current track. Any of the nodes with no input edges can be used as a starting point. Using the one with the largest number of output edges provides more choices for the second track.

4. Increment the current track.

5. Find all the nodes that can be assigned to the current track. This includes all the nodes that have no input edges and any node such that all of its input edges are already assigned to tracks. This condition ensures satisfying the precedence constraints.

6. Pick the node that produces the minimum fault index with the the node assigned to the above track considering only the horizontal critical area.

7. Repeat steps 4 through 6 till all nodes have been assigned.

This algorithm guarantees a complete tour and satisfies all the precedence constraints but does not provide an optimal solution. The assignment of nodes to consecutive tracks (step 4) ensures a single complete tour. Satisfaction of the precedence constraints occurs in step 5 while considering the nodes that can be assigned to the current track. Step 5 only considers nodes whose all ancestors are already assigned to tracks or nodes with no ancestors. The main heuristic used in the above algorithm is in picking the starting point which is the node assigned to track 1. A limitation of this algorithm is that the minimization of fault indices is done by minimizing the fault index between a certain track and the previous one regardless of how this would effect the sum of the fault indices in the channel.

6.3.3 Effect of Net Merging

The fact that each node may contain several nets after applying the net merging algorithm makes the implementation of the minimization process more complex. For example, if we need to calculate the resulting fault index due to assigning a certain node x which contains nets a and b just before or just after node y which contains nets c and d, then we must add the fault indices resulting from net a adjacent to nets c and d to the fault indices resulting from net b adjacent to nets c and d. The total of fault indices between these two nodes is then used in the minimization process.

6.4 Via Shifting Post-processor

To further reduce the critical area in the channel, a via shifting post-processor is used. The post-processor performs a via shifting based on a via sifting algorithm proposed by Kuo [44], but with several added enhancements.

The basic idea of the via shifting algorithm is to shift a via along a channel wire, thus, changing the layer of all or part of the wire. Since adjacent parallel wires are laid in the original layer, the critical area is eliminated or reduced depending on whether full or partial layer changing is performed. Partial layer switching is performed when full layer switching cannot be accomplished due to an intersecting wire. Via shifting reduces the critical area in both vertical and horizontal wires. The via shifting algorithm decreases the critical area without any increase in channel area. Figure 6.14 presents the via shifting algorithm. The via shifting algorithm developed here has the following enhancements over Kuo's algorithm [44].

1. When a via is connected to two wires, a horizontal wire and a vertical wire, Kuo shifts the via along the wire associated with a larger critical area. In this work, comparison is based on removable critical area rather than absolute critical area. It is common that even though a wire is associated with a large critical area, the via can be shifted along only a small portion of the wire because of intersecting wires that prevent the via from shifting along the entire wire. The amount of critical area that can be eliminated by the attainable via shifting is called the removable critical area. Removable critical area is always equal to or less than the absolute critical area associated with a wire.

2. In cases where switching the layer of a wire would not reduce critical area, the wire is still switched if this eliminates the via at the end of the wire. A layout with fewer vias provides better yield [44].

6.5 Results and Conclusions

This section introduces the YORCC (Yield Optimizing Router with Controllability Consideration) program which is a channel router and enhancer.

YORCC is the channel router and enhancer used in the PDFT design process. YORCC is based on YOR (Yield Optimization Router), a channel router developed by Kuo [44]. YOR consists of two parts, a router and post-processor. YORCC uses the net merging channel routing algorithm developed by Yoshimura and Kuh [86] with the cyclic channel extension presented in Section 6.2. For track assignment, YORCC uses a track assignment that minimizes fault index between wires

```
Algorithm ViaShifting
for(each via v){
    if(v has two branches b_h and b_v on different layers){
        C_h = size of the removable critical area adjacent to b_h;
        C_v = size of the removable critical area adjacent to b_v;
        if(C_h > C_v)
            Shift v along b_h;
        else if(C_v ≥ C_v)
            Shift v along b_h;
        else if(C_v = C_h = 0){
            if(b_v can switch layer){
                switch the layer of b_v;
                remove v;
                remove via at the other end if possible;
            }
            else if(b_h can switch layer){
                switch the layer of b_h;
                remove v;
                remove via at the other end if possible;
            }
        }
    }
    else {
        for(each branch b_vw){
            if(switching the layer of b_v reduces critical area){
                switch the layer of b_vw;
                if(all branches of w are on single layer)
                    delete w;
            }
        if(all branches of v are on single layer)
            delete v;
        else if(v has single horizontal layer branch)
            Shift v along the horizontal layer branch if it reduces critical area;
        else if(v has single vertical layer branch)
            Shift v along the vertical layer branch if it reduces critical area;
    }
}
```

Figure 6.14: Via shifting algorithm.

in the channel rather than minimizing the critical area as in YOR [44]. However, due to the complexity of implementing the precedence constrained traveling salesman problem formulated in Section 6.3.1, YORCC uses the heuristic-based algorithm presented in Section 6.3.2. A more elaborate implementation that utilizes the formulation in Section 6.3.1 is left for future work. Also, similar to YOR, YORCC includes a via shifting post-processor that further reduces the critical area and, hence, fault indices. The YOR post-processor also uses another method called net bumping to reduce critical area. Net bumping routes part of a wire on an adjacent track to remove critical area. Unlike via shifting, net bumping increases the length of wires and requires empty spaces in the channel. Net bumping is not implemented in YORCC.

YORCC is implemented in C++ on an IBM RS6000 SP2 workstation running the UNIX operating system. YORCC consists of about 18,000 lines of code. YORCC is designed to work as part of the OASIS [65] package.

The examples used to evaluate YOR in [44] are not suitable to evaluate YORCC since all of the examples are unconnected channels. To evaluate YORCC, the comparison must consider complete circuits since, otherwise, fault indices would be meaningless. Three of the large ISCAS 1985 benchmark circuits [13] were constructed using the OASIS system with YORCC as the channel router. Three layouts were generated for every circuit. The first layout was generated using Yoshimura and Kuh's algorithm without any enhancements. The second layout was generated using Yoshimura and Kuh's algorithm extended by the track assignment scheme developed here. The third layout utilized the track assignment and via shifting extensions. Table 6.3 compares the total fault index and Table 6.4 compares the total area.

Unfortunately, Tables 6.3 and 6.4 show that there was little or no improvement in the critical area or the fault index. To understand this result and to more accurately measure the effectiveness of YORCC, we look at the critical area and the fault index of the fault types affected by YORCC. Out of the eleven fault types listed in Figure 2.1, YORCC actually minimizes the fault indices for the following three types of faults.

1. wire to wire—same gate—input to input
2. wire to wire—same gate—input to output
3. wire to wire—different gates

Table 6.3: Total Critical Area ($10^9 \times centimicron^2$)

Circuit	Y&K	Y&K w/Track	Y&K w/Track and Via
c3540	98.92	99.01	98.08
c5315	202.64	202.43	199.50
c7552	275.72	276.37	273.60

Table 6.4: Total Fault Index

Circuit	Y&K	Y&K w/Track	Y&K w/Track and Via
c3540	1.696×10^{12}	1.698×10^{12}	1.696×10^{12}
c5315	7.506×10^{11}	7.501×10^{11}	7.445×10^{11}
c7552	1.935×10^{20}	1.935×10^{20}	1.941×10^{20}

Using the track assignment scheme and the via shifting scheme, YORCC minimizes the critical area for some of the faults that belong to the above three types. Other fault types should not be affected by YORCC. Section 8.4 discusses how different fault types are influenced by the tools developed in this work and compares the row enhancer and the channel enhancer.

Tables 6.5 and 6.6 give the total critical area and the total fault index, respectively, for the three fault types affected by YORCC. Tables 6.5 and 6.6 show that the contribution of the track assignment scheme is insignificant; there is less than a 1 percent reduction in the fault index created inside the channel. The reason for the marginal reduction is that the minimization of the fault index is based on information that is local to the channel itself, i.e., YORCC is unaware of the faults with the highest fault indices throughout the entire circuit. Instead, YORCC calculates the fault index for the faults in the channel based on the critical area component residing only in the given channel. Another reason is that the heuristics used are simple and do not provide an adequate implementation of the minimization criteria. The via shifting provided a minor reduction in critical area of about 2.5 percent for c3540 and c5315 and 7 percent for c7552.

While YORCC should not increase the critical area for other fault types, track assignment may increase the critical area between the wires running at the top and bottom of the channel and the V_{DD} and GND wires below and above the channel. Thus, the critical area for wire stuck-at faults

Table 6.5: Critical Area of Faults Enhanced by YORCC ($10^{10} \times centimicron^2$)

Circuit	**Fault Type**	**Y&K**	**Track**	**Track and via**
c3540	wire to wire—same gate—input to input	1.0472	1.0446	1.0370
	wire to wire—same gate—input to output	1.7067	1.7107	1.6882
	wire to wire—different gates	3.2604	3.2463	3.1761
	Sum	6.0143	6.0015	5.9013
c5315	wire to wire—same gate—input to input	2.0246	2.0309	1.9945
	wire to wire—same gate—input to output	2.4901	2.4852	2.4372
	wire to wire—different gates	9.2611	9.2179	8.9928
	Sum	13.776	13.734	13.425
c7552	wire to wire—same gate—input to input	2.1624	2.1629	2.1436
	wire to wire—same gate—input to output	4.0757	4.0704	4.0102
	wire to wire—different gates	12.089	12.099	11.887
	Sum	18.327	18.332	18.041

Table 6.6: Fault Index of Faults Affected by YORCC

Circuit	**Fault Type**	**Y&K**	**Track**	**Track and via**
c3540	wire to wire—same gate—input to input	3.17356×10^{10}	3.15238×10^{10}	3.13625×10^{10}
	wire to wire—same gate—input to output	2.72281×10^{10}	2.72816×10^{10}	2.69641×10^{10}
	wire to wire—different gates	8.24308×10^{10}	8.13858×10^{10}	7.95745×10^{10}
	Sum	1.41395×10^{11}	1.40191×10^{11}	1.37901×10^{11}
c5315	wire to wire—same gate—input to input	5.12721×10^{10}	5.14655×10^{10}	5.06091×10^{10}
	wire to wire—same gate—input to output	3.39437×10^{10}	3.38736×10^{10}	3.32375×10^{10}
	wire to wire—different gates	2.06796×10^{11}	2.05829×10^{11}	2.00764×10^{11}
	Sum	2.92012×10^{11}	2.91168×10^{11}	2.84611×10^{11}
c7552	wire to wire—same gate—input to input	2.32056×10^{16}	2.32056×10^{16}	2.16244×10^{16}
	wire to wire—same gate—input to output	5.31229×10^{10}	5.30646×10^{10}	5.23034×10^{10}
	wire to wire—different gates	9.36819×10^{13}	9.36775×10^{13}	8.40538×10^{13}
	Sum	2.32993×10^{16}	2.32993×10^{16}	2.17085×10^{16}

may increase slightly. This is the reason for some minor increases in critical area in Table 6.5 when track assignment is used.

Kuo also reports insignificant reduction in critical area using the track assignment and about 8 to 26 percent reduction using via shifting. Note that Kuo's goal for the track assignment is to reduce the total critical area inside the channel, while the goal of this work is to reduce the fault index. Nonetheless, both schemes produce insignificant results. Kuo's examples are small non-cyclic channels chosen from a set of problems used by the routing community to evaluate routers [44]. The examples used in this work are more realistic complete circuits.

Future versions of YORCC could utilize the full set of information provided by BRICON to recognize the faults with high fault indices throughout the entire circuit. Also, future work on via shifting should give priority to faults with high fault indices. This requires a new algorithm that also uses feedback information from BRICON and scans for faults with high fault indices and corrects them first. See Section 8.4 for more discussion about potential improvements in YORCC.

6.6 Summary

Channel routing is a standard problem in VLSI design. This chapter presents a channel router that supports PDFT. The router is based on the net merging channel routing algorithm. The net merging algorithm is extended to handle cyclic cases. A new track assignment scheme that minimizes the fault index between horizontal wires in the channel is presented. The router also includes a via shifting post-processor which minimizes critical area.

Unfortunately, the reductions in critical area and fault index are marginal. The limited knowledge that the channel router has about faults in the entire circuit leads to this marginal reduction in fault index. Future versions of the channel router should have more knowledge about faults throughout the entire layout.

Chapter 7

Design of a Defect-Tolerant Cell Library

This chapter presents the new defect-tolerant CMOS cell library designed for producing CMOS circuits with a reduced likelihood of bridging faults. Each cell in the defect-tolerant library is derived from an equivalent function cell in the MCNC standard cell library. Section 7.1 introduces standard cell design and discusses the need for a defect-tolerant cell library. Section 7.2 presents the defect-tolerant library designed for this work. An evaluation of the new fault-tolerant cell library, including a detailed comparison with the original normal cell library, is given in Section 7.3. Section 7.4 discusses the vulnerability of the new library to break faults.

7.1 Standard Cell Design

Standard cell VLSI design implies the interconnection of predesigned function blocks using a complex routing procedure [42]. These function blocks are called *standard cells*. Standard cells with similar design requirements are grouped into a cell library. Designers traditionally design cells based on specific design rules while minimizing area and delay. In general, no consideration is given to the occurrence of physical faults. Designers assume that if design rules are followed correctly then the effect of physical defects is minimized. In reality, defects do occur even when design rules are obeyed. In this work we design a small cell library that is defect-tolerant in terms of bridging faults. No new design rules are suggested since this would create an unrealistic increase in the area.

Instead, a set of guidelines that minimize critical area associated with bridging faults is proposed. These guidelines provide a design style for designing defect-tolerant standard cells.

7.2 A Defect-Tolerant Cell library

Each cell in the defect-tolerant cell library is designed by starting with the cell from the MCNC standard cell library having the equivalent function. Extra area is added to this cell, then the cell is redesigned to utilize the extra area. The modified cell is then analyzed by the fault extractor, CARAFE [38]. The output of CARAFE, a list of potential bridging faults, is processed by a script to calculate the total critical area associated with bridging faults in the cell. Also, the script divides every bridging fault into sub-faults based on the layers shorted by the sub-fault. Then, the script reorders the sub-faults depending on the associated critical area. This ordering makes it easy for the circuit designer to track every fault and to fine tune the cell to minimize the total critical area. The new design is fed back to CARAFE and after a few iterations the designer usually reaches a point where no further minimization in critical area can be achieved without a further increase in cell area. A new design style, that is more practical than using a new set of design rules, was derived from this process.

It is important to note that the defect-tolerant library and the normal library will be combined to form a multiple-layout cell library. To be able to combine them in a single library, the defect-tolerant library must follow all of the restrictions imposed on the normal library. The MCNC standard cell library places the following requirements on standard cells [65].

1. Cells must have a uniform height of 58λ.
2. Power and ground connections must be horizontal wires of layer metal-1 along the top and bottom of the cell, respectively.
3. Input/output signal wires must be run vertically in layer metal-2.
4. The width of the cell and the location of the input/output wires must follow a grid of 8λ.

These restrictions are mandated by the OASIS placement and routing algorithms [65] which require a grid. Gridless routers [18, 19] impose fewer restrictions on the design at the expense of more complex algorithms. The above restrictions on cell layouts limit the freedom of the designer. The

height of the cell cannot be extended while the width of a cell can only be extended according to the grid, in multiples of 8λ in this case. In this work the minimum amount of area is added to every normal cell to derive the defect-tolerant version of the cell, i.e. every defect-tolerant cell has the same height of the original normal cell while its width is equal to the width of the normal cell plus 8λ. Unfortunately, this means that the percentage of added area is high for small cells while it is low for large cells. It would have been preferable if the area of every cell in the normal library could be expanded by a fixed percentage.

The normal library used in this work is a subset of the MCNC standard cell library [65]. The normal library consists of the following seven cells. These cells are required for experiments with the ISCAS 1985 [13] benchmark circuits and are adequate to demonstrate the concept of a multiple layout cell library.

1. **i1**: an inverter.
2. **ai2**: a two-input NAND gate.
3. **ai3**: a three-input NAND gate.
4. **ai4**: a four-input NAND gate.
5. **oi2**: a two-input NOR gate.
6. **oi3**: a three-input NOR gate.
7. **oi4**: a four-input NOR gate.

In addition to these seven cells, there are two other cells that have no logic function: **fts** and **ftes**. **fts** is a pass-through cell consisting of two horizontal metal-1 wires at the top and bottom and one vertical metal-2 wire in the middle. The global routing program inserts a pass-through cell in the channel whenever it needs to wire a certain node through that channel. **ftes**, which is similar to **fts** but with no vertical wire, is used at the left and right ends of channels to make all the channels have equal width.

Each of the seven cells is expanded and the design process mentioned above is applied until a satisfactory defect-tolerant version is produced. **fts** and **ftes** are not altered due to restrictions imposed by the global routing program.

Several techniques were used to produce the defect-tolerant library. The techniques discovered through the redesign phase are listed below.

1. Utilize the extra area to spread wires apart to reduce intra-layer bridging faults.

2. Minimize any overlap that causes inter-layer bridging faults. Inter-layer bridging faults should be given priority compared to intra-layer bridging faults since the critical area due to inter-layer bridging faults was typically twice that of the critical area due to intra-layer bridging faults in the defect-tolerant cells.

3. Use straight transistors inside diffusion regions to minimize diffusion-to-diffusion bridging faults.

4. Do not increase channel length, otherwise the delay will increase to the extent that the fault-tolerant cells will have unacceptable delay. In this work the original dimensions of the transistor channels were maintained as much as possible. This ensures that the increase in delay and input capacitance is minimized for the defect-tolerant cells compared to their normal counterparts.

Figure 7.1, which shows the normal and defect-tolerant layouts for **ai2**, illustrates the design techniques. The extra area in the defect-tolerant layout is used to increase the distance between the vertical metal-2 wires. Also, the distance between the p-transistors and the distance between the n-transistors are increased. Inter-layer bridging faults are minimized by shifting the n-diffusion region up, thus eliminating the overlap between $n1$ and GND and the overlap between q and GND. Also, by shifting the horizontal metal-1 wire below the p-diffusion region, the overlap between q and V_{DD} is eliminated. The n-transistor at the left is made straight in the defect-tolerant layout to minimize the critical area between $n1$ and GND inside the n-diffusion area. The channel widths are maintained at 2λ to prevent any significant increase in the delay.

7.3 Evaluation of the Fault-Tolerant Library

This section evaluates the new defect-tolerant library and compares it to the original normal library. The evaluation uses four parameters: total area, total critical area, delay and input capacitance.

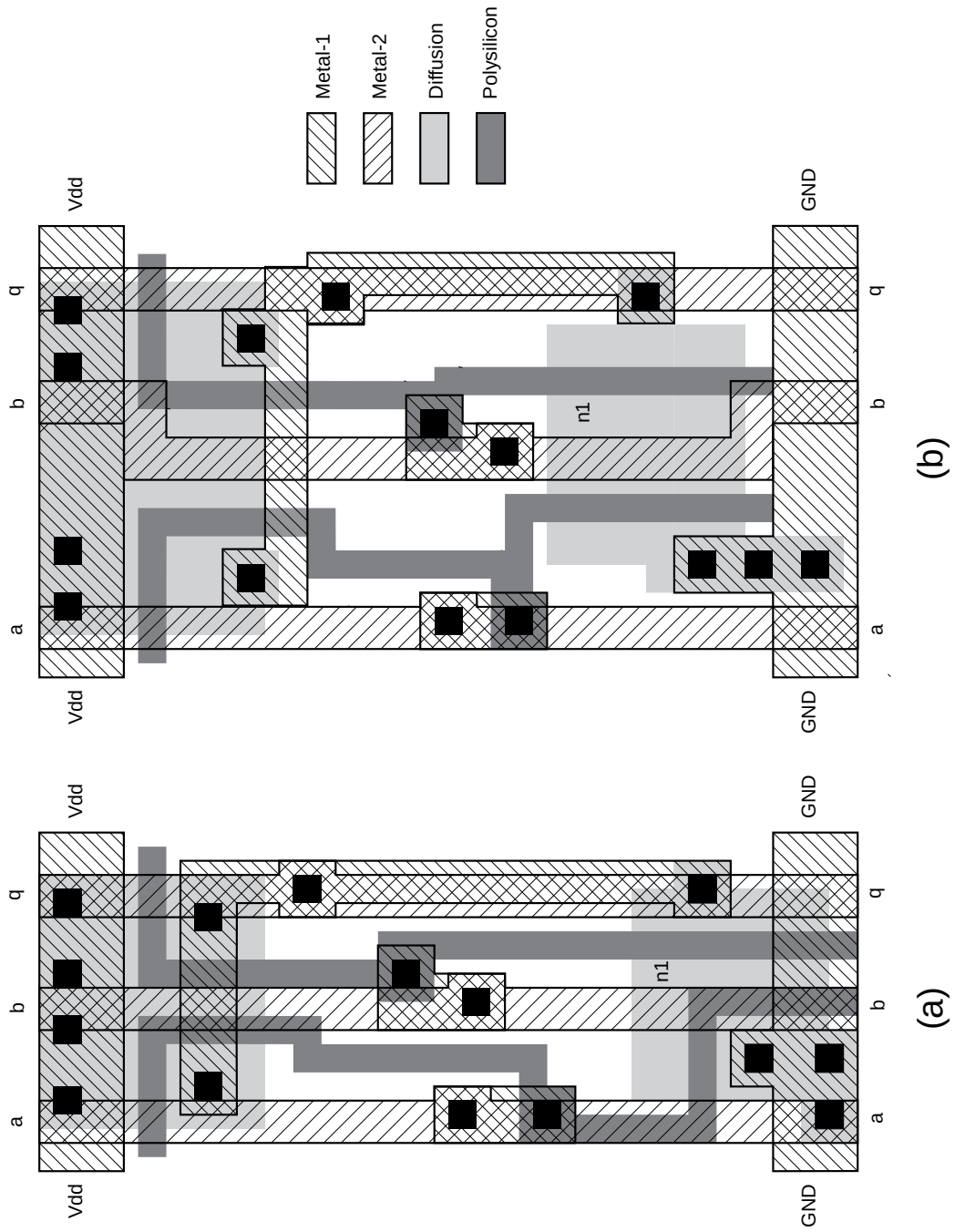

Figure 7.1: Layout of cell **ai2**. (a) Normal layout. (b) Defect-tolerant layout.

Originally, this work intended to limit the increase in cell area to 20 percent. However, due to the grid restriction discussed in Section 7.2, the increase in area is always $464\lambda^2$ since the cells are 58λ high and the width is increased by 8λ. This increase is the minimum possible given the grid restriction. Note that the design methodology proposed requires the ability to mix the normal cells and the defect-tolerant ones on the same layout, hence, the new library is actually an extension for the normal library. Defect-tolerant cells must have exactly the same height as the normal ones while the increase in width should adhere to the grid. The restriction on the height and the increment of increase in the width leads to an unfortunate variation in the percentage of area increase. For example, the defect-tolerant inverter cell has a 50 percent increase in area over the normal inverter, while the defect-tolerant two-input NAND gate has only a 20 percent increase in area compared to its normal counterpart.

Tables 7.1 and 7.2 compare the two libraries in terms of total area and total bridging fault critical area, respectively, as reported by CARAFE.

Table 7.1: Total Area (λ^2) for Normal and Fault-Tolerant Cell Libraries

Gate	**Total Area**		**Increase (%)**
	Normal	**Defect-Tolerant**	
i1	16	24	50.00
ai2	24	32	33.33
ai3	32	40	25.00
ai4	40	48	20.00
oi2	24	32	33.33
oi3	32	40	25.00
oi4	40	48	20.00
		Average Increase	29.52

OASIS characterizes the delay of a cell at every input node using two parameters: *delay* and *fan* [65]. *Delay* denotes the zero load delay while *fan* denotes the incremental increase in delay as a function of the load. Different load capacitances are applied at the circuit output, then the delay associated with every input node is measured. The delay is measured as the time difference between the 50 percent input transition level and the 50 percent output level. Two delays are measured for each input node: the rise delay and the fall delay. Rise delay and fall delay are measured when the transition at the input node causes the output node to switch from 0 to 1 and from 1 to 0,

Table 7.2: Total Critical Area ($10^6 \times centimicron^2$) for Normal and Fault-Tolerant Cell Libraries

Gate	**Total Critical Area**		**Decrease (%)**
	Normal	**Defect-Tolerant**	
i1	14.5	6.8	52.86
ai2	34.1	16.5	51.62
ai3	62.1	31.6	49.18
ai4	92.3	68.6	25.65
oi2	57.1	22.3	60.87
oi3	73.8	41.8	43.27
oi4	81.8	59.4	27.37
		Average Decrease	44.40

respectively. A straight line that connects different delays for different loads for every input node is drawn. *Delay* is set to the intersection of this line with the y axis while *fan* is set to the slope. Figure 7.2 shows how *delay* and *fan* are calculated [65]. Table 7.3 summarizes the delay values for the defect-tolerant and normal libraries. In general, the zero load delay, *delay*, of fault tolerant cells is higher than the normal cells while the fan increase is the same, i.e. defect-tolerant cells always start with a higher delay but the increase in delay due to load capacitance is similar.

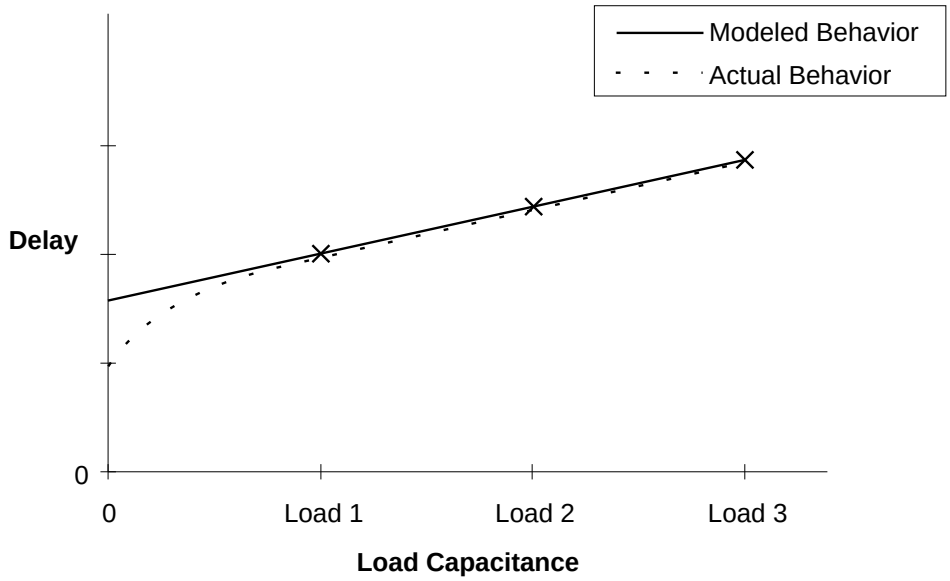

Figure 7.2: Calculation of *delay* and *fan* [65].

Table 7.3: Delay (nanosecond) for Normal and Fault-Tolerant Cell Libraries

Gate	Node	Normal		Defect-Tolerant		Delay	Fan
		Delay	Fan	Delay	Fan	Increase (%)	Increase (%)
i1	a rise	0.1713	2.8048	0.2102	2.7837	22.69	-0.75
	a fall	0.1375	2.6830	0.1774	2.6852	28.95	0.08
ai2	a rise	0.3811	2.4858	0.4020	2.3021	5.49	-7.39
	a fall	0.2440	2.0733	0.2885	2.1982	18.23	6.03
	b rise	0.2311	2.4716	0.2421	2.3060	4.75	-6.70
	b fall	0.1673	2.0777	0.1869	2.1999	11.70	5.88
ai3	a rise	0.5465	2.2655	0.7241	2.5505	32.49	12.58
	a fall	0.3742	2.0994	0.4368	2.2323	16.73	6.33
	b rise	0.4821	2.4948	0.5299	2.4633	9.90	-1.26
	b fall	0.3038	2.0917	0.3376	2.2349	11.13	6.85
	c rise	0.2798	2.6289	0.3619	3.0938	29.37	17.68
	c fall	0.1814	2.0923	0.2012	2.2275	10.91	6.46
ai4	a rise	0.8992	2.3716	1.1040	2.3951	22.77	0.99
	a fall	0.6079	2.0947	0.7274	2.1156	19.66	1.00
	b rise	0.7443	2.2710	1.0555	2.5472	41.81	12.17
	b fall	0.5159	2.1161	0.6134	2.0719	18.89	-2.09
	c rise	0.5967	2.2151	0.6775	2.1964	13.54	-0.85
	c fall	0.4174	2.1183	0.4572	2.1147	9.56	-0.17
	d rise	0.3522	2.1671	0.4165	2.1798	18.27	0.59
	d fall	0.2194	2.0955	0.2476	2.0849	12.88	-0.51
oi2	a rise	0.2696	2.4724	0.2869	3.0929	6.42	25.10
	a fall	0.2301	2.6871	0.2003	2.6956	-12.97	0.32
	b rise	0.3413	2.4705	0.3747	3.0981	9.79	25.41
	b fall	0.4251	2.7467	0.3867	2.7454	-9.02	-0.05
oi3	a rise	0.7142	3.4398	0.8040	3.7556	12.57	9.18
	a fall	0.6674	2.9446	0.6810	2.8919	2.03	-1.79
	b rise	0.6212	3.4623	0.6880	3.8271	10.76	10.54
	b fall	0.4973	2.7394	0.5144	2.7395	3.44	0.01
	c rise	0.3834	3.4907	0.4214	3.8639	9.91	10.69
	c fall	0.2786	2.6763	0.2856	2.6883	2.49	0.45
oi4	a rise	0.4461	4.8941	0.4213	4.9332	-5.56	0.80
	a fall	0.2839	2.6803	0.3050	2.6880	7.43	0.29
	b rise	0.8594	4.8418	0.8974	4.9883	4.43	3.02
	b fall	0.4640	2.7348	0.5235	2.7441	12.83	0.34
	c rise	1.1344	4.8903	1.2038	4.9978	6.12	2.20
	c fall	0.6227	2.9115	0.6899	2.8955	10.80	-0.55
	d rise	1.2002	4.8646	1.2759	5.0274	6.31	3.35
	d fall	0.6930	3.0436	0.7612	3.0685	9.84	0.82

Table 7.4 lists the input capacitance for every node for each cell in the cell library. The change in the input capacitance was insignificant because the defect-tolerant library maintained the same channel dimensions as the normal library.

Table 7.4: Input Capacitance (picofarads) for Normal and Fault-Tolerant Cell Libraries

Gate	**Node**	**Input Capacitance**		**Increase (%)**
		Normal	**Fault-Tolerant**	
i1	a	0.0524	0.0546	4.31
ai2	a	0.0830	0.0828	-0.30
	b	0.0671	0.0714	6.42
ai3	a	0.1002	0.0906	-9.53
	b	0.0723	0.0684	-5.47
	c	0.0701	0.0650	-7.23
ai4	a	0.1010	0.0977	-3.20
	b	0.0911	0.0881	-3.28
	c	0.0899	0.0889	-1.06
	d	0.0987	0.0944	-4.28
oi2	a	0.0786	0.0711	-9.60
	b	0.0906	0.0819	-9.55
oi3	a	0.0983	0.0895	-8.95
	b	0.0831	0.0764	-7.99
	c	0.0636	0.0606	-4.80
oi4	a	0.0625	0.0618	-1.13
	b	0.0750	0.0734	-2.09
	c	0.0742	0.0746	0.53
	d	0.0888	0.0908	2.20

7.4 Susceptibility to Break Faults

Although break faults are beyond the scope of this work, it is interesting to note how the likelihood of break faults is influenced by designing cells to be tolerant of bridging faults. This section discusses published techniques to design cells that are tolerant of break faults and examines whether the design guidelines presented in the last section conflict with those techniques. Also, a small experiment is conducted by running CARAFE on the cell library and comparing the critical area associated with break faults in the normal and defect-tolerant cell libraries.

Teixeira, *et al.* [83] suggest the following rules to avoid break faults in CMOS circuits.

1. Avoid metal-1 crossing over polysilicon lines.
2. Use wide metal power lines.
3. Use redundant closed loop power lines.
4. Connect CMOS gates in polysilicon.
5. Use redundant contacts in parallel to connect different layers.
6. Connect parallel transistors with diffusion layer.

These rules are not in conflict with the design guidelines proposed in the previous section. In fact, rule 1 is a special case of technique number 2 which states that overlaps between layers that could cause inter-layer bridging fault should be minimized. Rules 2 and 6 are also followed in designing both the normal and the defect-tolerant library. Rules 3, 4 and 5 are not satisfied in either the normal or the defect-tolerant cell libraries. This leads to the conclusion that the design style used to produce the defect-tolerant cell library should not have a significant negative impact on the occurrence of break faults. To strengthen this conclusion, the normal cell library and the defect-tolerant cell Library were both analyzed using CARAFE to determine the critical area associated with break faults for both libraries. Table 7.5 summarizes the results of this experiment. The results in Table 7.5 support the claim that the design guidelines given in Section 7.2 do not have a significant negative impact on the likelihood of break faults. The minor increase in break fault critical area in most cases is due to the extension of some wires in the defect-tolerant version either

Table 7.5: Critical Area ($10^6 \times centimicron^2$) for Break Faults for Normal and Fault-Tolerant Cell Libraries

Gate	**Critical Area**		**Increase (%)**
	Normal	**Defect-Tolerant**	
i1s	9.483	9.828	3.64
ai2s	14.528	14.215	-2.15
ai3s	18.345	18.703	1.95
ai4s	22.008	22.460	2.06
oi2s	14.175	15.138	6.79
oi3s	16.660	17.828	7.01
oi4s	20.440	21.520	5.28

to reroute the wire away from another wire or because the two points connected by the wire were further from one another when the cell area was extended.

7.5 Summary

The defect-tolerant cell library is derived from the OASIS standard cell library by expanding the area of the cells and then redesigning the cells to reduce the critical area. Several critical area reduction techniques were developed during the redesign stage. These techniques do not depend on the structure of OASIS cell library and should be applicable to any other cell library. Although the defect-tolerant cells and their normal counterparts in the library may not scale easily to other technologies, the techniques used to derive the defect-tolerant library are not technology dependent and can be used with other technologies. On average, the defect-tolerant layouts have 44 percent less critical area at the expense of a 30 percent increase in area. In general, the defect-tolerant cells have somewhat higher delay than their normal counterparts.

Chapter 8

The Row Enhancer

This chapter introduces the row enhancement program, ROWEN. ROWEN is the feedback element in the PDFT design process. After generating and evaluating a circuit layout, the designer can invoke ROWEN to start the process of constructing a new layout for the circuit. The new layout replaces some of the normal cells with their defect-tolerant counterparts. ROWEN examines a certain number of faults with the highest fault indices and finds which gates, if switched to defect-tolerant layouts, will minimize the fault indices for these faults. These gates are called the fault-related gates and are defined as the cells that have critical area rectangles belonging to the fault located inside them. ROWEN compiles a list of the gates that should use defect-tolerant layouts the next circuit layout. The next layout should have lower fault indices for the faults considered. Many other faults will also have lower fault indices.

Section 8.1 describes inputs, outputs and different running modes of ROWEN. Fault handling method is presented in Section 8.2. Section 8.3 presents an experiment to demonstrate the reduction in total fault index when ROWEN is used. Section 8.4 provides a comparison between the row enhancer and the channel enhancer.

8.1 Interface

The purpose of ROWEN is to identify the cells in the circuit such that if their layouts are switched from normal to defect-tolerant, the fault indices for faults with high fault indices will be reduced. Hence, ROWEN needs information about the bridging faults with high fault indices and information

about critical area located inside every cell. The main output of ROWEN is a list of gates which is a subset of the netlist. ROWEN's calculations indicate that switching these gates to their defect-tolerant version would reduce the fault indices of some of the faults and, hence, reduce the total fault index. The following is a detailed description of inputs, outputs and ROWEN's different operating modes.

8.1.1 Inputs

ROWEN's main inputs are the gate-level circuit netlist and the list of bridging faults. The netlist is the same one used by the placement and routing program, VPNR [65]. The list of bridging faults, generated by BRICON, includes the probability of occurrence, the bridging controllability and the fault index for each fault and is sorted by fault index. Also, ROWEN reads the circuit's MAGIC files to determine the layout area.

In addition to the above inputs, ROWEN includes a database of information about the standard cells in the cell library. The database provides ROWEN with every possible bridging fault that is internal to the cell. Moreover, the database lists the critical area associated with every possible fault for both the normal and the defect-tolerant versions. This information is compiled by running every cell by itself through CARAFE to extract all possible internal bridging faults and their associated critical areas. The database also lists the area for every cell in the library.

8.1.2 Modes and Options

ROWEN runs in two modes: trim fault index mode and minimize fault index mode. In the trim mode the system designer provides ROWEN with a threshold level, l, instructing ROWEN to track every fault with a fault index above l and to try to bring the fault index down to a level equal to or less than l. Another way to use the trim mode is for the system designer to specify a number of faults, n, to be considered. In this case, the threshold level l is set to the fault index of the nth fault. In the minimize mode the designer provides ROWEN with a number of faults n. ROWEN minimizes the fault index for the top n faults in the fault list as much as possible. Only the minimize mode is used in the PDFT design process developed here.

There is no need for ROWEN to trace a large number of faults. In most cases tracing 100 to 500 faults should produce adequate reduction in critical area or fault index. See Section 4.2 which

provides an example where the top 100 faults contribute 40 percent of the total fault index.

In addition to minimizing the total fault index, ROWEN can be used to minimize the total critical area of the circuit. In this case the fault list must be sorted according to critical area rather than fault index. Also, testability information produced by BRICON is not utilized.

8.1.3 Output

ROWEN generates two main outputs. The first output is a list of gates that includes all the normal-layout gates in the current circuit layout that should be converted to defect-tolerant layout in the next generated circuit layout. The other output is a set of estimates for the next circuit layout. This set includes the area, critical area, and fault index. These estimates are for reference only, they are not used in later processing. Critical path delay for the next layout can also be estimated at this stage using CRITICAL [65].

8.2 Fault Handling

The critical area for any bridging fault consists of two components, logic and interconnection [25]. The logic part of the critical area is the sum of the critical areas of all bridges located inside the fault-related gates. The interconnection part of the critical area is the sum of the critical areas of all the bridges in the routing part of the circuit or due to a certain placement situation. Figure 8.1 shows an example of a stuck-at-0 fault with both types of critical area components. If a certain fault has a logic component in its critical area, ROWEN looks further into the fault to check if it is possible to reduce the fault index. Reduction of the fault index is done by minimizing the critical area of the fault. ROWEN can only minimize the logic critical area, thus, faults with interconnection critical area only are ignored. ROWEN reduces the logic critical area by tracing the critical area to every gate related to the fault. A gate is related to the fault if part of the fault's critical area occurs inside the corresponding cell layout. Then, ROWEN checks if switching the gate to the defect tolerant version will reduce the logic components of critical area. If this is the case, ROWEN adds this gate to the output gate list.

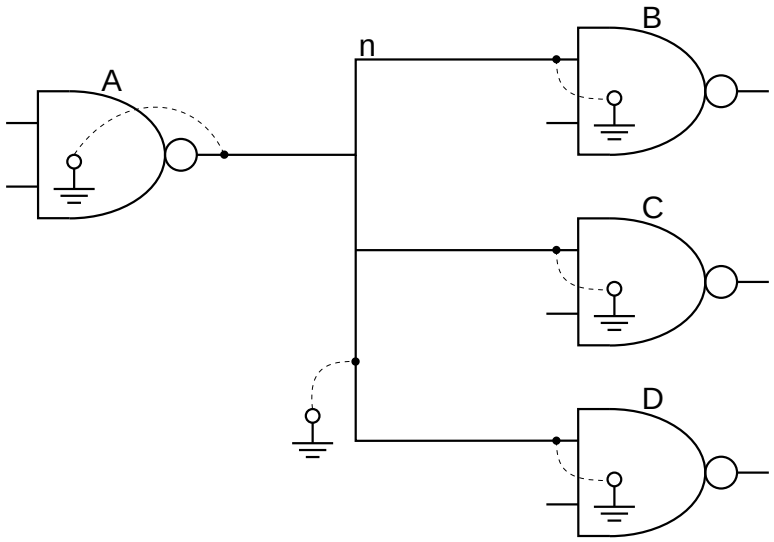

Figure 8.1: Example of multiple same-gate fault (stuck-at-0).

8.2.1 Fault Classification

To trace a fault to its related gates, ROWEN utilizes the bridging fault classification scheme proposed by Midkiff and Bollinger [59]. ROWEN divides these classifications into three groups based on the type of related gates as shown in Figure 8.2. The following is a description of the three fault groups.

1. *Multiple same-gate.* In this category, the two nodes of the fault are both connected to several gates and the two nodes are shorted inside some or all of these gates. This creates a bridging fault that consists of several bridging faults occurring at different gates, yet, these faults are equivalent at the gate-level or switch-level and hence, represented by a single bridging fault. This fault has a critical area that is equivalent to the sum of the critical areas of the smaller faults. Moreover, some of the smaller faults may be interconnection faults, i.e. caused by placement and routing.

   Figure 8.1, above, shows an example of a multiple same-gate fault of type wire-stuck-at-0. Node n is shorted to ground by several bridging subfaults. The first subfault, at the left, is caused by a critical area located inside gate A. The three subfaults, at the right, are caused by critical areas located inside gates B, C and D. These four subfaults are all logic faults.

Multiple Same-Gate

1.1	Wire Stuck-At
1.2.1.1	Wire to Wire—Same Gate—Input to Input
1.2.1.2	Wire to Wire—Same Gate—Input to Output

Single Same-Gate

2.1	Internal Node Stuck-At
2.2	Transistor Stuck-On
2.3.1.1	Wire to Internal Node—Input to Node—Same Gate
2.3.1.2	Wire to Internal Node—Output to Node—Same Gate
2.4.1	Internal Node to Internal Node—Same Gate

Different Gates

1.2.2	Wire to Wire—Different Gates
2.3.2	Wire to Internal Node—Different Gates
2.4.2	Internal Node to Internal Node—Different Gates

Figure 8.2: Bridging fault categorization used by ROWEN (fault class numbers are taken from Figure 2.1).

The middle subfault caused by a critical area located at the wiring between gate A and the other gates, typically in the channel area, is an interconnection subfault.

2. *Single same-gate.* This category includes fault types in which the two nodes of the fault are shorted by a bridging fault inside the layout of a single gate only. This case is a special case of the previous one. It is possible, though rare, that there exists an interconnection component in the critical area. In such case, the interconnection component would be much less than the logic component of critical area and can be ignored.

3. *Different gates.* This category includes fault types in which the two nodes of the fault are shorted in the interconnection part only. ROWEN ignores these types of faults since ROWEN cannot minimize interconnection critical area.

8.2.2 Fault Processing

After reading a fault from the fault list, ROWEN tries to minimize the fault index of the fault based on the above classification. The first category, multiple same-gate faults, is the most general

and complex case, while the other two categories can be considered as simpler special cases. The critical area for a fault that belongs to the multiple same-gate category consists of both logic and interconnection components. The following equation describes the critical area A of a bridging fault.

$$A = \sum A_{\text{logic}} + \sum A_{\text{interconnection}} \tag{8.1}$$

This equation is a general equation that applies to all bridging faults. Unfortunately, ROWEN can only minimize the logic part of the critical area. To minimize the critical area of a fault from this category, ROWEN creates a set of normal-layout gates that are related to the fault, i.e. both of the fault nodes are connected to each gate. Under fault index minimize mode, ROWEN converts every related gate to defect tolerant versions if the defect tolerant version reduces the critical area of the considered fault. Under fault index trim mode, ROWEN sorts the list based on the critical area contribution. ROWEN calculates the new fault index if the first gate is implemented using the defect-tolerant version. If the new fault index is still higher than the trim value, ROWEN goes to the next gate in the list and so on. If the minimum possible fault index is still higher than the desired level, ROWEN checks if the reduction in the critical area is less or equal to the compromise factor times the original critical area. If this also fails then ROWEN does not convert the related gates to their defect-tolerant version.

ROWEN works best on the single same-gate category. The critical area for a fault in this category consists of a single logic element (see Equation 8.1). The interconnection component of critical area is usually zero or much less than the logic component such that it can be ignored. In essence, the fault is formed by a bridge occurring inside the layout of a single gate. Under the fault index minimize mode, ROWEN converts the related gate to defect tolerant version if the defect tolerant version reduces the critical area of the fault. Under the fault index trim mode, ROWEN checks if switching the layout of this gate to defect-tolerant will reduce the fault index to the desired level.

ROWEN cannot decrease the critical areas of the faults in the different gates category since ROWEN can only reduce the logic component of Equation 8.1. Faults of this type have critical area with only an interconnection component. These types of faults must be handled during the placement and routing stage. See Section 8.4 for a discussion about these faults and the relationship between the row and the channel enhancer.

8.3 An Experiment

This section presents a small experiment that illustrates the function of ROWEN. Three of the large ISCAS 1985 [13] circuits were constructed using standard OASIS. Then, using ROWEN, a second layout was generated for every circuit. ROWEN was used in the fault minimize mode targeting the top 100 faults. This experiment is part of the extensive experiment described in Chapter 9. The reduction in total fault index for c3540 and c7552 was more than 30 percent. Unfortunately, c5315 received a marginal reduction in fault index. Careful examination of these circuits reveals that the total fault index of c5315 has a high interconnection component while the total fault index of c3540 and c7552 have high logic components. Since the row enhancer minimizes only the logic portion of fault index, the row enhancer is not able to minimize the total fault index of c5315. This case indicates that some circuits require more than the row enhancement.

Table 8.1: Fault Index Reduction Using ROWEN

Circuit	**OASIS**	**ROWEN**	**Reduction %**
c3540	1.696×10^{12}	1.050×10^{12}	38.12
c5315	7.445×10^{11}	7.196×10^{11}	3.35
c7552	1.941×10^{20}	1.308×10^{20}	32.60

8.4 Relation Between the Row Enhancer and the Channel Enhancer

This section discusses the similarities and differences between the row enhancer YORCC and the channel enhancer ROWEN. While in essence they share a common goal they actually complement one another by concentrating on different parts of the circuit. Unfortunately, there also exist some areas that none of these tools enhance. This section also addresses these unenhanced areas and the need for other tools.

The row enhancer ROWEN and the channel enhancer YORCC share the same goal, reducing the total fault index. Also, both try to target the faults with high fault indices. However, there are several differences between ROWEN and YORCC. ROWEN is an analysis tool that is outside

the layout construction process while the channel enhancing part of YORCC is integrated with the channel router. Although both reduce the critical areas associated with faults, ROWEN reduces the logic component, while YORCC reduces the interconnection component. Moreover, ROWEN has access to the fault list for the whole circuit while YORCC calculates the potential faults inside the channel only. Furthermore, ROWEN utilizes feedback information while YORCC does not. All this makes ROWEN more "mature" compared to YORCC. Future versions of YORCC should utilize the bridging fault information for the whole circuit to accomplish better results.

ROWEN and YORCC also differ at handling different fault types. As discussed in Section 8.2.2, the row enhancer works best on the single same-gate faults where the interconnection component of the critical area is zero or minimal and the logic component can be reduced easily by switching the layout of a single gate. The row enhancer partially reduces critical area for multiple same-gate faults, but produces no improvement for the different-gates category. The channel enhancer deals with the wire-to-wire faults regardless of the type of fault-related gates. Hence, the channel enhancer deals with the following fault types.

1. Wire to Wire—Same Gate—Input to Input.
2. Wire to Wire—Same Gate—Input to Output.
3. Wire to Wire—Different Gates.

Notice that the first two types are considered by both the row enhancer and the channel enhancer.

It is important to note that while total critical area may be mostly due to interconnection, the fault index can be mostly due to logic. In such a case, where the fault index is dominated by logic, the row enhancer significantly reduces the total fault index for the circuit, as it does for c3540 and c7552. However, there are circuits where a large portion of the total fault index is due to interconnection, like c5315. In such a case, the row enhancer can only reduce the total fault index slightly and the channel enhancer can play a larger role in reducing the total fault index for the circuit.

Unfortunately, the following faults are not considered by either the row enhancer or the channel enhancer.

1. Wire to Internal Node—Different Gates.

2. Internal Node to Internal Node—Different Gates.

Both of these two fault types have only an interconnection component. However, they are not considered by the channel enhancer because it can only handle gate-level faults while these two types are switch-level. Minimization of the critical area of these faults has to handled by the placement algorithm. So, there is a need to design a new type of placement tool that tries to place the two nodes of these two types of faults away from each other if placing the two nodes adjacent to one another creates a fault with a large fault index. Such a placement tool should also consider stuck-at-0 and stuck-at-1 faults by placing a wire involved in a stuck-at fault with high interconnection critical area away from V_{DD} (GND). Experiments show that many stuck-at faults have high interconnection components of critical area that can only be minimized during the placement stage. Moreover, the critical area occuring due to placement is predominantly due to stuck-at faults. Wire to internal node—different gates and internal node to internal node—different gates fault types contributed insignificant amounts of critical area, hence, a placement tool should only consider stuck-at faults.

Table 8.2 compares the row enhancer to the channel enhancer with respect to minimizing critical area for different fault types.

To minimize the fault index, both ROWEN and YORCC minimize only the critical area. This suggests the need for another new tool that tries to increase the bridging controllability of bridging faults. Since the bridging controllability is a function of the controllabilities of the two bridged nodes, the new tool must trace every considered fault to its related nodes and then compile a list of gate-level nodes that should have their controllabilities increased. The logic mapping program should read this list to produce another gate-level implementation for the circuit. The new tool could increase controllabilities of gate-level nodes associated with high index faults.

A common case that requires a reimplementation of the gate-level netlist to reduce the fault index is a high fan-out node. It is common for high fan-out nodes to produce stuck-at-0 and stuck-at-1 faults with a high sensitive area. Also, these types of faults usually have high interconnection component of critical area. Since the interconnection portion of these faults is not enhanced by either the row enhancer or the channel enhancer, it is better to avoid creating high fan-out nodes in the gate level implementation unless the node has a high controllability.

Figure 8.3 illustrates a potentially more powerful, and also much more complex, PDFT design

Table 8.2: Enhancement of Different Fault Types

Fault Classification Number	**Fault Type**	**Row Enhancer**	**Channel Enhancer**
1.1	Wire Stuck-At	Yes	No
1.2.1.1	Wire to Wire—Same Gate—Input to Input	Yes	Yes
1.2.1.2	Wire to Wire—Same Gate—Input to Output	Yes	Yes
1.2.2	Wire to Wire–Different Gates	Yes	Yes
2.1	Internal Node Stuck-At	Yes	No
2.2	Transistor Stuck-On	Yes	No
2.3.1.1	Wire to Internal Node—Same Gate—Input to Node	Yes	No
2.3.1.2	Wire to Internal Node—Same Gate—Output to Node	Yes	No
2.3.2	Wire to Internal Node—Different Gates	No	No
2.4.1	Internal Node to Internal Node—Same Gate	Yes	No
2.4.2	Internal Node to Internal Node—Different Gates	No	No

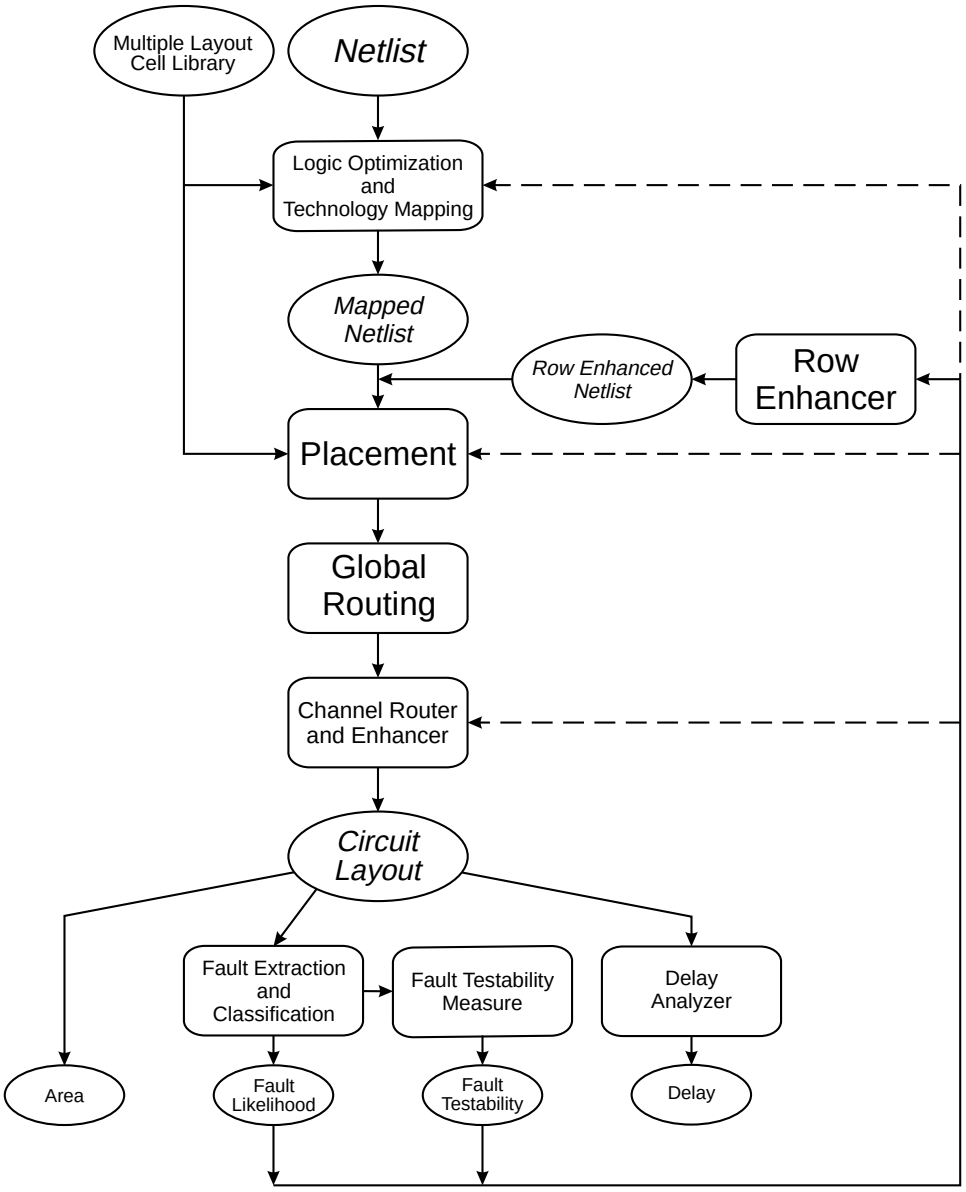

Figure 8.3: Potentially improved PDFT design process.

process that includes a placement tool, a gate-level controllability enhancement tool and a row enhancer that all utilize feedback information. More discussion about the PDFT design process in Figure 8.3 is beyond the scope of this work and is kept for future work.

8.5 Summary

The row enhancer program, ROWEN, is the feedback element in the PDFT design process. ROWEN reads the list of bridging faults and tries to minimize the fault indices of the faults with the highest fault indices. ROWEN minimizes the fault indices by minimizing the critical area associated with the fault. The critical area of a bridging fault occurs inside one or more logic gates or are due to placement and routing. The first part is called logic critical area and the second is called interconnection critical area. ROWEN minimizes the logic part of critical area by replacing the fault-related gates with their defect-tolerant counterparts if this will reduce critical area. In most cases the defect tolerant version minimizes or totally eliminates the critical area located inside the normal version. ROWEN produces a list of gates that should use the defect-tolerant version in the next circuit layout. Row enhancement reduces the total fault indices by more than 30 percent in some cases.

Some faults have logic critical area only, others have interconnection critical area only and some faults have both. While ROWEN minimizes the logic part of critical area for all faults with logic critical area, the channel enhancer, YORCC, minimizes the interconnection critical area for three fault types only. This leaves some faults that cannot be enhanced, totally or partially, by either the row enhancer or the channel enhancer. This suggests a need for a new placement tool that can reduce the interconnection critical area for fault types YORCC cannot enhance. Fault indices can also be decreased by increasing the controllabilities for gate-level nodes. Hence, a new mapping tool that increases the controllability for circuit nodes associated with high index faults is also desirable.

Chapter 9

Experimental Results

This chapter presents an experiment that evaluates the effectiveness and demonstrates the capabilities of the PDFT design process presented in Chapter 4. Section 9.1 explains the goals of the experiment and how the experiment was conducted. Section 9.2 discusses the results of the experiment. Section 9.3 discusses the success of the new design process in achieving the goals of PDFT. Finally, Section 9.5 provides a summary.

9.1 Design Process Evaluation Method

This section describes the process to evaluate the effectiveness of the new PDFT design process presented in Section 4.2. The evaluation process has the following goals.

1. Demonstrate that the proposed design procedure works. All tools, whether developed in this work or in previous work, should work together seamlessly.

2. Test how successful the new PDFT design process is in achieving the goals of PDFT. Several runs of the design procedure should be conducted to provide enough data to judge the PDFT design process.

3. Generate different points in the design space to demonstrate the capabilities of the new design process.

To accomplish these goals, an extensive experiment is conducted. This experiment makes use of all the tools discussed in previous chapters. Section 9.1.1 presents the experiment flow chart.

Five layouts for each of three of the four largest ISCAS 1985 circuits [13] are generated during the experiment. Section 9.1.2 explains the layout generation process. Section 9.1.3 discusses how the experimental data is used to evaluate the new PDFT design process.

9.1.1 The Experiment Flow Chart

The flow chart of the experiment is shown in Figure 9.1. Figure 9.1, similar to Figure 4.1, is an implementation of the PDFT design process shown in Figure 3.1. We reiterate here what was mentioned in Section 4.2 and give more details about the implementation.

The netlist is the circuit netlist file for any of the circuits used in this experiment. MCMAP [65] is mapping program that maps the netlist to cells from the multi-layout cell library. VPNR [65] does placement and global routing while YORCC does channel routing and enhancement. Several programs examine the layout to measure certain parameters. BODEM [9] calculates fault coverage, CARAFE [39] extracts possible bridging faults and finds the likelihood of each fault, BRICON calculates the testability of each fault and CRITICAL [65] calculates critical path delay. ROWEN reads the fault information generated by CARAFE and BRICON and then generates a new netlist for the same circuit. The new netlist is equivalent to the previous netlist at the gate level, but certain gates are realized with a defect-tolerant layout as discussed in Section 7.2. The new netlist is provided to VPNR to start the generation of a new layout for the circuit.

9.1.2 Layout Generation

Three of the four largest ISCAS 1985 circuits [13], c7552, c5315 and c3540, are used for the experiment. c6822 is not used in the experiment for the reasons discussed in Section 5.5. The experiment starts with a circuit netlist. The logic mapping technology program MCMAP [50] generates a netlist using the multiple-layout cell library described in Chapter 7. The generated netlist minimizes the area by minimizing the number of transistors. This netlist applies to all versions of the circuit since different cell layouts share the same switch-level diagram. The layout synthesis tool OASIS [65] produces the first layout. However, instead of using the standard channel router MCROUTE [51], YORCC, the channel router described in Chapter 6, is used. After the first layout is generated, the evaluation of the layout using CARP [9] and CARAFE generates the list of bridging faults that includes the critical area for each fault. BRICON calculates the bridging controllability for each

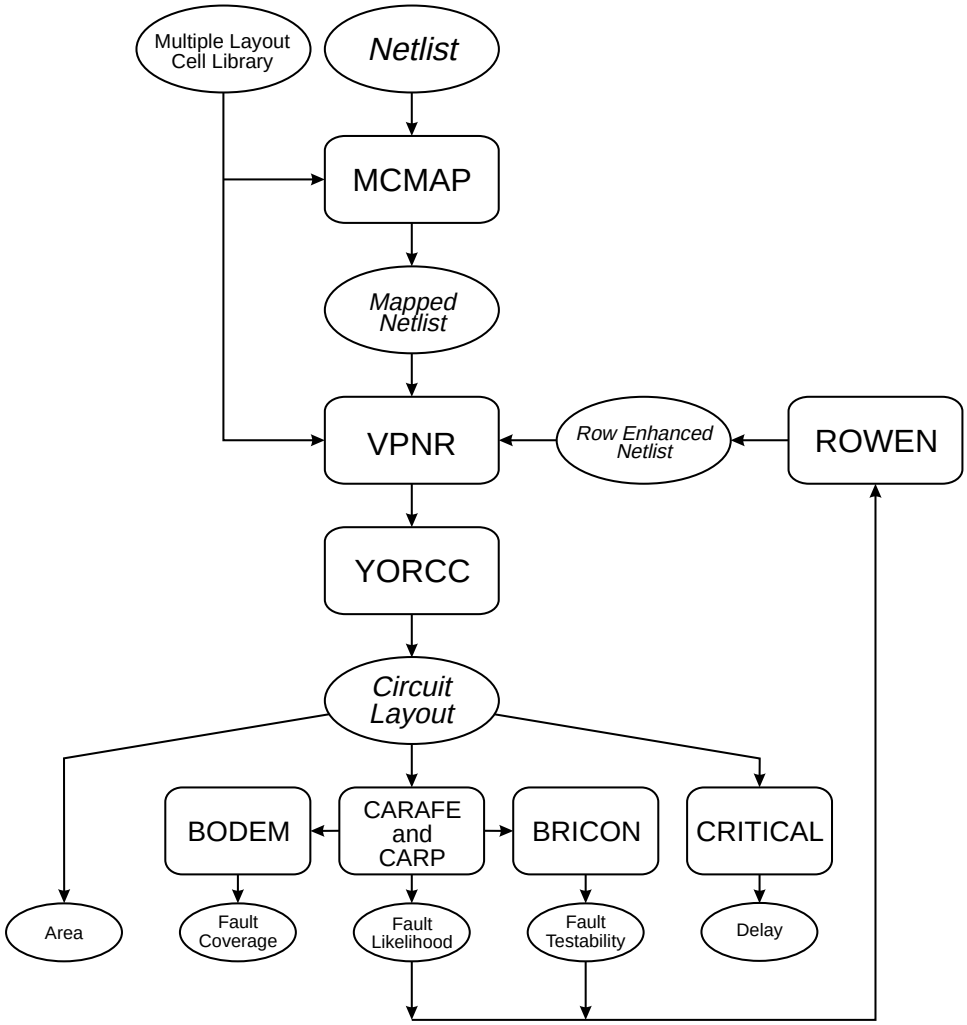

Figure 9.1: PDFT evaluation experiment.

fault and CRITICAL [65] finds the critical path delay for the layout.

The generation of the second layout starts with the ROWEN, the row enhancer. ROWEN minimizes fault indices starting with the fault with the highest fault index. ROWEN finds all gates related to every fault examined and checks if using the defect-tolerant cell will minimize the critical area of the fault. If critical area for a certain fault cannot be reduced, ROWEN does not count it and continues to the next fault in the fault list until it has successfully minimized 100 faults. ROWEN produces a list of the gates that should use defect-tolerant cells in the next layout. A script file generates the new netlist. OASIS reads this new netlist and produces the new layout. The third through fifth layouts are generated in the same manner.

9.1.3 Evaluation of the Design Process

The next step is to examine the layouts that are generated. Layouts are evaluated by measuring area, delay, testability and fault coverage. We consider testability to determine if the total fault index is reduced. The change in area and delay indicate the "cost" of achieving improved testability. Weighted fault coverage is also examined to determine how the design process affects fault coverage.

9.2 Layout Evaluation

The experiment generates five layouts for each circuit. Every layout utilizes information about potential bridging faults from the previous layout. The information utilized consists of the probability of occurrence and the testability of each fault. The probability of occurrence of a bridging fault is represented by its critical area and the testability of a bridging fault is represented by its bridging controllability. These two parameters are combined to form the fault index. The goal of the PDFT design process is to minimize the total fault index for the circuit. Since minimization of fault indices is achieved by minimizing the critical area, the total critical area is expected to decrease also. Minimizing the critical area of a bridging fault is achieved by replacing the fault-related gates with their defect-tolerant counterparts. Thus, it is expected that there will be some increase in delay and area. The following examines the change in layout parameters as layouts are generated by the PDFT design process. Also, these layouts are compared to a layout generated using the standard configuration of OASIS which uses MCROUTE [51] for channel routing.

9.2.1 Defect-Tolerant Cells

The first layout generated consists of only normal versions of cells, while subsequent layouts contain some defect-tolerant versions in addition to the normal versions. Table 9.1 indicates the number of defect-tolerant cells in every circuit layout and Table 9.2 indicates the percentage of defect-tolerant cells in every circuit layout.

Table 9.1: Number of Defect-Tolerant Cells

Circuit	**Number of Gates**	**Standard OASIS**	**Design Iteration**				
			1	**2**	**3**	**4**	**5**
c3540	1229	0	0	180	354	537	690
c5315	2111	0	0	308	559	842	1005
c7552	3186	0	0	57	353	532	700

Table 9.2: Percentage of Defect-Tolerant Cells

Circuit	**Design Iteration**				
	1	**2**	**3**	**4**	**5**
c3540	0	14.65	28.80	43.69	56.14
c5315	0	14.59	26.48	39.89	47.61
c7552	0	1.79	11.08	16.70	21.97

9.2.2 Area

As expected, there is an increase in the area as we go through the design process as shown in Table 9.3 and Table 9.4. However, the increase is small. Also, the percentage of increase is less for larger circuits. Note that in some cases the layout is even smaller than the previously generated circuit. This is because the routing algorithm uses heuristics which can depend on specific circuit features.

The area of the standard OASIS layout is much smaller because MCROUTE, the channel router in OASIS, uses a compaction post-processor [51].

Table 9.3: Total Area (λ^2)

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	6160104	8097952	7999296	9317600	9244584	9502584
c5315	14350000	23327360	21374976	21812088	21367416	24028344
c7552	18991984	27434000	28324896	29762496	28727776	26996704

Table 9.4: Percentage Change in Total Area

Circuit	Design Iteration				
	1	2	3	4	5
c3540	0	-1.22	15.06	14.16	17.35
c5315	0	-8.37	-6.50	-8.40	3.00
c7552	0	3.25	8.49	4.72	-1.59

9.2.3 Critical Path Delay

Table 9.5 lists the rise time and fall time for the critical path in the three benchmark circuits. There is hardly any change in the delay for c3540 and c5315. Circuit c7552 actually has a minor decrease in the delay. The reason for this minor decrease in the delay is that while most of the gates in the critical path use normal layouts, several of defect-tolerant gates are of type **oi2** (two-input NOR) which has lower delay in the defect-tolerant version (see Table 7.3).

9.2.4 Testability

One of the main goals of the new PDFT design process is to decrease the total fault index of a layout. Table 9.6 and Table 9.7 demonstrate that this goal was successfully achieved. The design process decreased the total fault index significantly, even though relatively few faults were targeted. This demonstrates that it is sufficient to consider only a few faults with the highest fault indices. The minimization of the fault indices was achieved by decreasing the critical area for the faults with the highest fault indices. This caused the total critical area of the layout to decrease as shown in Table 9.8. Another important benefit to the decrease in the critical area is better yield.

Table 9.5: Critical Path Delay (nanoseconds)

Circuit	**Standard OASIS**		**Design Iteration**									
			1		**2**		**3**		**4**		**5**	
	rise	fall	rise	fall	rise	fall	rise	fall	rise	fall	rise	fall
c3540	42.60	42.27	42.60	42.27	42.53	42.29	42.70	42.73	42.81	42.58	42.85	42.82
c5315	32.24	32.25	32.24	32.25	32.63	32.63	32.62	32.63	32.75	32.76	32.81	32.81
c7552	69.63	69.61	69.63	69.61	69.63	69.61	66.57	66.55	66.58	66.55	66.45	66.46

Table 9.6: Total Fault Index

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	1.736×10^{12}	1.696×10^{12}	1.050×10^{12}	1.032×10^{12}	1.012×10^{12}	9.974×10^{11}
c5315	8.643×10^{11}	7.445×10^{11}	7.196×10^{11}	6.463×10^{11}	6.128×10^{11}	6.089×10^{11}
c7552	1.995×10^{20}	1.941×10^{20}	1.308×10^{20}	1.404×10^{20}	1.337×10^{20}	1.533×10^{20}

Table 9.7: Percentage Reduction in Total Fault Index

Circuit	**Design Iteration**				
	1	**2**	**3**	**4**	**5**
c3540	0	38.12	39.15	40.34	41.20
c5315	0	3.35	13.19	17.68	18.21
c7552	0	32.60	27.67	31.11	21.04

Table 9.8: Total Critical Area ($10^9 \times centimicron^2$)

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	115.54	98.08	90.61	91.16	83.49	82.21
c5315	244.19	199.50	183.51	182.27	172.05	176.82
c7552	340.42	273.60	268.46	264.86	253.66	244.00

Table 9.9: Percentage Reduction in Total Critical Area

Circuit	Design Iteration				
	1	2	3	4	5
c3540	0	7.62	7.05	14.87	16.17
c5315	0	8.01	8.63	13.76	11.37
c7552	0	1.88	3.20	7.29	10.82

9.2.5 Weighted Fault Coverage

Fault coverage is the percentage of possible faults that are tested using a set of test vectors [9]. When bridging faults are considered, there are many bridging faults, some with a high probability of occurrence and others with extremely low probability of occurrence. A more appropriate way to express fault coverage is weighted fault coverage which is a more accurate indication of test set quality [9]. Weighted fault coverage is defined as

$$WFC = \sum_{i=1}^{n} d_i w_i$$

where n is the number of possible bridging faults, d_i is 1 if the ith fault is tested by the test vectors and 0 if not, and w_i is the probability of occurrence for the ith fault. When critical area is used as a measure of probability of occurrence, weighted fault coverage can be written as

$$WFC = \frac{\sum_{i=1}^{n} d_i a_i}{\sum_{i=1}^{n} a_i}$$

where a_i is the critical area of the ith fault. Hence, the weighted fault coverage is the sum of the critical areas of tested faults divided by the total critical area. Note that d_i is a function of the set of test vectors used.

Tables 9.10 and 9.11 list the fault coverages for all generated layouts using 5 and 15 random test vectors, respectively. Table 9.12 shows the fault coverages attained using deterministic sets of test vectors generated by BODEM. For every layout, BODEM generates a set of test vectors that provides high fault coverage. Table 9.13 lists the number of vectors BODEM generated for each layout.

Table 9.10: Weighted Fault Coverage Using 5 Random Test Vectors

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	87.14	86.28	86.06	86.49	86.63	87.24
c5315	92.70	91.97	92.28	92.55	92.68	92.66
c7552	92.91	92.18	92.45	92.45	92.34	92.35

Table 9.11: Weighted Fault Coverage Using 15 Random Test Vectors

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	95.41	94.81	95.09	95.47	95.43	95.79
c5315	98.61	98.39	98.51	98.65	98.59	98.68
c7552	98.28	97.93	98.15	98.21	98.17	98.17

Table 9.12: Weighted Fault Coverage Using BODEM Test Vectors

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	99.60	99.55	99.58	99.49	99.55	99.59
c5315	99.90	99.91	99.93	99.94	99.90	99.93
c7552	99.87	99.85	99.92	99.89	99.86	99.84

By looking at Table 9.10 and Table 9.11, we notice a drop in weighted fault coverage for the first layout compared to the standard OASIS layout. Then, the weighted fault coverage increases as we go from iteration 1 to 5. This shows that the new PDFT design process increases the weighted fault coverage from iteration to iteration.

An explanation is needed regarding the lower weighted fault coverage for iteration 1 versus the

Table 9.13: Number of BODEM Test Vectors

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	87	73	81	71	76	71
c5315	57	62	58	59	53	58
c7552	97	100	100	95	95	97

standard OASIS layout. Unlike MCROUTE used in the standard OASIS layout, YORCC tries to decrease the total critical area for the layout. Although YORCC tries to reduce the critical area of the faults with high indices within the channel, YORCC is not aware of the faults that have high fault indices throughout the whole layout. In other words, YORCC considers only the channel rather than the full layout. Consequently, the critical area of faults with high indices is unaltered by YORCC. This leads to the drop in the fault coverage because the total critical area decreases while the critical area of the aborted faults remains the same or is reduced only slightly. The weighted fault coverage starts to improve only when ROWEN is involved because, unlike YORCC, ROWEN is aware of the high-index faults throughout the whole layout.

The drop in fault coverage is misleading since it suggests that the new layout is worse than the previous one. In fact, the number of aborted faults in Tables 9.10, 9.11 and 9.12 is virtually the same. Moreover, the new layout has less critical area and, hence, better yield. A price is paid, however, in using a larger area.

The above discussion suggests the need for a closer look at the effect of the PDFT design process on fault coverage. We examine fault coverage because it gives us some idea about the quality of a certain set of test vectors. In this work no new method of generating test vectors is considered. Instead, this work considers the testability and the probability of occurrence. In particular, how to reduce the probability of occurrence for "bad" faults. Bad faults are the faults that are both hard to test and likely to occur. We expect the critical area for the untested faults to be reduced as we go through the iterations. Tables 9.14, 9.15 and 9.16 show that the number of aborted faults is roughly the same throughout the iterations. Yet, Tables 9.17, 9.18 and 9.19 show that the critical area for the aborted faults decreases as we go through the iterations. Moreover, the rate of decrease for the critical area for aborted faults is greater than that for all faults as given in Table 9.8. We conclude

that the design process, which reduces the critical area for many faults, reduces the critical area for hard to test faults more than for typical faults.

Table 9.14: Number of Aborted Faults using 5 Test Vectors

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	3129	3061	3103	3038	3153	3105
c5315	4020	3932	3801	3843	3621	3756
c7552	5696	5595	5459	5473	5396	5413

Table 9.15: Number of Aborted Faults using 15 Test Vectors

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	1078	1074	1062	993	1066	990
c5315	712	706	663	644	646	643
c7552	1210	1201	1188	1172	1164	1194

Table 9.16: Number of Aborted Faults using BODEM Test Vectors

Circuit	Standard OASIS	Design Iteration				
		1	2	3	4	5
c3540	103	103	98	92	99	96
c5315	51	48	46	39	44	36
c7552	81	79	68	70	66	97

9.2.6 Execution Time

Execution time for the PDFT process is divided into two parts, the layout generation time and the feedback time. Layout generation is the time required to generate the layout starting with a gate-level netlist. The feedback time is the time required to extract the faults and process the faults to find the gates that will have defect-tolerant layouts in the next iteration. Feedback execution time is the time needed by CARP, BRICON, CARAFE, and ROWEN. Table 9.20 shows both times and compares the layout generation time of PDFT with standard OASIS. Other iterations require

Table 9.17: Critical Area of Faults Aborted by BODEM using 5 Random Vectors ($10^{10} \times$ centimicron^2)

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	1.486	1.346	1.263	1.232	1.116	1.049
c5315	1.784	1.602	1.416	1.358	1.259	1.298
c7552	2.412	2.141	2.027	1.999	1.944	1.866

Table 9.18: Critical Area of Faults Aborted by BODEM using 15 Random Vectors ($10^9 \times$ centimicron^2)

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	5.298	5.093	4.451	4.132	3.816	3.463
c5315	3.395	3.222	2.740	2.468	2.431	2.333
c7552	5.840	5.657	4.965	4.742	4.647	4.464

Table 9.19: Critical Area of Faults Aborted by BODEM using BODEM Vectors ($10^8 \times$centimicron^2)

Circuit	**Standard OASIS**	**Design Iteration**				
		1	**2**	**3**	**4**	**5**
c3540	4.648	4.386	3.766	4.657	3.719	3.368
c5315	2.328	1.741	1.358	1.169	1.688	1.305
c7552	4.422	4.147	2.109	2.968	3.664	3.995

similar execution time. The execution times recorded in Table 9.20 were attained by running all the programs on an IBM RS/6000 SP2 machine.

Table 9.20: Execution Time (hour:min:sec)

Circuit	**Layout Generation**		**Feedback Loop**
	Standard OASIS	**1st Design Iteration**	
c3540	0:07:18	0:37:16	0:18:06
c5315	0:32:47	3:27:11	1:32:23
c7552	0:55:12	3:59:29	2:57:39

Layout generation execution time of the new design process is higher than that of standard OASIS. This is due to the longer execution time needed by YORCC compared to MCROUTE. Careful examination of YORCC behavior reveals that the extra time needed by YORCC is due to the current implementation of the net merging channel routing algorithm and not due to the cyclic resolving algorithm, new track assignment scheme or via shifting post-processor. It should be emphasized that the net merging algorithm is not a element of the PDFT design process. Other, more efficient routing algorithms can be investigated for implementing a faster channel enhancer. The feedback time, which is consumed mainly by CARP to combine equivalent faults, can be reduced by using more efficient search algorithms.

9.3 Design Methodology Evaluation

The new design process is successful in achieving the goals of PDFT, i.e. providing layouts with fewer faults and minimizing the probability of occurrence of hard-to-test faults. All the tools were successfully connected together and functioned correctly. The price paid was increased area and execution time. In general, the new design process does not increase the delay. The weakest part of the new design process is the channel router and enhancer, YORCC. More work is needed to design a channel router and a channel enhancer that further reduce area and enhance testability.

9.4 Choices in the Design Space

The new PDFT design process generates a different layout for each iteration. Every layout generated implements the same logic function, yet has different characteristics in terms of area, yield, testability, delay and fault coverage. These different layouts are different design alternatives available to the circuit designer. The designer can choose from the several different points in the design space. As an example, Figures 9.2, 9.3, 9.4 and 9.5 show the change in critical area, fault index, fault coverage and delay, respectively, relative to the change in area for c5315.

9.5 Summary

This chapter presented an experiment that tests the effectiveness of the new PDFT design process and demonstrates its capabilities. All of the tools, whether developed in this work or in previous work, are connected together to form the PDFT design process.

The experiments starts with a netlist of a circuit, then several layouts are generated for every circuit. Every layout, except for the first one, utilizes information from the previous layout to minimize the probability of occurrence for faults with high fault indices. Layouts are generated for three of the large circuits from the ISCAS 1985 benchmark circuits.

The following is a summary of parameters that were monitored during the experiment and how they are affected.

1. *Area.* There is virtually no increase in area as we go through the PDFT design process. However, the layouts are larger in area than layouts produced by OASIS using its built-in channel router MCROUTE. The use of an improved router should decrease the initial increase in area.

2. *Delay.* Delay is not significantly increased.

3. *Fault Index.* The new design process successfully minimizes the total fault index of the layout. The decrease in the fault index is due mainly to the row enhancer, ROWEN, with marginal improvement due to the channel enhancer, YORCC.

4. *Weighted Fault Coverage.* There is no improvement in the weighted fault coverage when using a set of deterministic test vectors since the test vectors are so effective. However, there is an

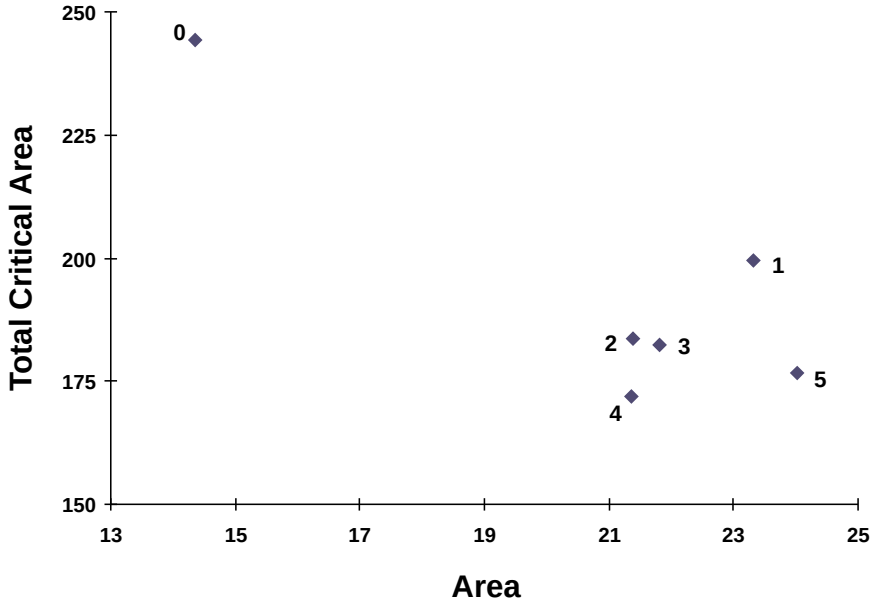

Figure 9.2: Total critical area versus area for c5315.

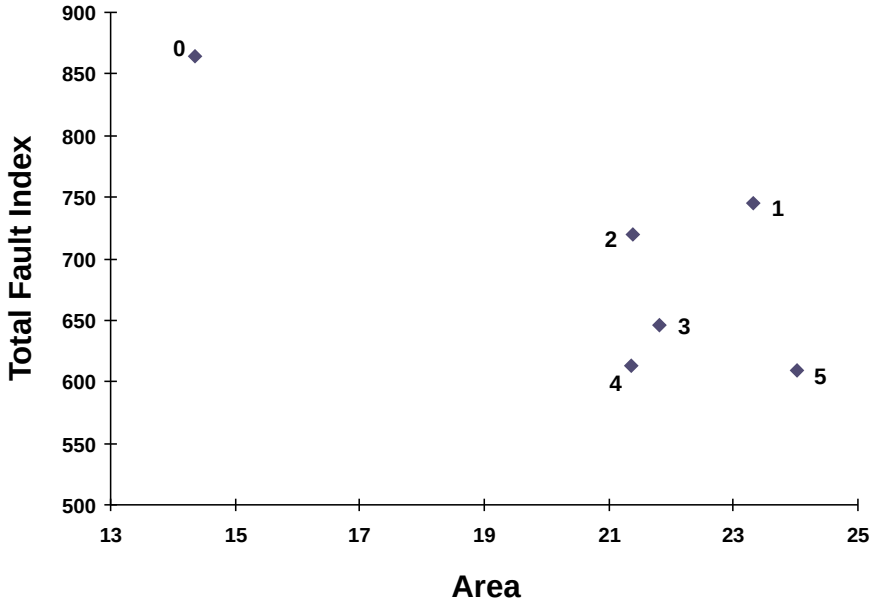

Figure 9.3: Total fault index versus area for c5315.

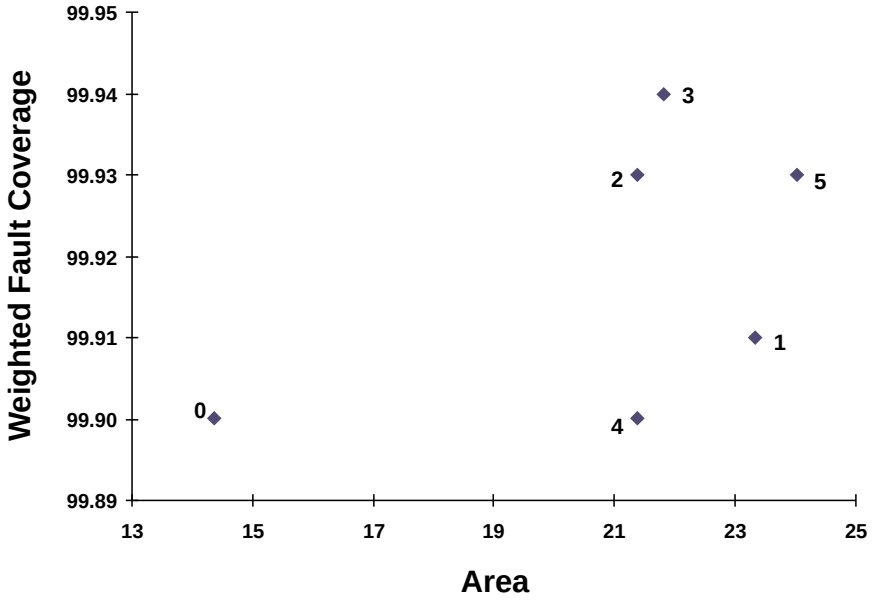

Figure 9.4: Weighted fault coverage versus area for c5315.

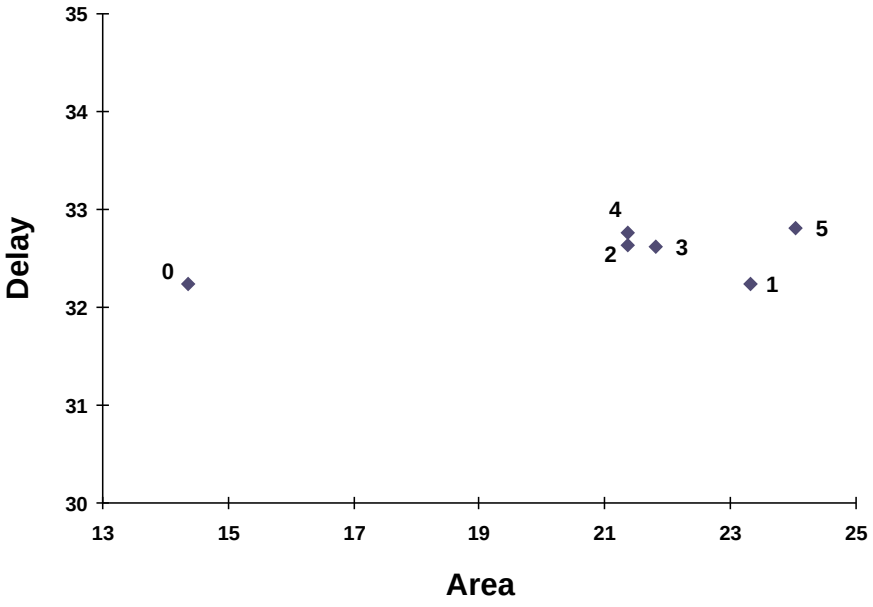

Figure 9.5: Delay versus area for c5315.

improvement when a less effective set of random test vectors is used. Moreover, the critical area for hard-to-test faults decreases at a higher rate than for average faults. Targeting the hard-to-test faults more than the average fault is a strength of the new PDFT design process.

5. *Execution Time.* The execution time consists of two parts: the layout generation part and the feedback time. The new PDFT design process requires more execution time due to the current implementation of the channel router and not due to the channel enhancing features.

These results show that the new PDFT design process successfully achieves two of the PDFT goals, providing layouts with fewer faults and minimizing the probability of occurrence of hard-to-test faults. There is still need for improvement, especially in designing a better channel router that minimizes area while minimizing the critical area. The third goal of PDFT, that is making difficult-to-test faults easier to detect, is not achievable by the PDFT design process since increasing the detectability of a fault requires increasing the controllabilities of the fault-related nodes. The controllabilities of circuit nodes can only be increased by altering the gate-level implementation of the circuit which is beyond the scope of this work. See Section 10.3 which suggests developing a new technology mapping tool that enhances controllability for fault-related nodes.

Chapter 10

Conclusions

10.1 Summary of Work

This work contributes to the establishment and understanding of PDFT in several ways. This work formally defines PDFT, provides a detailed discussion of the goals of PDFT and sheds some light on how these goals can be achieved. To provide a practical aspect to PDFT, this work considers bridging faults in CMOS standard cell circuits and adapts I_{DDQ} testing. PDFT is divided into two parts: logic and interconnection. Logic PDFT deals with faults occurring inside the logic elements like standard cells and logic arrays. Interconnection PDFT deals with faults occurring in the routing part of the circuit or due to placement. Applying this concept to bridging faults allows the critical area associated with each bridging fault to be divided into logic and interconnection components. The logic component of critical area is the sum of the critical area of all bridges located inside the fault-related cells. The interconnection component of critical area is the sum of the critical area of all bridges occurring in the routing part of the circuit or due to placement. The division of bridging fault critical area into a logic component and an interconnection component divides bridging faults into three groups: faults with only logic critical area, faults with only interconnection critical area and faults with both logic and interconnection critical area. This arrangement suggests developing two different tools to minimize the critical area logic component and critical area interconnection component of bridging faults, respectively.

This work proposes a new PDFT design process that assumes standard cell design. Instead of designing the circuit and then generating tests, the proposed PDFT process incorporates the

evaluation of the realistic faults into the design process. Using feedback information from a previously generated layout, the PDFT design process generates a more testable layout. The feedback information is the list of potential bridging faults in the circuit. While generating the new layout, the PDFT design process eliminates or reduces the probability of occurrence for faults with high fault indices. Fault index is defined as the testability of the fault divided by the probability of occurrence. Faults with the highest fault indices are considered the worst faults because they have a high probability of occurrence and are hard to test.

An important missing block in previous PDFT work is a testability measure for bridging faults. This work provides a testability measure for bridging faults by developing a methodology to extend current stuck-at fault testability measures to bridging faults. Since I_{DDQ} testing grants observability, the bridging fault testability measure is based only on controllability. Three of the traditional controllability measures defined for stuck-at faults are extended to bridging faults: CAMELOT, SCOAP and COP. Comparing the three controllability methods discussed, COP is preferred over SCOAP and CAMELOT since it has a solid mathematical foundation and is easier to implement. The experiments conducted demonstrate that bridging controllability provides a good estimate of detectability, yet requires relatively low execution time. BRICON, a program that calculates bridging controllability based on SCOAP and COP, is presented. BRICON's output gives the probability of occurrence of bridging faults and the controllability of each fault. The probability of occurrence is generated by CARAFE, the bridging fault extractor, and preserved in BRICON's fault list.

The channel enhancer tool is developed to handle the bridging faults in the routing channels of the circuit. The channel enhancer consists of a channel router that supports PDFT and a post-processor. The router uses an extended version of the net merging channel routing algorithm that is extended to handle cyclic cases. A new track assignment scheme that attempts to minimize fault indices for bridging faults between horizontal wires in the channel is presented. A via shifting post-processor attempts to minimize critical area by, fully or partially, moving some wires to a different layer. Unfortunately, the reduction in critical area and fault index is marginal. The limited knowledge that the channel enhancer has about faults in the entire circuit leads to this marginal reduction in fault index. Future version of the channel enhancer should have more knowledge about faults throughout the entire layout.

To eliminate or reduce the critical area of bridging faults inside the cells, a defect-tolerant cell

library is designed. The defect-tolerant cell library is derived from the OASIS standard cell library by expanding the area of the cells and then redesigning the cells to reduce the critical area. Several critical area reduction techniques were developed during the redesign stage. These techniques do not depend on the structure of OASIS cell library and should be applicable to any other cell library. On average, the defect-tolerant layouts have 44 percent less critical area at the expense of a 30 percent increase in area and a minor increase in delay.

The row enhancer program, ROWEN, is the feedback element in the PDFT design process. ROWEN reads the list of bridging faults and tries to minimize the fault indices of the faults with the highest fault indices. ROWEN minimizes the logic part of critical area by replacing the fault related gates with their defect-tolerant counterparts. In most cases, the defect-tolerant version minimizes or totally eliminates the critical area located inside the normal version. ROWEN produces a list of gates that should use the defect tolerant version in the next circuit layout. Building circuits that utilize row enhancement reduces the total fault index by more than 30 percent in some cases.

This work presented an experiment that tests the effectiveness of the new PDFT design process and demonstrates its capabilities. All the tools, whether developed in this work or in previous work, are connected together to form the PDFT design process. The experiment starts with a netlist of a circuit, then several layouts are generated for every circuit. Every layout, except for the first one, utilizes information from the previous layout to minimize the probability of occurrence for faults with high fault indices. Five layouts are generated for three of the large circuits from the ISCAS 1985 benchmark circuits. The new design process successfully minimized the total fault index of the layout with no significant additional area or delay. Although there is no improvement in the weighted fault coverage when using a set of deterministic test vectors since coverage is already high, there is an improvement when a set of random test vectors is used. Moreover, the critical area for the hard-to-test faults decreased at a higher rate than the critical area for the average fault. These results show that the new PDFT design process successfully achieved two of the goals of PDFT, providing layouts with fewer faults and minimizing the probability of occurrence of hard-to-test faults. Yet, there is still need for improvement, especially in designing a better channel router that minimizes area while minimizing the critical area.

10.2 Contributions of this Work

This work produced the following contributions in the area of PDFT and in electronic design automation (EDA).

1. A new PDFT design process that incorporates the evaluation of the realistic faults into the design process was developed. Using feedback information from a previously generated layout, a new layout is generated which eliminates or reduces the probability of occurrence for faults with high fault indices.

2. This work provides new insight into the concept of physical design for testability, includes a formal definition of PDFT, a detailed discussion of the goals of PDFT and a method to achieve these goals. Also, this work shows the relationship between the bridging fault classes and the two divisions of PDFT, logic and interconnection.

3. A new testability measure for bridging faults, denoted as bridging controllability, was developed. Bridging controllability assumes I_{DDQ} testing which is the most effective testing method for bridging faults.

4. The testability figure represented by bridging controllability and the probability of occurrence figure represented by critical area are combined in a new parameter, the fault index. Comparing faults in terms of fault indices enables the automatic comparison of faults and provides a simple way to identify the worst faults.

5. A defect-tolerant cell library that eliminates or reduces the critical area of bridging faults and critical area reduction techniques was developed. These techniques do not depend on the structure of any particular cell library and should be applicable to other cell libraries.

6. A row enhancer tool, ROWEN, that minimizes the fault indices of the faults with the highest fault indices was developed. Building circuits that utilize row enhancement reduces the total fault index by more than 30 percent in some cases. ROWEN minimizes the fault indices of faults with a logic critical area component by minimizing the logic critical area associated with the fault.

7. A channel enhancer tool, YORCC, that minimizes the fault indices of faults with interconnection critical area component was developed. Even though the current implementation of the channel enhancer produces insignificant results, this work contributed in providing a detailed formulation for the channel enhancing process and clarifying how the role of the channel enhancer complements the role of the row enhancer.

8. This work proved the need for better mapping technology that increases the controllability of certain logic nodes. Also, the work suggests the need for a placement enhancement tool that handles faults not being targeted by the row enhancer or channel enhancer.

9. A new algorithm to extend the net merging channel routing algorithm to handle cyclic cases was developed.

10. A new and comprehensive formulation of the track assignment problem based on optimization theory was presented.

10.3 Directions for Future Research

Although this work provided many answers about how to achieve the goals of PDFT, unanswered questions remain. The following are some suggested topics for future research.

1. The interconnection component of bridging faults critical area occurs in the routing part of the circuit or due to placement. The channel enhancer deals only with faults occuring inside the routing part of the circuit. A new type of placement tool that handles bridging faults that occur due to placement is needed. Such a placement tool should try to place the two nodes of a bridging fault due to placement away from each other if the fault has a high fault index. The placement tool should consider stuck-at-0 and stuck-at-1 faults by placing a wire involved in a stuck-at fault with high critical area in the interconnection away from V_{DD} or GND.

2. Despite the extensive formulation of the channel enhancer, the current tool provides insignificant reduction of fault indices. Several new methods should be investigated to develop an effective channel enhancer. For example, the fault index figure used could be obtained from

a previous layout rather than using the local channel information. Other routing algorithms, like simulated annealing [43], could also be investigated.

3. A new technology mapping tool that enhances controllability for certain logic-level nodes that are known to cause high index bridging faults is needed.

4. The relationship between bridging fault classification and critical area and fault index should be studied through the compilation of a large set of data. Such data should be used to find out which types of faults tend to be more problematic than others, then, reflect that in improvements of the channel enhancing and row enhancing tools.

5. Gridless placement and routing tools that allow more freedom in designing defect-tolerant cell libraries could be investigated. Also, channel enhancing techniques should be applied to gridless routers.

REFERENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, New York, NY: Computer Science Press, 1990.

[2] J. M. Acken, "Testing for bridging faults (shorts) in CMOS circuits," *ACM/IEEE Design Automation Conf.*, 1983, pp. 717-718.

[3] V. D. Agrawal and M. R. Mercer, "Testability measures - what do they tell us?" *Int. Test Conf.*, 1982, pp. 391-396.

[4] G. A. Allan and A. J. Walton, "Hierarchical critical area extraction with the EYE tool," *IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Sys.*, 1995, pp. 28-36.

[5] H. Ando, "Testing VLSI with random access scan," *COMPCON*, 1980, pp. 50-52.

[6] K. Baker, "QTAG: A standard for test fixture based I_{DDQ}/I_{SSQ} monitors," *Int. Test Conf.*, 1994, pp. 194-202.

[7] K. Balachandran, S. Bhaskaran, H. Ganesan, and C. Lurisinsap, "A yield enhancing router," *IEEE Int. Symp. Circuits and Systems*, 1991, pp. 1936-1939.

[8] R. G. Bennetts, C. M. Maunder, and G. D. Robinson, "CAMELOT: A computer-aided measure for logic testability," *IEE Proc. E, Computer & Digital Tech.*, vol. 128, no. 5, Sept. 1981, pp. 177-198.

[9] S. W. Bollinger, "Hierarchical test generation for CMOS circuits," Ph.D. Dissertation, Virginia Polytechnic Institute and State University, 1992.

[10] S. W. Bollinger and S. F. Midkiff, "Test generation for I_{DDQ} testing of bridging faults in CMOS circuits," *IEEE Trans. Computer-Aided Design*, vol. 13, no. 11, Nov. 1994, pp. 1413-1418.

[11] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Hingham, MA: Kulwer Academic Publishers, 1984.

[12] F. Brglez, "On testability analysis of combinational networks," *IEEE Int. Symp. Circuits and Systems*, 1984, pp. 221-225.

[13] F. Brglez and H. Fujiwara, "Neural netlist of ten combinational benchmark circuits and a target translator in FORTRAN," presented at the *IEEE Int. Symp. Circuits. Syst.*, 1985.

[14] F. Brglez, P. Pownall, and P. Hum, "Applications of testability analysis: From ATPG to critical path tracing," *IEEE Int. Test Conf.*, 1984, pp. 705-712.

[15] S. Chakravarty and P. Thadikaran, "A study of subset selection algorithms for bridging faults," *Int. Test Conf.*, 1994, pp. 403-412.

[16] S. J. Chandra and J. H. Patel, "Experimental evaluation of testability measures for test generation," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 1, Jan. 1989, pp. 93-97.

[17] V. Chandramouli, R. K. Gulati, R. Dandapani, and D. K. Goel, "Bridging faults and their implication to PLAs," *Int. Test Conf.*, 1990, pp. 852-859.

[18] H. H. Chen and E. S. Kuh, "Glitter: A gridless varaible-width channel router," *IEEE Trans. Computer-Aided Design*, vol. 5, no. 10, Oct. 1986, pp. 459-465.

[19] H. H. Chen, "Breaking cycles and vertical constraints in Deulsch's new and more difficult channel-routing problems," *Midwest Symp. Circuits and Systems*, 1989, pp. 539-542.

[20] G. Chiorboli and A. Ferrari, "Influence of guiding testability measure on number of backtracks of ATPG program," *IEE Proc. E, Computer & Digital Tech.*, vol. 136, no. 4, July 1989, pp. 316-320.

[21] M. Dalpasso, M. Favalli, P. Olivo, and B. Ricco, "Influence of IC synthesis on the random pattern testability of parametric bridging faults," *European Test Conf.*, 1993, pp. 398-407.

[22] M. Desrochers and G. Laporte, "Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constrains," *Oper. Res. Letters*, vol. 10, no. 1, Feb. 1991, pp. 27-36.

[23] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testing," *ACM/IEEE Design Automation Conf.*, 1977, pp. 462-468.

[24] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," *J. Design Automat. Fault-Tolerant Comput.*, vol. 2, no. 2, May 1978, pp. 165-178.

[25] F. J. Ferguson, "Physical design for testability for bridges in CMOS circuits," *IEEE VLSI Test Symp.*, 1993, pp. 290-295.

[26] F. J. Ferguson and T. Larrabee, "Test pattern generation for realistic bridge faults in CMOS ICs," *IEEE Int. Test Conf.*, 1991, pp. 492-499.

[27] F. J. Ferguson and J. P. Shen, "A CMOS fault extractor for inductive fault analysis," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 11, Nov. 1988, pp. 1181-1194.

[28] R. Fritzemeier, H. T. Nagle, and C. F. Hawkins, "Fundamentals of testability - A tutorial," *IEEE Trans. Industrial Electronics*, vol. 36, no. 2, May 1989, pp. 117-128.

[29] S. Funatsu, N. Wakatsuki, and T. Arima, "Test generation systems in Japan," *Design Automation Symp.*, 1975, pp. 114-122.

[30] J. Galiay, Y. Crouzet, and M. Vergniault, "Physical versus logical fault models for MOS LSI circuits: Impact on their testability," *IEEE Trans. Comput.*, vol. 29, no. 6, June 1980, pp. 527-531.

[31] D. W. Gillies and J. W. Liu, "Scheduling tasks with and/or presedence constraints," *SIAM J. Comput.*, vol. 24, no. 4, Aug. 1995, pp. 797-810.

[32] L. H. Goldstein, "Controllability/observability analysis of digital circuits," *IEEE Trans. Circuits and Systems*, vol. CAS-26, no. 9, Sept. 1979, pp. 685-693.

[33] T. Guckert, P. Schani, M. Phillips, M. Seeley, and N. Herr, "Design and process issues for elimination of device failures due to drooping vias," *Int. Symp. for Test and Failure Analysis (ISTFA)*, 1991, pp. 97-101.

[34] D. V. Heinbuch, *CMOS3 Cell Library*, Reading, MA: Addison Wesley, 1988.

[35] E. Isern and J. Figueras, "Analysis of I_{DDQ} detectable bridges in combinational CMOS circuits," *IEEE VLSI Test Symp.*, 1994, pp. 368-373.

[36] M. Jacomet and W. Guggenbuhl, "Layout-dependent fault analysis and test synthesis for CMOS circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 6, June 1993, pp. 888-899.

[37] S. K. Jain and V. D. Agrawal, "STAFAN: an alternative to fault simulation," *ACM/IEEE Design Automation Conf.*, 1984, pp. 18-23.

[38] A. Jee and C. Bazeghi, "Carafe users manual: Alpha.4 release," University of California at Santa Cruz, 1994.

[39] A. Jee and F. J. Ferguson, "Carafe: an inductive fault analysis tool for CMOS VLSI Circuits," *IEEE VLSI Test Symp.*, 1993, pp. 92-98.

[40] R. Johnson, *Elementary Statistics*, Boston, MA: PWS-KENT, 1988.

[41] R. Kapur, K. M. Butler, D. E. Ross, and M. R. Mercer, "On bridging fault controllability and observability and their correlation to detectability," *2nd European Test Conf.*, 1991, pp. 333-339.

[42] A. J. Kessler and A. Ganesan, "Standard cell VLSI design: A tutorial," *IEEE Cir. Dev. Magazine*, vol. 1, no. 1, Jan., 1985 pp. 17-33.

[43] J. Khare, S. Mitra, P. K. Nag, W. Maly, and R. Rutenbar, "Testability-oriented channel routing," *Int. Conf. VLSI Design.*, 1995, pp. 208-213.

[44] S. Kuo, "YOR: A yield-optimizing routing algorithm by minimizing critical areas and vias," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 9, Sept. 1993, pp. 1303-1311.

[45] V. Kulkarni, "Increase fault coverage with I_{DDQ} testing," *Electronic Design*, vol. 42, no. 16, Aug. 1994, pp. 87-88, 90, 92-93.

[46] E. Lawler, J. Lenstra, A. Kan, and D. Shmoys, *The Traveling Salesman Problem*, New York, NY: John Wiley and Sons, 1992.

[47] K. Lee and M. A. Breuer, "Design and test rules for CMOS circuits to facilitate IDDQ testing of bridging faults," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 5, May 1992, pp. 659-670.

[48] M. E. Levitt and J. A. Abraham, "Physical design of testable VLSI: techniques and experiments," *IEEE J. Solid-State Circuits*, vol. 25, no. 2, April 1990, pp. 474-481.

[49] R. Lisanke, G. Kedem, and F. Brglez, "DECAF: Decomposition and factoring for multi-level logic synthesis," Technical Report TR87-15, MCNC, Research Triangle Park, NC, 1987.

[50] R. Lisanke, F. Brglez and G. Kedem, "McMAP: A fast technology mapping procedure for multi-level logic synthesis," *IEEE Int. Conf. Computer Design*, 1988, pp. 252-256.

[51] M. J. Lorenzitti, M. S. Nifong and J. E. Rose, "Channel routing for compaction," *Proc. Int. Workshop on Placement and Routing*, 1988.

[52] R. Luescher and J. S. De Zaldivar, "A high density CMOS process," *IEEE Int. Solid-State Circuits Conf.*, 1985, pp. 260-261.

[53] Y. K. Malaiya and A. P. Jayasumana, "Enhancement of resolution in supply current based testing for large ICs," *IEEE VLSI Test Symp.*, 1991, pp. 291-296.

[54] Y. K. Malaiya, A. P. Jayasumana, and R. Rajsuman, "A detailed examination of bridging faults," *IEEE Int. Conf. Comp. Design*, 1986, pp. 78-81.

[55] Y. K. Malaiya and S. Y. Su, "A new fault model and testing techniques for CMOS devices," *Int. Test Conf.*, 1982, pp. 25-34.

[56] W. Mao and R. K. Gulati, "QUITEST: a methodology for selecting I_{DDQ} test vectors," *J. Electron. Test.*, vol. 3, no. 4, Dec. 1992, pp. 349-357.

[57] R. McGowen and F. J. Ferguson, "Eliminating undetectable shorts between horizontal wires during channel routing," *IEEE VLSI Test Symp.*, 1994, pp. 402-407.

[58] S. F. Midkiff and S. W. Bollinger, "Circuit-level classification and testability analysis for CMOS faults," *IEEE VLSI Test Symp.*, 1991, pp. 193-198.

[59] S. F. Midkiff and S. W. Bollinger, "Classification of bridging faults in CMOS circuits: experimental results and implications for test," *IEEE VLSI Test Symp.*, 1993, pp. 112-115.

[60] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *J. ACM*, vol. 7, no. 4, Oct. 1960, pp. 326-329.

[61] P. K. Nag and W. Maly, "Hierarchical extraction of critical area for shorts in very large ICs," *IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Syst.*, 1995, pp. 19-27.

[62] V. V. Nickel, "VLSI - The inadequacy of the stuck-at fault model," *Int. Test Conf.*, 1980, pp. 378-381.

[63] P. Nigh and W. Maly, "Test generation for current testing," *European Test Conf.*, 1989, pp. 194-200.

[64] J. Novellino, "I_{DDQ} testing finds further faults," *Electronic Design*, vol. 43, no. 18, Sept. 1995, pp. 77-78, 80, 82.

[65] OASIS Manual, version 2.0, MCNC, Research Triangle Park, NC, 1992.

[66] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "The Magic VLSI layout system," *IEEE Design and Test Comp.*, vol. 2, no. 1, Feb. 1985, pp. 19-30.

[67] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.*, vol. 24, no. 6, June 1975, pp. 668-670.

[68] A. Pitaksanonkul, S. Thanawastien, C. Lursinsap, and J. A. Gandhi, "DTR: A defect-tolerant routing algorithm," *ACM/IEEE Design Automation Conf.*, 1989, pp. 247-253

[69] D. K. Pradhan, *Fault-Tolerant Computing: Theory and Techniques*, Englewood Cliffs, NJ: Prentice-Hall, 1986.

[70] I. M. Ratiu, A. S. Vincentelli, and D. O. Pederson, "VICTOR: A fast VLSI testability analysis program," *IEEE Test Conf.*, 1982, pp. 397-401.

[71] R. S. Reddy, I. Pomeranz, S. M. Reddy, and, S. Kajihara, "Compact test generation for bridging faults under I_{DDQ} testing," *VLSI Test Symp.*, 1995, pp. 310-316.

[72] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits and Devices.*, vol. 5, no. 1, Jan. 1989, pp. 19-26.

[73] J. M. Soden, C. F. Hawkins, R. K. Gulati, and W. Mao, "I_{DDQ} testing: A review," *J. Electron. Test.*, vol. 3, no. 4, Dec. 1992, pp. 5-16.

[74] J. M. Soden, R. K. Treece, M. T. Taylor, and C. F. Hawkins, "CMOS IC stuck-open fault electrical effects and design considerations," *Int. Test Conf.*, 1989, pp. 423-427.

[75] M. Saraiva, P. Casimiro, M. Santos, J. T. Sousa, F. Goncalves, I. Teixeira, and J. P. Teixeira, "Physical DFT for high coverage of realistic faults," *Int. Test Conf.*, 1992, pp. 642-647.

[76] J. P. Shen, W. Maly, and F. J. Ferguson, "Inductive fault analysis of MOS integrated circuits," *IEEE Design and Test Comp.*, vol. 2, no. 6, Dec. 1985, pp. 13-26.

[77] H. D. Sherali and P. J. Driscoll, "On tightening the relaxations of Miller-Tucker-Zemlin formulations for asymmetric traveling salesman problem," Technical Report, Industrial and Systems Engineering Dept., Virginia Polytechnic Institute and State University, 1996.

[78] J. J. Sousa, F. M. Fernando, M. Goncalves, and J. P. Teixeira, "Physical design of testable CMOS digital integrated circuits," *IEEE J. Solid-State Circuits*, vol. 26, no. 7, July 1991, pp. 1064-1072.

[79] J. H. Stewart, "Future testing of large LSI circuit cards," *Semiconduct. Test Symp.*, Oct. 1977, pp. 6-15.

[80] T. M. Storey and W. Maly, "CMOS bridging fault detection," *Int. Test Conf.*, 1990, pp. 842-851.

[81] H. A. Taha, *Integer Programming*, New York, NY: Academic Press, 1975.

[82] J. P. Teixeira, C. F. B. Almeida, J. A. Gracio, P. A. Bicudo, A. L. Oliveira, and N. Rua, "Bottom-up testing methodology for VLSI," *IEEE Custom IC Conf.*, 1988, pp. 16.6.1-4.

[83] J. P. Teixeira, F. M. Goncalves, and J. H. T. Sousa, "Layout-driven testability enhancement," *European Test Conf.*, 1989, pp. 101-109.

[84] T. W. Williams and K. P. Parker, "Design for testability - A survey," *Proc. IEEE*, vol. 71, no. 1, Jan. 1983, pp. 98-112.

[85] M. J. Williams and J. B. Angel, "Enhancing testability of large scale integrated circuits via test points and additional logic," *IEEE Trans. Comput.*, vol. 22, no. 1, Jan. 1973, pp. 46-60.

[86] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Computer-Aided Design*, vol. 1, no. 1, Jan. 1982, pp. 180-190.

VITA

Salahuddin A. Almajdoub was born in Kuwait in 17 January 1964. He finished his B.Sc. in Electrical Engineering from King Fahd University of Petroleum and minerals in 1985 with honors. In 1988 he finished M.S. in Electrical Engineering from King Fahd University of Petroleum and Minerals. During 1988 and 1989 he worked as a lecturer in University of Bahrain. He finished his Ph.D. in Electrical Engineering in July 1996 from Virginia Tech. He is expected to join the Electrical Engineering Department at the University of Bahrain in September 1996. His current research activities are focused on Hardware Description Languages and Embedded Systems.