TEST GENERATION AND RESYNTHESIS PROCEDURES FOR TEST AND DIAGNOSIS QUALITY

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Naixing Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF DISSERTATION APPROVAL

To my parents for their unconditional love and support
and to Yue.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Professor Irith Pomeranz, for her inspiration, encouragement and guidance throughout my Ph.D. study. Her advices and feedback has been priceless. This work would not have been accomplished without her guidance.

I am also greatly thankful to my lab mate Dr. Shraddha Bodhe for always being there encourage me and give me valuable advices.

I also would like to thank Professor Raymond A. Decarlo, Professor Anand Raghunathan and Professor Vijay Raghunathan for serving on my advisory committee. Particularly, I would like to thank Dr. Xijiang Lin and Dr. Brady Benware from Mentor, A Siemens Business, Dr. Bo Yao, Dr. Srikanth Venkataraman, Dr. Enamul Amyeen and Dr. Arani Sinha from Intel Corporation, Professor Sudhakar M. Reddy from University of Iowa for their contributions to this work.

In addition, I would like to thank Semiconductor Research Corporation (SRC) and National Science Foundation (NSF) for providing the grants to support this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Wang, Naixing Ph.D., Purdue University, December 2019. Test Generation and Resynthesis Procedures for Test and Diagnosis Quality. Major Professor: Irith Pomeranz.

Testing and diagnosis are performed to detect and identify manufacturing failures in integrated circuits. In this dissertation, we focus on three important issues in test and diagnosis. The solutions to these issues are implemented using commercial EDA tools. No modification to the commercial tools is required. Thus, they can be easily applied to complex designs with state-of-the-art features. We first address overtesting of delay faults. Overtesting may occur when the circuit is brought into states that cannot be reached during functional operations. We address this issue by generating functional broadside tests using reachable states as scan-in states. Next, we address the issue of improving the resolution of multiple-defect diagnosis by ignoring certain tests. A feature of commercial defect diagnosis tools is used to avoid losing accuracy. Last, we address the issue of avoiding undetectable faults that model potential systematic defects caused by design-for-manufacturability (DFM) guideline violations in a cell-based design. We demonstrate that these undetectable faults tend to cluster in certain areas of the circuit, resulting in circuit areas with low coverage. The missing tests may allow detectable defects in these areas to escape detection. This can impact the defective-parts-per-million (DPPM) and reliability significantly since the defects are likely to be systematic. We address this issue in a cell-based design by eliminating the undetectable faults related to DFM guidelines that are internal and external to cells. We first propose a logic resynthesis procedure to eliminate large clusters of undetectable faults that are internal to cells. Next, we propose a layout resynthesis procedure that eliminates undetectable faults external to cells by making fine changes to the layout so as to fix the corresponding DFM guideline violations.

1. INTRODUCTION

Aggressive scaling of integrated circuit (IC) technologies continues to decrease device sizes and increase circuit complexity. The scaling of integrated circuit (IC) technologies has brought about many benefits including faster devices, lower power consumption, reduced chip sizes and advanced functionality. However, the continuous shrinking of device sizes also increases the systematic imperfection and random variation in a highly complex manufacturing process. As a result, for each smaller process technology node, the chips are increasingly impacted by deviations in manufactured patterns from the intended design. This can impact the yield and defective-parts-per-million (DPPM) significantly. As the occurrence of defects is expected to be prominent, it is more important than ever to perform testing and diagnosis with high quality.

In this dissertation, three important issues related to delay fault testing, defect diagnosis and systematic defects based on design-for-manufacturability (DFM) guidelines are discussed to improve the quality of testing and diagnosis. The solutions to these issues are implemented using commercial EDA tools. No modification to the commercial tools is required. Thus, they can be easily applied to complex designs with state-of-the-art features.

1.1 Delay Fault Testing

1.1.1 Transition Delay Fault Model

Defects that cause faulty timing behaviors of a circuit are modeled by delay faults. One of the most widely used delay fault models is transition delay fault model [1]. The transition fault model captures timing-related defects that cause slow-to-rise transitions or/and slow-to-fall transitions at gate pins in the circuit. Under transition delay fault model, it is assumed that the extra delay caused by a transition fault at a gate pin is large enough such that the delay of every path passing through this gate pin exceeds the desired clock period.

For illustration, in Fig. 1.1, a slow-to-rise transition fault is at the output Y of a two-input AND gate. Input B has a constant logic value 0. The logic value of input A changes from 0 to 1 at t_1. If the circuit is fault-free, the logic value of Y should be changed to 1 at the required time point t_2, where $t_2 - t_1$ is the desired clock period. However, due to the slow-to-rise transition fault at Y, the logic value of Y remains 0 at t_2. Therefore, the circuit cannot operate correctly.

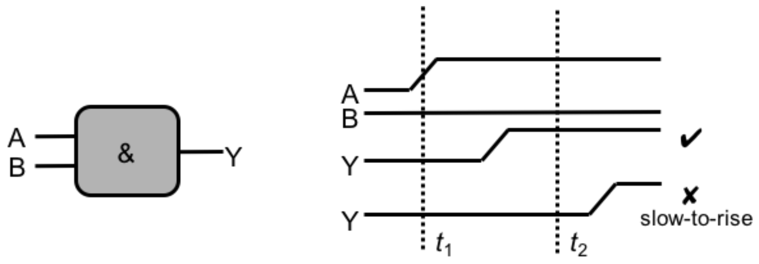

Fig. 1.1.: Example of a transition delay fault.

1.1.2 Scan-based Testing for Transition Delay Fault

Sequential circuits contain state elements such as latches and flip-flops. The existence of state elements increases the complexity of generating tests for the circuit. In order to improve the testability of a sequential circuit, scan structure is inserted by replacing the state elements with scannable state elements (scan cells) and then stitching the scan cells into scan chains [2]. The scan cells can then be used for controlling circuit states and observing test responses.

To test a transition fault in a scan-based circuit, a two-pattern test are required. We denote a two-pattern test by $\langle s_0, a_0 \rangle$ and $\langle s_1, a_1 \rangle$. s_0 and s_1 are state values. a_0 and a_1 are primary input values. The first pattern $\langle s_0, a_0 \rangle$ is used for initializing the target fault site. The second pattern $\langle s_1, a_1 \rangle$ is used for activating the fault by launching a transition at the fault site, and propagating the response to a primary output or state element. If the captured value indicates that the logic involved does not transit as expected during the desired clock period, the transition fault is expected to be present at the target fault site.

One of the widely used types of test for detecting transition delay fault is broadside test [3]. The advantage of broadside test is that it does not require scan enable signal to be shifted at-speed. Therefore, the scan enable does not need to be routed as a timing critical clock. Under a broadside test, s_0 is the initial state that is scanned into the scan cells, and s_1 is determined by the response of the circuit to $\langle s_0, a_0 \rangle$. A broadside test can thus be denoted by $\langle s_0, a_0, a_1 \rangle$. The circuit is tested by first shifting in the scan-in state s_0 at slow speed in test mode. Two at-speed clock cycles are then pulsed for launch transition and capture response in functional mode. Once the test response is captured, the scan-out state can be shifted out at slow speed in test mode. The timing requirement for a broadside test is illustrated in Fig. 1.2. Note that the dead cycle is used for guaranteeing that the scan enable signal is off, and it is optional.

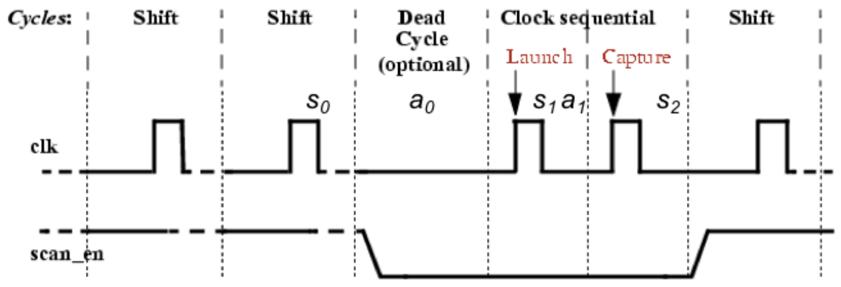

Fig. 1.2.: Timing waveform of a broadside test.

1.1.3 Overtesting and Functional Broadside Test

Scan-based testing may cause overtesting of transition faults when the circuit is brought to states that cannot be reached during functional operations [4]. The existence of overtesting can fail a good circuit, and thus resulting in significant yield loss. Specifically, overtesting can impact the yield in the following ways.

- Overtesting can cause non-functional operations, which can lead to a higher switching activity [5,6]. This can result in excessive current demands, and thus resulting in voltage drops that slow down the circuit and cause a good circuit to fail.

- Overtesting may activate slow logic that can only be activated during non-functional mode, and thus causing the circuit to appear slower [4].

Thus, detecting transition faults under functional operation conditions is necessary in order to address overtesting and avoid unnecessary yield loss.

Different strategies to address overtesting are described in [4–15]. The solutions proposed in [4–8, 12–15] use functional constraints during test generation to avoid non-functional states and the resulting high current demands. These methods do not guarantee functional operation conditions during the functional capture cycles where delay faults are detected, and therefore, do not guarantee that overtesting can be avoided.

The approaches described in [9–11] create functional operation conditions during the functional capture cycles of every generated test by using reachable states as scan-in states. The generated tests are referred to as functional broadside tests, as the functional operations are guaranteed when applying these tests.

1.1.4 Contribution

The procedures for generating functional broadside tests from [9–11] are implemented using academic ATPG tools. The part of contribution of this dissertation is demonstrating that it is possible to use a commercial ATPG tool for generating functional broadside tests. In particular, a commercial ATPG tool is used for addressing the challenge of identifying reachable states that are useful as scan-in states efficiently.

This will allow functional broadside tests to be generated for state-of-the-art circuits with features that are not handled by academic tools.

1.2 Defect Diagnosis

1.2.1 Simulation-based Diagnosis

After generating a test set, the testing of a circuit is accomplished by applying the tests to this circuit on an Automatic Test Equipment (ATE). If the observed responses and the expected responses to all the tests in the test set are the same, then the circuit under test passes the test, otherwise it fails. For circuits that fail during testing, it is important to identify the locations and the root causes of the failures, so as to achieve a fast yield ramp up. For this purpose, the differences between the observed and the expected responses are recorded in a fail-log.

Defect diagnosis is then carried out to analyze the fail-log of every failing circuit. The aim of defect diagnosis is to determine the root causes in the defective circuits. With a high-quality defect diagnosis procedure, yield ramp up can be achieved more quickly. Typically, The output of defect diagnosis is a list of fault candidates that are identified as the potential causes of the failing circuit.

One of the wildly used diagnosis procedures is based on effect-cause fault simulation [16]. The similar procedure is also used in leading commercial defect diagnosis tools. The procedure can be summarized as follows.

1. Perform path-tracing to obtain an initial candidate list by analyzing all the failing tests. An initial candidate fault is identified if it satisfies the following requirements:

    - The fault resides in the fan-in cone of an affected observe point of a failing test.
    - There exists a parity-consistent path from the fault site to the affected observe point.

- If a failing test has more than one affected observe points, the fault must reside in the intersection of all the fan-in cones of all the affected observe points. This is based on the single-defect assumption, such that, for a failing test, only one defect is activated, and the corresponding faulty responses are observed.

2. Perform fault simulate for each fault in the initial candidate list to see if it perfectly explains any of the failing tests. If so, assign it a weight equaling to the number of tests it explains. Store the candidate fault with the greatest weight, and exclude the failing tests explained by it for consideration.

3. The procedure terminates when all the failing tests have been explained, or all the initial candidate faults have been examined. Rank the obtained candidate faults using their weights obtained in Step 2, and report the candidate fault list.

1.2.2 Diagnosis for Multiple Defects

In order to collect additional information about the candidate faults obtained during diagnosis, failure isolation is typically performed through optical and electrical physical tools after the defect diagnosis. Physical failure analysis is the last step in this process to pinpoint the defects or the root causes of failures in the circuit, by taking a defect sample through a scanning electron microscope or transmission electron microscope. Typically, physical failure analysis is applied only if the number of candidate faults small enough, as the process of physical failure analysis is very time-consuming and expensive. However, large sets of candidate faults are typically obtained in the case where multiple defects are present in a circuit, which makes it impossible for physical failure analysis.

High defect densities are very common due to very fine geometries and lithography pushed to its limit with multi-patterning. While defect densities improve as the process matures, multiple defects are prevalent until the process yield reaches very high mature levels. The reason that the existence of multiple defects can cause large candidate fault sets is shown next.

The interactions among the defects that are present together in the circuit may result in an output response that is difficult for the defect diagnosis procedure to analyze. This occurs, for example, if the circuit contains multiple defects, and the interaction among them creates output responses that any single defect in the circuit cannot create alone. Considering a defect diagnosis procedure that is based on a fault model, even if the fault model used perfectly matches every defect in the circuit alone, the procedure may not be able to interpret the joint response of the defects to a particular test. This is illustrated in Table 1.1. In this case, two defects are present in the circuit, and they can be modeled by faults f_0 and f_1 respectively. Suppose that the output responses of f_0 and f_1 under test t are O_0 and O_1, and the joint response of them is O_2. If there exists another fault f_2 that can produce a output response O_2 under test t, the defect diagnosis procedure will conclude that f_2 is likely to be present in the circuit. When this happens multiple times during defect diagnosis, a large candidate fault list can be obtained.

Table 1.1.: Multiple Defects Cause Large Candidate Fault Set

Candidate faults	**Response under t**
f_0	O_0
f_1	O_1
f_0 and f_1	O_2
f_2	O_2

In order to address this issue, the procedures described in [17–19] consider to use reduced test sets for diagnosis, so as to allow the defect diagnosis procedure to compute a more accurate candidate fault set. As a result, the overall defect diagnosis quality can be improved.

1.2.3 Contribution

The procedures from [17–19] reduce the number of candidate faults to a manageable number for physical failure analysis in cases where large candidate fault sets are obtained. However, they may lose the candidate faults that model the defects that are present in a faulty chip. This is inevitable with the defect diagnosis tool they use. In addition, they rely on the ability to modify the defect diagnosis tool, which is not possible for the usage of a commercial defect diagnosis tool.

As part of the work in this dissertation, we develop a new procedure that reduces the number of candidate faults obtained for multiple-defect diagnosis by ignoring certain tests. The main contribution of the new procedure is the following.

- The new procedure does not need to modify the defect diagnosis tool at source code level. This is appropriate when a commercial tool is used for defect diagnosis, and the source code is not available.

- The new procedure uses a partition of the candidate faults into subsets, so as to avoid losing the ones that match the defects present in the circuit. This feature is provided by commercial defect diagnosis tools. Avoiding losing the candidate faults matching the real defects is crucial when physical failure analysis is performed. It helps pinpoint more of the defects that are present in the circuit. It also helps avoid spending physical failure analysis effort on candidate faults that do not model any defects in the circuit.

1.3 Systematic Defects Based on DFM guidelines

1.3.1 Testing Systematic Defects Based on DFM Guidelines

The gap between the feature size and wavelength is increasing due to continuous shrinking of process technology. Due to the drawing of sub-wavelength feature sizes in lithography processes, deformities typically occur in the taped-out chips. As a result, for each smaller process technology node, the chips are increasingly impacted by deviations in manufactured patterns from the intended design. In particular, certain layout features are more difficult to manufacture than others, and are more likely to cause circuit failures. When such features are present multiple times in a chip, they can result in repeated or systematic defects, which can impact the yield and DPPM significantly [20–25]. Due to modeling errors and algorithmic inaccuracies in removing the resulting systematic variations, process-related corrective actions using OPC/RET techniques are not sufficient for acceptable yield and DPPM [26]. Thus, appropriate interventions on design side are inevitable so as to remedy the potential manufacturing issues and address the systematic defects.

Such design interventions are formulated as design rules and DFM guidelines. Design rules are mandatory, and must all be applied when designing the chip. DFM guidelines are typically taken as recommendations, and they are adhered to when possible to improve the quality of the design within the constraints of area, delay and power. When DFM guidelines are not adhered to, potential systematic defects may occur. The relationship between DFM guideline violations and potential systematic defects was discussed in [27–29]. In all these works, the layout sites where DFM guidelines are violated are first pinpointed. The affected transistors are then identified at the schematic level. The expected defect behaviors are translated into gate-level logic faults by using switch-level simulation. Test patterns are generated to target the resulting logic faults so as to avoid potential test holes.

Among the logic faults resulting from DFM guideline violations, there are undetectable faults. The clustering of undetectable faults related to DFM guideline violations in certain areas of the circuit can impact the coverage for the potential systematic defects in these areas. The missing tests may allow detectable defects in these areas to escape detection. This can impact the yield and DPPM significantly.

1.3.2 Contribution

As part of the work in this dissertation, we first demonstrate that the undetectable faults caused by DFM guideline violations tend to cluster in areas of the circuit, resulting in circuit areas with low coverage for potential systematic defects. In order to improve the coverage of the circuit for potential systematic defects caused by DFM guideline violations, we propose two procedures based on logic and layout resynthesis respectively. The procedures are developed for cell-based designs. For this discussion we distinguish between faults that are internal to the standard cells (internal faults), and faults that are external to the standard cells (external faults).

First, we propose a procedure that is based on logic resynthesis followed by physical design process to eliminate undetectable internal faults. In this process of this procedure, the large clusters of undetectable faults related to DFM guideline violations are eliminated, and thus the coverage of the circuit for potential systematic defects is improved. Specifically, the procedure targets clusters of potential systematic defects related to DFM guidelines when the clusters may remain uncovered. It leaves other areas unaffected by logic resynthesis. This is important for allowing the design to be implemented in the constraints of delay, power and area.

The proposed logic resynthesis procedure is based on the fact that every time a gate, or an instance of a standard cell, is used in the circuit, it introduces the same internal faults caused by DFM guideline violations. The procedure eliminates the undetectable internal faults by resynthesizing the circuit with standard cells containing fewer internal faults. The procedure does not require to modify the cell library, but only uses different standard cells from the same cell library. The total number of undetectable internal and external faults is only allowed to decrease monotonically when applying the procedure. Hence, the increase in circuit coverage and the reduction in the number of undetectable faults related to DFM guideline violations are significant when undetectable internal faults are eliminated by logic resynthesis.

Next, we propose a layout resynthesis procedure that makes fine changes to the layout so as to improve the coverage of areas with low coverage because of the presence of undetectable external faults. The proposed layout resynthesis procedure eliminates undetectable external faults by fixing the DFM guideline violations that lead to them. The procedure prefers to eliminate faults whose effect on the circuit coverage is more significant. The DFM guideline violations are fixed by automatically changing the layout with the help of a place and route tool. The procedure maintains the same critical path delay, power consumption and die area when making changes to the layout.

The layout resynthesis method of the proposed procedure is not the main contribution of the proposed procedure. The contribution is using DFM guidelines to identify areas of the circuit with low coverage, and improving their coverage with the help of layout resynthesis. From a test point of view, we demonstrate which DFM guideline violations need to be fixed first so as to improve the coverage for potential systematic defects. As part of this solution, we suggest a layout-based coverage metric that can be used for identifying areas with low coverage.

Both two resynthesis procedures described above can be embedded into a standard cell based design flow as follows. In a cell based design flow, after the initial design, several iterations of incremental logic and physical design processes are typically required for satisfying the design constraints of delay, power and area. Both two procedures are iterative, and can thus fit within the overall iterative design process. In particular, an iteration of the design process can include one or more iterations of the proposed procedures to eliminate undetectable faults in poorly covered circuit areas, and improve the coverage for potential systematic defects in these areas. For a large chip, to maintain an acceptable computational effort, the proposed procedures can be applied to each logic block separately.

1.4 Organization

This dissertation is organized as follows. Chapter 2 describes the functional broadside test generation procedure to address the overtesting of delay faults. Chapter 3 describes the procedure to improve the quality of multiple defect diagnosis by removing and selecting tests. Chapter 4 and Chapter 5 describe the logic and layout resynthesis procedures for avoiding undetectable faults caused by DFM guideline violations respectively. The summary of this dissertation is shown in Chapter 6.

2. FUNCTIONAL BROADSIDE TEST GENERATION USING A COMMERCIAL ATPG TOOL

Scan-based tests may lead to overtesting of delay faults by bringing a circuit to states that the circuit cannot enter during functional operation. Functional broadside tests address this issue by using reachable states as scan-in states. Different strategies for generating functional broadside tests have been studied and implemented by academic tools. The main challenge that these procedures address is the identification of reachable states that are useful as scan-in states. This chapter describes the generation of functional broadside tests using a commercial test generation tool. Our results demonstrate that it is possible to generate functional broadside tests without requiring any modifications to the commercial tool, and using the tests that the tool produces to obtain reachable states. This is expected to enable the generation of functional broadside tests for state-of-the-art designs that cannot be handled by academic tools. To demonstrate this point, we apply the procedure to two large logic blocks of the OpenSPARC T1 microprocessor.

2.1 Introduction

The rapid decrease of geometry size and increase of clock frequency in ICs have lead to a growing of timing-related defects. These timing-related defects can generally be modeled by delay faults. One of the widely used delay fault models is the transition fault model.

Scan-based tests may cause overtesting of delay faults when the circuit is brought to states that cannot be reached during normal functional operation [4]. This can cause yield loss in the following ways. (1) Non-functional operation may lead to a higher switching activity, which will cause excessive current demands [5, 6]. These current demands may lead to voltage drops that slow down the circuit and cause a good circuit to fail. (2) Slow logic that can only be activated during non-functional operation may cause the circuit to appear slower [4]. Based on theses observations, detecting delay faults under functional operation conditions is necessary in order to avoid unnecessary yield loss.

Different strategies to address overtesting are described in [4–15]. To reduce the current demands, [5] and [6] describe solutions for generating scan-based tests with low switching activity. In [7] and [8], methods are proposed to avoid detecting functionally untestable faults. The methods in [12–14] apply functional constraints extracted from the circuit to avoid unreachable states. The tests generated by [12–14] are called pseudo-functional tests since the functional constraints may not avoid all the unreachable states. Pseudo-functional tests are also generated in [15]. These solutions do not guarantee functional operation conditions during the functional capture cycles where delay faults are detected, and therefore, do not guarantee that overtesting will be avoided.

The test sets in [9–11] create functional operation conditions during the functional capture cycles of a test by using reachable states as scan-in states. In [9] and [10], the methods target circuits that can be synchronized; while in [11], the method targets circuits with a hardware reset state. Two-cycle broadside tests are generated in [10] and [11]. The tests are referred to as functional broadside tests.

Functional constraints on primary input sequences exist when a circuit is embedded in a design. To simplify the generation of functional broadside tests, the test generation procedures typically assume that the primary input sequences are unconstrained during functional operation. We make the same simplifying assumption in this work.

With reachable scan-in states and unconstrained primary input vectors, the state transitions caused by a functional broadside test that is generated by the procedure from [10] or [11] can occur during functional operation. Delay faults detected by these tests can also be activated during functional operation, and the switching activity can occur during functional operation. Therefore, the tests avoid the unnecessary yield loss described before.

We denote a broadside test by $\langle s_{\mathrm{i}}, a_0, a_1 \rangle$, where s_{i} is the scan-in state, and a_0 and a_1 are primary input vectors that are applied during two functional capture cycles. In a functional broadside test, s_{i} is a reachable state, and the state transitions from s_{i} under a_0 and then a_1 can be obtained during functional operation as well.

The procedures for generating functional broadside tests from [10, 11] are implemented using academic tools. The goal of this work is to show that it is possible to use a commercial tool for implementing the generation of functional broadside tests. In particular, a commercial tool is used for addressing the challenge of identifying reachable states that are useful as scan-in states efficiently. This will allow functional broadside tests to be generated for state-of-the-art circuits with features that are not handled by academic tools.

This chapter describes an implementation of a functional broadside test generation procedure that is based on the procedure presented in [11] using a commercial ATPG tool. The procedure is applied to two large logic blocks of the OpenSPARC T1 microprocessor to demonstrate its applicability to such designs.

The test generation procedure described in this chapter starts from a single known reachable state denoted by s_{r}. For circuits with hardware reset, s_{r} is the reset state. For circuits that are synchronized by applying a synchronizing sequence, s_{r} is entered after applying the synchronizing sequence. Similar to the procedure in [11], we use the all-0 state as s_{r}.

The test generation procedure is iterative. In every iteration, it uses a set of reachable states R_{next} as scan-in states for functional broadside tests. Initially, R_{next} contains only s_{r}. In an arbitrary iteration, the procedure uses the commercial tool to generate functional broadside tests with scan-in states from R_{next}. The scan-out states of the generated tests, which are also reachable states, are used for reconstructing R_{next} for the next iteration. Thus, the commercial tool is used for test generation as well as for identifying new reachable states.

The commercial tool can generate tests with several additional features that are also useful for functional broadside tests. We discuss these features next. In all the cases, we rely on the commercial tool to produce tests with the additional features.

The limitations of ATEs may require the input vectors applied during different functional capture cycles of a broadside test to be equal. The commercial tool can be applied with this constraint.

The studies in [30–32] have demonstrated that the number of clock cycles needed for test application, as well as the test data volume, can be reduced significantly with multi-cycle tests. This was shown for functional broadside tests in [32]. Multi-cycle tests were also used in [33]. We denote an l-cycle broadside test by $\langle s_{\text{i}}, a_0, ..., a_{l-1} \rangle$, where s_{i} is the scan-in state, and a_0, ..., a_{l-1} are primary input vectors that are applied during l consecutive functional capture cycles. It is possible to use the commercial tool for generating multi-cycle tests. The commercial tool is also used for fault simulation of multi-cycle tests.

We focus on the use of the commercial tool for the generation of multi-cycle functional broadside tests with unconstrained primary input vectors, for the following reasons. (1) Two-cycle tests are a special case of multi-cycle tests, and the procedure we develop can be used for generating two-cycle tests as a special case. (2) In addition to providing test compaction, the use of multi-cycle tests reduces the number of calls to the commercial tool during the iterative test generation procedure. This reduces the runtime. (3) Equal primary input vectors reduce the fault coverage achievable for benchmark circuits.

2.2 Generation of Functional Broadside Tests

The test generation procedure from [11] iterates through a process where it generates functional broadside tests and identifies new reachable states. Fig. 2.1 presents the structure of the functional broadside test generation procedure. We implement the parts of the procedure around the commercial ATPG tool using TCL, Python, and C shell language. In addition, the commercial tool is used whenever fault simulation or logic simulation is needed.

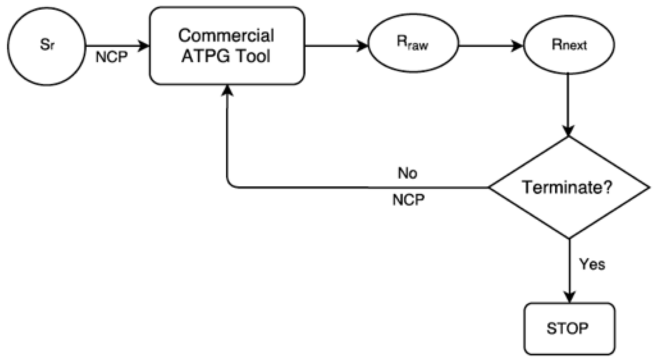

Fig. 2.1.: Functional Broadside Test Generation Flowchart.

The procedure maintains a set of target faults that is denoted by Φ. As tests are generated, Φ is simulated with fault dropping. The procedure terminates when Φ is empty, or when certain other termination conditions are satisfied.

In the following subsections of this section, we describe the details of this procedure. In subsection 2.2.1, we describe the details of test generation. In subsection 2.2.2, we describe the details of finding reachable states to construct R_{next}. In subsection 2.2.3, we describe the termination conditions. In subsection 2.2.4, we describe our strategy for generating multi-cycle tests.

2.2.1 Test Generation

In this part, we describe the procedure for generating functional broadside tests using the commercial tool. Suppose that R_{next} contains n reachable states, denoted by $s_0, s_1, ..., s_{n-1}$. For every state s_i from R_next, the procedure needs to be able to call the commercial tool to generate functional broadside tests with scan-in state s_i.

There are two ways to specify a scan-in state for the commercial tool. The first is through ATPG constraints on the values of the state variables. This can be done using a command to specify the constrained values of the state variables before performing test generation. This approach turned out to be slow. In particular, it can only use one reachable state to generate functional broadside tests in each call. A significantly faster approach, which we used for our implementation, is to use what is called the Named Capture Procedure (NCP) [34].

NCP was originally introduced for explicitly defining the legal relationship between external and internal clock sequences, and controlling the complex clock-generator circuits. In this work, we use an NCP to define a scan-in state by specifying a scan-in value for every scan cell. In addition, we use NCP to specify the number of functional capture cycles in a test. We use multiple NCPs in each call to the commercial tool. This makes it possible to use several reachable states as scan-in states in each call.

The number of NCPs we use in a call to the commercial tool is denoted by m. The number of capture cycles is denoted by l, and it is the same in all the m NCPs. The selection of values for m and l is described later. For each call, we use the first m reachable states $s_0, s_1, ..., s_{m-1}$ from R_next to define NCPs. If there exist fewer than m reachable states in R_next, we use all of them. With these m NCPs, the commercial tool generates l-cycle functional broadside tests to detect as many faults from Φ as possible. The set of tests that is returned is denoted by Ψ_cur. We apply fault simulation with fault dropping of Φ under Ψ_cur. We also add the tests to a test set denoted by Ψ. We then remove $s_0, s_1, ..., s_{m-1}$ from R_next.

The test generation procedure described above is shown next.

Procedure 2.1 Test Generation with R_{next}

```
1: Define $m$ NCPs for the first $m$ reachable states $s_0, s_1, ..., s_{m-1}$ in $R_{\text{next}}$.
2: Use the commercial tool to generate functional broadside tests for the faults in $\Phi$
   with the NCPs defined in step 1. Collect the generated tests into $\Psi_{\text{cur}}$.
3: Run fault simulation of $\Phi$ under $\Psi_{\text{cur}}$, and remove the detected faults from $\Phi$. Add
   $\Psi_{\text{cur}}$ to $\Psi$.
4: Remove $s_0, s_1, ..., s_{m-1}$ from $R_{\text{next}}$.
```

2.2.2 Finding Reachable States and Constructing R_{next}

In this part, we describe the procedure we use to update the set of reachable states R_{next} for the next iteration.

Considering the tests that the commercial tool generates, the procedure collects the scan-out states, and places them in a set denoted by R_{raw}. The scan-out state of a functional broadside test is a reachable state for the following reason. With a reachable scan-in state, the test takes the circuit through state-transitions that the circuit can make during functional operation. Therefore, the circuit visits reachable states during the test. This includes the final state, which is the scan-out state of the test. Although all the states that the circuit visits during a functional broadside test are reachable states, we only use the scan-out state because it is available after fault simulation, while extracting other states requires additional simulations. In addition, these states have no special features compared with the scan-out states. Therefore, using these states does not increase the fault coverage.

To avoid using the same reachable states in different iterations, the procedure maintains a set called R_{used} that includes all the reachable states that it has used. For the next iteration, the procedure includes in R_{next} every state from R_{raw} that is not included in R_{used}.

If R_{next} is empty, we need to find more reachable states using a different process. Different strategies for finding reachable states have been described in [9] and [11]. The strategy we use here is the following.

For every state s_i in R_{next}, we apply a preselected number r of random input vectors $v_0, v_1, ..., v_{r-1}$. We simulate each pattern, denoted by $\langle s_i, v_k \rangle$, to find a next-state, which is also reachable. The state is added to a set denoted by R_{sim}. The states in R_{sim} that are not included in R_{used} are added to R_{next}. If R_{next} is empty after using R_{sim}, we use the states in R_{sim} to generate additional reachable states using the same process with random primary input vectors. We only apply this process once again. It does not indicate all the reachable states have beeen visited if R_{next} is empty after applying this process. However, it is desirable to terminate this process after one iteration, in order to limit the runtime.

The procedure described above is shown next. The procedure stops when R_{next} is not empty to allow the test generation process to continue to the next iteration. The test generation process stops if R_{next} is empty after Procedure 2.2 terminates.

Procedure 2.2 Finding Reachable States and Constructing R_{next}

```
1: For each newly generated test in $\Psi_{cur}$, add its final state to $R_{raw}$.
2: For each state in $R_{raw}$, add it to $R_{next}$ and $R_{used}$ if it is not included in $R_{used}$.
3: If $R_{next}$ is not empty, stop.
4: Simulate each state currently in $R_{raw}$ with random input vectors $v_0, v_1, ..., v_{r-1}$
   and add the states into $R_{sim}$.
5: Add the states in $R_{sim}$ that are not in $R_{used}$ into $R_{next}$ and $R_{used}$.
6: If $R_{next}$ is not empty, stop.
7: Repeat steps 4, 5 and 6 with $R_{sim}$.
8: Stop.
```

2.2.3 Termination Conditions

The test generation procedure terminates if or R_{next} (after the application of Procedure 2.2) are empty. In addition, the procedure terminates when it appears that the fault coverage has saturated. This is measured as follows.

Suppose that after i iterations the fault coverage is fc_i. Suppose that after $2i$ iterations the fault coverage is still fc_i. This implies that doubling the number of iterations did not increase the fault coverage. In this case, the procedure terminates.

2.2.4 Multi-cycle Test Generation

Experimental results show that the runtime of an iteration increases with the number of capture cycles of a functional broadside test. Therefore, it is important to avoid generating multi-cycle tests with large numbers of capture cycles when this is not necessary for test compaction. At the same time, using multi-cycle tests with more functional capture cycles increases the number of faults that the commercial tool can detect with every additional test, thus reducing the number of iterations. To balance these observations, we generate tests with decreasing numbers of functional capture cycles, and use smaller numbers of reachable states for the higher numbers of functional capture cycles. The process is described by Procedure 2.3.

In Procedure 2.3, we use Procedure 2.1 to generate multi-cycle functional broadside tests, and Procedure 2.2 to find more reachable states and construct R_{next}. The pair $(l,\ m)$ implies that we use m reachable states from R_{next} to generate l-cycle tests. The selection of values for l and m is based on experimental results that show the following. (1) Using $l > 8$ results in a high runtime, and we avoid it. (2) For $l = 8$ it is important to limit the number of reachable states in order to limit the runtime. The lowest possible limit is $m = 1$ and we use this value. (3) After using $l = 8$, using $l = 7$ and 6 does not increase the fault coverage significantly enough to use these values. (4) Using $l = 5$, 4, 3 and 2 is important for increasing the fault coverage. It is important to continue limiting the number of reachable states in order to limit the runtime. Increasing m gradually to use $m = 8$, 16, 32 and 256 proved to be an effective choice. (5) Iterative calls to the commercial tool are cost-effective only with $l = 2$.

2.3 Experimental Results

The procedure described in Section 2.2 is applied to ISCAS-89, ITC-99 and OpenCores® [35] benchmark circuits. To further demonstrate the capability and effectiveness of the procedure, it is also applied to two large blocks of the OpenSPARC

Procedure 2.3 Multi-cycle Test Generation

```
1: Assign $R_{\text{next}} = \{s_{\text{r}}\}$.
2: for $(l, m) = (8,1), (5,8), (4,16), (3,32)$ do
3:     Call Procedure 2.1 to generate $l$-cycle tests with $R_{\text{next}}$ and $m$.
4:     Call Procedure 2.2 to update $R_{\text{next}}$.
5:     if any termination condition is satisfied then
6:         stop.
7:     end if
8: end for
9: if no termination condition is satisfied then
10:    Call Procedure 2.1 to generate 2-cycle tests with $R_{\text{next}}$ and $m = 256$.
11:    Call Procedure 2.2 to update $R_{\text{next}}$.
12: end if
```

T1 [36] microprocessor. OpenSPARC T1 is a 64-bit open-source microprocessor. It has eight cores and each core can support up to four threads for a total of thirty-two threads. Within OpenSPARC T1, we apply the proposed procedure to a single SPARC core (*sparc*) and the floating-point unit (*sparc-fpu*). We run the procedure on a Linux machine with 2.6GHz processors.

We experimented with different values of r, which is the number of random input vectors to apply during the simulation procedure for finding reachable states. Both the computational effort and the number of reachable states increase with r. We found experimentally that $r = 256$ balances the two parameters well.

For comparison, we use the commercial tool to generate a multi-cycle broadside test set without considering functional constraints. When multi-cycle tests are generated for the purpose of increasing the fault coverage, it is typical to supplement two-cycle tests with tests that have higher numbers of capture cycles. To match our functional broadside test set, we generate 2-, 3-, 4-, 5-, and 8-cycle tests, in this order. In every case, the commercial tool generates tests to detect as many faults as possible that were not detected with lower numbers of capture cycles. The resulting test set is denoted by Ψ_{nonfunc}.

We show the results in Table 2.1. In column G, we show the number of gates in the circuit. In column *ff*, we show the number of flip-flops in the circuit. In column

F, we show the number of transition faults. In column R, we show the number of reachable states the procedure used to generate tests. It should be noted that not all the reachable states lead to a contribution to the test set. In column *Tests*, we show the number of tests generated by the proposed procedure. We also show the number of multi-cycle tests in the non-functional broadside test set Ψ_{nonfunc} generated by the commercial tool. Next, we show the fault coverage achieved by the proposed procedure. This is followed by the fault coverage of Ψ_{nonfunc}. In the first column under *ATPG time*, we show the runtime of the commercial tool as part of the proposed test generation procedure. We then show the total runtime of the proposed test generation procedure. We also show the runtime for generating Ψ_{nonfunc} in the last column under *ATPG time* for comparison.

Table 2.1.: Multi-cycle Functional Broadside Tests with Unconstrained Input Vectors

Circuit	**G**	**ff**	**F**	**R**	**Tests**		**Fault Coverage**		**ATPG time**		
					func	**non func**	**func**	**non func**	**tool**	**func**	**non func**
b04	1176	66	2400	709	108	117	81.25	85.38	15s	38s	4s
b10	332	17	746	42	41	53	84.58	86.73	2s	5s	6s
b14	6780	215	21444	481	480	551	93.19	93.91	121s	137s	155s
b20	14641	430	47986	2674	1473	904	92.7	94.46	906s	1020s	379s
s1423	1169	74	2090	504	120	108	84.3	92.73	10s	31s	5s
s5378	2535	163	4468	618	117	174	70.12	89.86	2s	16s	2s
s35932	22909	1728	26456	635	113	70	96.31	97.18	25s	115s	9s
s38584	18577	1164	34646	2973	1072	329	82.57	95.41	262s	545s	19s
aes_core	23516	530	85118	472	471	599	99.94	99.95	368s	497s	55s
des_perf	157078	8808	355028	6190	6195	202	100	100	12119s	17542s	76s
i2c	2047	128	3872	439	155	122	86.24	94.4	8s	20s	4s
systemcaes	13644	670	29306	1377	294	383	61.6	94.2	606s	696s	99s
systemcdes	4301	190	11182	684	83	164	98.33	99.82	28s	50s	5s
spi	4763	229	11864	1039	938	965	98.68	99.29	174s	198s	20s
usb_phy	1391	98	2010	353	52	75	32.95	92.9	1s	10s	2s
wb_dma	8621	523	15258	193	192	212	71.21	98.94	3s	7s	5s
sparc-fpu	87748	4431	333420	2599	1594	785	92.13	98.73	3324s	4329s	842s
sparc	350468	17942	658844	4791	4678	1217	86.37	92.11	14453s	19453s	8167s

From Table 2.1 it can be seen that the proposed procedure achieves a fault coverage that is typically close to the one achieved by non-functional broadside tests. The existence of functionally redundant logic in the circuits leads to functionally-undetectable faults that cannot be detected by functional broadside tests. These faults may be detectable by non-functional broadside tests, resulting in the fault coverage differences. The difference in fault coverage is larger when a circuit contains more functionally redundant logic. Differences between the fault coverages that can be achieved with functional and non-functional tests were also observed in [10] and explained by the existence of functionally-undetectable faults [37]. Close-to-functional tests can be used to reduce the occurrence of overtesting while increasing the fault coverage (or reducing the fault coverage difference). However, using close-to-functional tests does not guarantee that unreachable states will be avoided. In this chapter, we focus on functional broadside tests, in order to ensure that overtesting is avoided.

The need to satisfy functional constraints also causes the number of functional broadside tests to be higher than the number of non-functional tests for a similar fault coverage. With reachable states as scan-in states, fewer faults can be detected by each functional broadside test, and more tests are needed for detecting the same or similar number of faults.

The number of reachable states is typically larger than the number of tests. This is reasonable, since not all the reachable states contribute to the functional broadside test set.

The proposed multi-cycle functional broadside test generation procedure has a higher runtime than when non-functional broadside tests are generated. The proposed procedure needs to iteratively call the commercial tool to generate functional broadside tests, and it needs to compute reachable states. This increases the runtime. In addition, because of the simplicity of Python, we use it to implement the procedure for finding reachable states and constructing R_{next}. Running Python scripts increases the runtime. For complex logic blocks such as *sparc-fpu* and *sparc*, the runtime of the proposed procedure is around 5 and 2 times the runtime of the non-functional broadside test generation procedure, respectively. This demonstrates the feasibility and scalability of the proposed multi-cycle functional broadside test generation procedure.

To further illustrate the benefits of using multi-cycle functional broadside tests, in Table 2.2, we present the results obtained when the procedure uses only the all-0 state as a scan-in state. We generate functional broadside tests with different numbers of capture cycles for comparison. The results are shown for b04, b14, s1423 and des_perf. For each of the circuits and every number of capture cycles, we show the number of functional broadside tests, the fault coverage achieved by the tests, and the runtime.

Table 2.2.: Fault Coverage Achieved with All-0 State

Circuit	**2-cycle**			**3-cycle**			**5-cycle**			**8-cycle**		
	T	**fc**	**time**	**T**	**fc**	**time**	**T**	**fc**	**time**	**T**	**fc**	**time**
b04	2	7.88%	1s	51	47.17%	1s	95	69.33%	4s	96	78.71%	10s
b14	275	48.26%	2s	311	53.22%	5s	622	90.49%	16s	364	91.18%	35s
s1423	27	32.97%	1s	46	53.01%	2s	73	67.61%	5s	65	80.38%	11s
des_perf	89	9.78%	65s	133	17.28%	209s	174	28.31%	954s	89	39.53%	5532s

It can be seen that the fault coverage increases rapidly as the number of capture cycles increases. The comparison shows that generating multi-cycle functional broadside tests can reduce the need to find additional reachable states.

Next, we consider an additional parameter of the non-functional tests, the distance between the scan-in state and a reachable state. For functional broadside tests this distance is zero. For non-functional tests, a lower distance implies that the circuit operates closer to functional operation conditions. This is the basis for the generation of what are called partially-functional broadside tests. An accurate computation of the distances requires enumeration of all the reachable states, which is infeasible. We obtain a pessimistic estimate of these distances by using the set R_{used} of all the reachable states that were computed during the generation of functional broadside tests. Let S be the set of scan-in states of the non-functional broadside test set. For a state s in S, the minimum distance between s and a reachable state in R_{used} is denoted by $d(\text{s})$. For a circuit with k state variables, we include in a subset S_{p} every state s from S for which $d(\text{s}) \leq p\%\ k$. When p is small, the states in S_{p} have low distances from reachable states and thus maintain close-to-functional operation conditions.

Table 2.3 shows the percentage of tests included in S_{10}, S_{20}, S_{30} and S_{40}.

Table 2.3.: Hamming Distance

Circuit	S_{10}	S_{20}	S_{30}	S_{40}
s38584	0	0	0	0
systemcaes	0	0	0	1.04
usb_phy	0	0	0	10.67
s5378	0	0	0	41.23
wb_dma	0	0	3.21	5.33
sparc	0	0	4.55	10.11
des_perf	0	0.5	4.67	21.1
fpu	0.07	0.22	5.11	15.81
b20	0.39	1.3	7.1	27.55
systemcdes	1.67	5.21	7.99	59.31
aes_core	2.11	5.13	10.11	71.11
spi	2.44	3.56	7.33	51.71
i2c	2.51	6.31	18.03	83.32
b04	2.56	9.21	10.26	99.14
s35932	3.11	3.11	7.97	29.99
b14	4.31	5.77	12.11	98.55
s1423	4.51	7.43	18.52	93.52
b10	9.23	31.21	61.33	100

From Table 2.3 it can be seen that only small percentages of non-functional tests have a distance of 10% or lower from a reachable state in R_{used}. This is consistent with the observation made in [10] that non-functional tests rarely use reachable states accidentally. As p increases, the number of states in S_{p} increases. However, even with $p = 30$ or 40, only small percentages of states in S have $d(\text{s}) \leq p\%\ k$. This implies that non-functional tests do not create close-to-functional operation conditions accidentally.

3. IMPROVING THE RESOLUTION OF MULTIPLE DEFECT DIAGNOSIS BY REMOVING AND SELECTING TESTS

Earlier works showed that the resolution of defect diagnosis when multiple defects are present in a chip can be improved by instructing the defect diagnosis procedure to ignore certain tests. Specifically, these procedures reduce the number of candidate faults when the defect diagnosis procedure produces large numbers of candidates. Diagnosis with a large number of candidates poses challenges to failure isolation as optical emission and electrical probing physical tools need to eliminate a large number of candidates to isolate the defects. The procedures from the earlier works improved the diagnostic resolution by reducing the number of candidates at the cost of a reduced accuracy, or a reduced overlap between the candidates and the defects present in the faulty chip. In addition, they relied on the ability to modify the defect diagnosis tool. This chapter develops a procedure that improves the diagnostic resolution for multiple defects by ignoring certain tests without modifying the defect diagnosis tool. Moreover, the procedure uses a feature of commercial defect diagnosis tools to avoid losing accuracy. Experimental results for multiple defects indicate that reductions in the numbers of candidate faults are typically achieved without losing accuracy.

3.1 Introduction

A defect diagnosis procedure identifies the locations of the defects in a faulty chip using the faulty output response produced by the chip [38–57]. The output of a defect diagnosis procedure is a set of faults that is referred to as a candidate fault set. After obtaining a candidate fault set, failure isolation is performed through optical and electrical physical tools to obtain additional information about the candidates. Physical failure analysis is the last step in this process, taking a defect sample through a scanning electron microscope or transmission electron microscope. Physical failure analysis is applied only if the candidate fault set is small enough (such as 20 or fewer candidate faults). Large sets of candidate faults are obtained in the case where multiple defects are present in a chip. In modern process technologies, due to very fine geometries and lithography pushed to its limit with multi-patterning, high defect densities are very common. It takes a while for the process to reach a mature yield level. With high defect densities, multiple defects are common. While defect densities improve as the process matures, multiple defects are prevalent until the process yield reaches very high mature levels. The interactions between the defects that are present together in the circuit may result in an output response that is difficult for the defect diagnosis procedure to analyze. As a result, the procedure may yield a large set of candidate faults. If a candidate fault set is large, diagnostic tests may be added and used to reduce the number of faults in the candidate fault set. The additional tests are expected to improve the defect diagnosis results by providing more information.

However, the inclusion of a test in the test set used for diagnosis does not always improve the results of defect diagnosis. The observation that a defect diagnosis procedure does not require the output response obtained from the complete test set to produce accurate results is the basis for the approaches described in [58–60]. The need to ignore certain tests when computing the actual output response from a compacted output response is noted in [61]. The procedure in [17] goes further to show that certain tests reduce the resolution of diagnosis, and removing them from consideration is advantageous.

Improvements of the procedure from [17] are described in [18] and [19]. Similar to [17], the procedure from [18] is also based on the removal of tests from the test set used for diagnosis. In contrast, the procedure from [19] starts from an empty test set, and adds tests one by one such that the best diagnosis results will be obtained after the addition of every test. After selecting one test, the procedure typically produces a single candidate that matches one of the defects that exists in the circuit. As the number of tests is increased, the number of candidate faults increases, and more of the defects are identified correctly.

The procedures from [17–19] reduce the number of candidate faults to a manageable number in cases where the defect diagnosis procedure produces large sets of candidate faults, thus improving the diagnostic resolution. However, they may lose accuracy by identifying fewer of the defects that are present in a faulty chip. This is inevitable with the defect diagnosis tool they use. In addition, they rely on the ability to modify the defect diagnosis tool.

As part of the work in this dissertation, we develop a new procedure that improves the diagnostic resolution for multiple defects by ignoring certain tests. The main differences between the new procedure and the previous ones are the following. (1) The new procedure does not need to modify the defect diagnosis tool. This is appropriate when a commercial tool is used for defect diagnosis, and the source code is not available. (2) The new procedure uses a partition of the candidate faults into subsets, so as to avoid losing accuracy. This feature is provided by commercial defect diagnosis tools. When n defects are present in a faulty chip, the defect diagnosis tool is typically able to compute n subsets of candidate faults where each subset corresponds to one of the defects that is present in the faulty chip. By considering the subsets one by one, and ensuring that the same number of subsets is obtained after test removal, the procedure described in this chapter avoids losing candidates that match the defects. It thus rarely loses accuracy. A higher accuracy is crucial when physical failure analysis is applied. It helps single out more of the defects that exist in the circuit. It also helps avoid spending physical failure analysis effort on candidate faults that do not correspond to any defects in the circuit.

Considering the issue of modifying the defect diagnosis tool, the procedures from [17–19] call the defect diagnosis procedure not only with reduced sets of tests, but also with reduced sets of faults from which candidates may be selected. After computing a basic set of candidate faults C_{B} using the complete test set, these procedures do not require new candidate faults to be identified. As tests are removed or selected, the procedures compute new sets of candidate faults out of the faults in C_{B}. This reduces the computational effort for additional calls to the defect diagnosis procedure. With the defect diagnosis tool used in this chapter, limiting the set of tests is possible without interfering with the tool, but limiting the set of faults requires the tool to be modified. We prefer to use the tool as a black box, and avoid interfering with its internal operation.

To address the computational effort, we develop a new algorithm that has two phases. The first phase is based on the removal of subsets of tests without omitting any candidate faults. The second phase is based on the selection of subsets of tests from the test set obtained in the first phase, so as to obtain smaller sets of candidate faults, and thus improve the resolution of defect diagnosis. In the first phase, most of the tests are removed from the initial test set to reduce the number of tests considered by the second phase. Experimental results show that the reduction in the computational effort of the second phase is higher than the additional computational effort required by the first phase. Consequently, the first phase improves the efficiency of the new algorithm significantly.

As in [17–19], considering fewer tests for defect diagnosis implies that the defect diagnosis procedure is given a subset of tests with their corresponding output responses. A new application of the test set to the faulty chip in order to obtain a new output response is not required.

3.2 Background

The following example explains the reason why the diagnostic resolution can be potentially improved by ignoring certain tests, when multiple defects are present in a chip.

Considering a defect diagnosis tool that is based on a fault model, suppose that the fault model used by the tool can perfectly match every one of the defects that are present in the chip if they are present alone. However, the interactions between the defects under a particular test may result in an observed response that is different from the response of any one of the faults. The removal of such tests from consideration by the defect diagnosis tool improves the results of defect diagnosis. For example, suppose that two defects are present in a faulty chip, modeled by faults f_0 and f_1. Considering a test t, suppose that the observed responses produced by f_0 and f_1 are z_0 and z_1, respectively. Suppose that when f_0 and f_1 are present in the circuit together, the joint observed response produced is z_2. Other faults, such as f_2, in the fault model may produce z_2 under test t individually. Therefore, the defect diagnosis tool would conclude that f_2 might be present in the chip based on the observed response under test t. A more accurate candidate fault set can be computed by ignoring test t.

We denote the defect diagnosis tool by $Diag()$. The procedure described in this chapter first computes a basic candidate fault set, denoted by C_{B}. This is accomplished by calling the defect diagnosis procedure $Diag()$ with the complete test set, denoted by T_{B}. Every additional candidate fault set that the procedure computes is intersected with C_{B}. We do this because a small subset of T_{B} does not provide sufficient information to produce accurate diagnosis results by itself [19]. Similar to [19], the procedure described in this chapter considers the candidate faults in C_{B} as appropriate and sufficient, thus, it does not attempt to obtain new candidate faults that are not contained in C_{B}. By taking intersections with C_{B}, it only attempts to reduce the number of candidate faults in C_{B} to improve the resolution of defect diagnosis.

To define quality metrics for the results of defect diagnosis, we assume that the defects in the chip are best described by a subset of modeled faults f_{in}. Let C be candidate fault set obtained for f_{in}. The number of subsets of candidate faults in C is denoted by n. The overlap between C and f_{in} is denoted by $OVLP = C \cap f_{\mathrm{in}}$. We define the following quality metrics.

- ***Coverage***: The diagnosis coverage is defined as $COV = |OVLP|/|f_{\mathrm{in}}|$. A higher coverage indicates that the defect diagnosis procedure singles out more of the defects correctly.

- ***Precision***: The diagnosis precision is defined as $PRC = |OVLP|/n$. A higher precision is important when physical failure analysis is guided by the subsets of candidate faults. It helps avoid spending physical failure analysis effort on candidate faults that do not correspond to any defects in the circuit. The precision rarely reduces by keeping n unchanged.

- ***Resolution***: The diagnosis resolution is defined as $RESO = |OVLP|/|C|$. A higher resolution implies that the fraction of correct locations is higher, which facilitates physical failure analysis. The resolution is increased, as well as in [17–19], by reducing C.

3.3 Procedure Based on Test Removal

In this section, we describe the procedure for removing tests from T_{B}. The computed subset of T_{B} is denoted by T_{R}. To provide sufficient tests for the second phase of test selection, the procedure restricts the test removal by setting a parameter $BOUND$ indicating the minimum number of tests in T_{R}. Initially, we assign $T_{\mathrm{R}} = T_{\mathrm{B}}$. The procedure then iteratively removes tests from T_{R}, as follows.

At the beginning of each iteration, the procedure randomly divides T_{R} into l subsets, denoted by T_{r0}, $T_{\mathrm{r1}}, ...,$ $T_{\mathrm{r(l-1)}}$, each of them containing m tests. To speed up the procedure, l and m are parameters that change according to $|T_{\mathrm{R}}|$. In general, the procedure removes these subsets from T_{B} one at a time. For every subset T_{ri}, it calls the defect diagnosis procedure $Diag(T_{\mathrm{R}} - T_{\mathrm{ri}})$ to check the effects of the removal of T_{ri} on the candidate fault set. The removal is accepted only if this candidate fault set contains C_{B}. This implies that the tests in $T_{\mathrm{R}} - T_{\mathrm{ri}}$ are sufficient to obtain C_{B}. When the removal of T_{ri} from T_{R} is accepted, the removal of additional tests is considered with T_{ri} excluded from T_{R}.

The experimental results indicate that, as the procedure removes more tests, lower numbers of candidate faults are obtained. This may result in none of these removals being accepted during one iteration. When this occurs, the procedure switches to the following strategy. It selects the subset T_{ri} that produces the largest intersection between the obtained candidate fault set and C_{B} of all the subsets. It then removes the tests in T_{ri} from T_{R} one at a time. For every test t, it calls the defect diagnosis procedure $Diag(T_{\mathrm{R}} - t)$ to obtain the output candidate fault set. We use the same criteria described earlier to determine whether the removal of t is acceptable. Similarly, when the removal of t is accepted, T_{R} is updated by removing t from it, and additional tests are considered for removal from the reduced test set.

The procedure described above is shown next. The procedure terminates when the removal of any test from T_{ri} is not acceptable. In addition, since the maximum number of tests removed from T_{R} in one iteration is m, the procedure terminates when $|T_{\mathrm{R}}| < BOUND + m$ to guarantee that the number of tests in T_{R} is no less than *BOUND*.

3.4 Procedure Based on Test Selection

In this part, we describe the test selection procedure, which is based on a subset-by-subset strategy. A partition of the candidate faults into subsets is provided by the defect diagnosis tool $Diag()$. We assume that C_{B} contains n subsets of candidate faults, denoted by $C_{\mathrm{B0}}, C_{\mathrm{B1}}, ..., C_{\mathrm{B(n-1)}}$. During test selection, we ensure that every set C_{A} of candidate faults is also partitioned into n subsets. The subsets

Procedure 3.1 Test Removal

```
1: Assign $T_{\rm R} = T_{\rm B}$.
2: while $T_{\rm R}$ is not empty and $|T_{\rm R}| \geq BOUND + m$ do
3:    Randomly divide $T_{\rm R}$ into $l$ subsets $T_{\rm r0}, T_{\rm r1}, ..., T_{\rm r(l-1)}$, each of them containing
      $m$ tests.
4:    for every subset $T_{\rm r1}$ do
5:       Call $Diag(T_{\rm R} - T_{\rm ri})$ and compute the intersection between the obtained can-
         didate fault set and $C_{\rm B}$.
6:       If the intersection is $C_{\rm B}$, then $T_{\rm R} = T_{\rm R} - T_{\rm ri}$.
7:    end for
8:    if at least one subset is removed from $T_{\rm R}$ then
9:       go to step 2.
10:   end if
11:   Select the subset $T_{\rm ri}$ that produces the largest intersection between the obtained
      candidate fault set and $C_{\rm B}$ of all the subsets.
12:   for every test pattern $t$ in $T_{\rm ri}$ do
13:      Call $Diag(T_{\rm R} - t)$ and compute the intersection between the obtained candi-
         date fault set and $C_{\rm B}$.
14:      If the intersection is $C_{\rm B}$, then $T_{\rm R} = T_{\rm R} - t$.
15:   end for
16:   If no test is removed from $T_{\rm R}$, then stop.
17: end while
```

of $C_{\rm A}$ are denoted by $C_{\rm A0}, C_{\rm A1}, ..., C_{\rm A(n-1)}$. Initially, we assign $C_{\rm A0} = C_{\rm B0}, C_{\rm A1} = C_{\rm B1}, ..., C_{\rm A(n-1)} = C_{\rm B(n-1)}$. For every subset of candidate faults $C_{\rm Aj}$, whenever a smaller candidate fault set is obtained during test selection, $C_{\rm Aj}$ is replaced with the smaller candidate fault set. The test selection procedure considers the subsets of candidate faults one at a time in a descending order of the number of candidate faults in $C_{\rm B0}, C_{\rm B1}, ..., C_{\rm B(n-1)}$. When $C_{\rm Aj}$ is considered, the test selection procedure performs the following steps. This procedure repeats for every subset of candidate faults.

The reduced test set is denoted by $T_{\rm A}$. Initially, when $C_{\rm Aj}$ is considered, $T_{\rm A}$ is empty. We also denote $T_{\rm C} = T_{\rm R} - T_{\rm A}$. Tests from $T_{\rm C}$ will be added to $T_{\rm A}$.

As in the test removal phase, we consider subsets of $T_{\rm C}$. In each iteration, we first randomly divide $T_{\rm C}$ into l subsets, denoted by $T_{\rm a0}$, $T_{\rm a1}$, ..., $T_{\rm a(l-1)}$, each of them containing m tests. Again, l and m are parameters that change according to $|T_{\rm C}|$. The procedure calls the defect diagnosis tool $Diag(T_{\rm A} + T_{\rm ai})$ for every subset $T_{\rm ai}$. Of all the options for T_{ai}, the procedure selects the one that yields the candidate fault set with the smallest number of candidate faults. The procedure then adds T_{ai} to T_{A}. This repeats until at least one of the termination conditions given below is satisfied.

As shown in [19], the number of candidate faults increases as the procedure selects more tests. Accordingly, we use a similar strategy to speed up this procedure. The procedure maintains a target number of candidate faults for C_{Aj}, which is denoted by N_{C}. Initially, $N_{\mathrm{C}} = 1$, since a smaller set cannot be produced. After selecting T_{ai}, the procedure assigns N_{C} equal to the number of candidates in the intersection between the obtained candidate fault set and C_{B}. If adding an additional subset in the next iteration leads to at most N_{C} candidate faults, the procedure selects the subset without considering additional tests.

The procedure adds a subset of T_{C} into T_{A} in each iteration. In many cases, especially when m is large, removing some of the tests added to T_{A} may improve the resolution of diagnosis further. The procedure accomplishes this by reconsidering every test t in T_{ai} after the selection of T_{ai} is accepted. For each test t, we call the defect diagnosis tool $Diag(T_{\mathrm{A}} - t)$ to check the effects of the removal of t. The procedure accepts the removal when the number of computed candidate faults is less than or equal to N_{C}. However, if all the preceding tests in T_{ai} are removed from T_{A}, the procedure will not remove the last test in T_{ai} from T_{A}. The procedure then assigns N_{C} equal to the number of computed candidate faults. When the removal of t is accepted, the removal of additional tests is considered with t excluded from T_{A}.

The consideration of C_{Aj} terminates if at least one of the following conditions is satisfied. (1) The total number of candidate faults in $C_{\mathrm{A0}}, C_{\mathrm{A1}}, ..., C_{\mathrm{A(n-1)}}$ does not exceed 20, which is assumed to be a feasible number for physical failure analysis. (2) All the candidate faults in C_{Aj} are equivalent faults, which are identified by the defect diagnosis tool.

The test selection procedure applying the subset-by-subset strategy described above is shown next. For conciseness, we denote the intersection between the current candidate fault set the tool returns and C_{B} by C_{temp}. The procedure terminates when all the subsets of candidate faults satisfy the termination conditions described above.

Procedure 3.2 Test Selection

```
1: for $j = 0, 1, ..., n-1$, considering the subsets of candidate faults in descending
   order of the number of candidate faults in $C_{\mathrm{Bj}}$ do
2:     $T_{\mathrm{A}} = 0; N_{\mathrm{C}} = 1$.
3:     Randomly divide $T_{\mathrm{C}} = T_{\mathrm{R}} - T_{\mathrm{A}}$ into $l$ subsets $T_{\mathrm{a0}}, T_{\mathrm{a1}}, ..., T_{\mathrm{a(l-1)}}$, each of them
       containing $m$ tests.
4:     for every subset $T_{\mathrm{ai}}$ do
5:         Call $Diag(T_{\mathrm{A}} + T_{\mathrm{ai}})$ and compute $C_{\mathrm{temp}}$.
6:         If $|C_{\mathrm{temp}}| \leq N_{\mathrm{C}}$, then add $T_{\mathrm{ai}}$ into $T_{\mathrm{A}}$, assign $N_{\mathrm{C}} = |C_{\mathrm{temp}}|$, and go to step
           9.
7:     end for
8:     Select the subset $T_{\mathrm{ai}}$ for which $C_{\mathrm{temp}}$ is the smallest of all the subsets. Assign
       $N_{\mathrm{C}} = |C_{\mathrm{temp}}|$.
9:     Replace every subset of candidate faults with a smaller candidate fault set, if
       available.
10:    for each test $t$ in $T_{\mathrm{ai}}$, if $t$ is not the last test in $T_{\mathrm{ai}}$ that is not removed from $T_{\mathrm{A}}$
       do
11:        Call $Diag(T_{\mathrm{A}} - t)$ and compute $C_{\mathrm{temp}}$.
12:        If $|C_{\mathrm{temp}}| \leq N_{\mathrm{C}}$, then remove $t$ from $T_{\mathrm{A}}$, assign $N_{\mathrm{C}} = |C_{\mathrm{temp}}|$, and replace
           every subset of candidate faults with a smaller candidate fault set, if available.
13:    end for
14:    If $T_{\mathrm{A}} \neq T_{\mathrm{R}}$, then go to step 3.
15: end for
```

3.5 Experimental Results

To demonstrate the effectiveness of the proposed procedure, we apply it to three different groups of benchmark circuits, ISCAS-89, ITC-99 and OpenCores®. The defect diagnosis procedure *Diag*() used in this chapter is a commercial defect diagnosis tool. We run the procedure on a Linux machine with 2.6GHz processors.

Faulty output responses for defect diagnosis are produced by injecting multiple stuck-at faults into the circuit, and computing the output responses. We denote a multiple fault as f_{in}. A fault f_{in} of multiplicity k consists of k single stuck-at faults. To cover a range of possible multiplicities, we use $2 \leq k \leq 10$. The test set T is a fault detection test set for single stuck-at faults that is also generated by a commercial tool.

We randomly select three different multiplicities k of f_{in} for each benchmark circuit. For each k, we run 100 cases of basic defect diagnosis with different f_{in}. We only show the case with f_{in} that results in the largest basic set of candidate faults C_{B} of the 100 cases for each k. In all the cases, $|C_{\text{B}}| > 20$. These cases demonstrate the effectiveness of the procedure for large sets of candidates, which are its targets. As described in Section 3.1, large sets of candidate faults are typical when multiple defects are present in faulty chips.

We experimented with different values of $BOUND$, which is the minimum number of tests in T_{R}. Both the computational effort and the effectiveness of test selection increase with $BOUND$. Experimental results indicate that $BOUND = 20$ balances the two parameters well.

We select the values of l and m as follows. The computational effort of all the calls to *Diag*() is approximately the same. Therefore, the goal of the selection of l and m is to minimize the number of calls to *Diag*(). For a test set T, when $m \neq 1$, in the worst case, an arbitrary iteration requires $l + m$ calls to *Diag*(), where $l = |T|/m$. To minimize the number of calls to *Diag*(), $m = l = \sqrt{|T|}$. When $m = 1$, in the worst case, an arbitrary iteration requires $|T|$ calls to *Diag*(). The procedure only accepts $m = 1$ when $|T| < 2$. This indicates that $|T| < 4$. To summarize, the selection of l and l is as follows. (1) When $|T| < 4$, $m = 1$, and $l = |T|$. (2) When $|T| \geq 4$, $m = l = \sqrt{|T|}$.

We present experimental results for the cases where the coverage and precision are not affected by the proposed procedure in Table 3.1. We also present results for the cases where the coverage or precision are affected in Table 3.2.

Table 3.1 and Table 3.2 are arranged as follows. In each case, column k provides the multiplicity of f_{in}. Column *Cnd* provides the number of candidate faults. Column *Ovlp* provides the size of the overlap. Column *Sub* provides the number of subsets of candidate faults in C_{B}, which is also the number of subsets of candidate faults after applying the procedure described in this chapter. Column *Cov* provides the coverage, which is $Ovlp/k$. Column *Prc* provides the precision, which is $Ovlp/Sub$. Column *Reso* provides the resolution, which is $Ovlp/Cnd$. Column *Ratio* provides the ratio $|C_{\mathrm{RA}}|/|C_{\mathrm{B}}|$, where C_{RA} is the candidate fault set obtained by the procedure described in this chapter. In every column, two sub-columns are provided if the results obtained by the procedure described in this chapter and the ones obtained by the basic defect diagnosis procedure are different. The first sub-column provides the results of the procedure described in this chapter, while the second sub-column provides the results of the basic defect diagnosis procedure. Otherwise, the single entry applies to both defect diagnosis procedures.

Column *ntime* provides the run time for the procedure described in this chapter relative to the run time for the basic defect diagnosis procedure. In Column *Tests*, the first sub-column provides the number of tests in T_{R}, while the second sub-column provides the number of tests in T_{B}.

From Table 3.1 and Table 3.2 it can be seen that the procedure described in this chapter reduces the number of candidate faults in all the cases considered. This can be seen from column *Ratio*. For most of the observed responses considered, the reduction in the number of candidate faults is significant. For some of the cases, the candidate faults with the highest scores are removed, while the ones with lower scores that are present in the circuit are kept in the candidate fault set.

Because the procedure calls Procedure 3.2 for every subset of candidate faults in C_{B} independently, it never loses a subset of candidate faults. Consequently, the coverage and precision rarely reduce, while the resolution improves significantly. The procedure described in this chapter only loses coverage or precision in eight cases out of forty-two cases that are considered. For every case that loses coverage or precision, only one candidate fault that is present in the faulty circuit is missed by the procedure described in this chapter. The reduction in the number of candidate faults is significant in all these cases, which are reported in Table 3.2, and justifies the loss in coverage and precision. For example, reducing the number of candidate faults

Table 3.1.: Experimental Results (Same Coverage and Precision)

Circuit	**k**	**Cnd**		**Ovlp**	**Sub**	**Cov**	**Prc**	**Reso**		**Ratio**	**ntime**	**Tests**	
b04	5	15	57	3	4	60.00%	75.00%	20.00%	5.26%	26.32%	210.2	40	79
	8	24	92	4	5	50.00%	80.00%	16.67%	4.35%	26.09%	141.2	66	79
	9	29	65	3	4	33.33%	75.00%	10.34%	4.62%	44.62%	60	23	79
b14	2	2	24	2	2	100.00%	100.00%	100.00%	8.33%	8.33%	93.29	21	429
	4	15	95	3	4	75.00%	75.00%	20.00%	3.16%	15.79%	166.57	65	429
	8	46	104	6	7	75.00%	85.71%	13.04%	5.77%	44.23%	349.5	65	429
b17	2	22	45	2	2	100.00%	100.00%	9.09%	4.44%	48.89%	71.44	22	589
	3	20	45	3	3	100.00%	100.00%	15.00%	6.67%	44.44%	122.08	21	589
	6	30	54	5	5	83.33%	100.00%	16.67%	9.26%	55.56%	142.53	21	589
b18	2	8	32	2	2	100.00%	100.00%	25.00%	6.25%	25.00%	49.57	21	658
	3	33	56	3	3	100.00%	100.00%	9.09%	5.36%	58.93%	104.21	22	658
b19	3	21	39	3	3	100.00%	100.00%	14.29%	7.69%	53.85%	59.56	20	659
	5	45	75	5	5	100.00%	100.00%	11.11%	6.67%	60.00%	50.89	22	659
b20	2	22	31	2	2	100.00%	100.00%	9.09%	6.45%	70.96%	112.45	20	468
	5	19	120	4	4	80.00%	100.00%	21.05%	3.33%	15.83%	380.3	157	468
	8	17	117	3	4	37.50%	75.00%	17.65%	2.56%	14.53%	116	22	468
s1423	2	10	35	2	2	100.00%	100.00%	20.00%	5.71%	28.57%	54.5	21	73
	5	23	50	2	4	40.00%	50.00%	8.70%	4.00%	46.00%	153.13	23	73
s35932	4	16	44	4	4	100.00%	100.00%	25.00%	9.09%	36.36%	199.5	26	50
s38584	3	31	46	3	3	100.00%	100.00%	9.68%	6.52%	67.39%	78.6	21	145
	5	34	53	5	5	100.00%	100.00%	14.71%	9.43%	64.15%	82.1	20	145
	9	47	69	5	6	55.57%	83.33%	10.64%	7.25%	68.12%	484.29	36	145
aes_core	4	47	168	4	4	100.00%	100.00%	8.51%	2.38%	27.98%	91.36	26	380
	6	85	167	5	5	83.33%	100.00%	5.89%	2.99%	50.89%	175.69	54	380
des_perf	3	27	61	3	3	100.00%	100.00%	11.11%	4.92%	44.26%	252.44	37	137
	5	18	68	4	4	80.00%	100.00%	22.22%	5.88%	26.47%	280.41	31	137
	7	59	83	7	7	100.00%	100.00%	11.86%	8.43%	71.08%	349.18	37	137
i2c	2	9	31	2	2	100.00%	100.00%	22.22%	6.45%	29.03%	21.88	21	73
	3	26	49	3	3	100.00%	100.00%	11.54%	6.12%	53.06%	177.2	21	73
	6	50	81	4	5	66.67%	80.00%	8.00%	4.94%	61.73%	260.42	37	73
spi	4	7	104	3	3	75.00%	100.00%	42.86%	2.88%	6.73%	571.21	183	495
systemcaes	2	5	68	2	2	100.00%	100.00%	40.00%	2.94%	7.35%	203.4	21	205
	4	14	48	3	4	75.00%	75.00%	21.43%	6.25%	29.17%	430.6	23	205
	5	57	84	4	5	80.00%	80.00%	7.02%	4.76%	67.86%	249.8	35	205

Table 3.2.: Experimental Results (Different Coverage or Precision)

Circuit	**k**	**Cnd**		**Ovlp**		**Sub**	**Cov**		**Prc**		**Reso**		**Ratio**	**ntime**	**Tests**	
b18	8	29	151	3	4	5	37.50%	50.00%	60.00%	80.00%	10.34%	2.65%	19.21%	132.25	33	658
b19	10	35	112	4	5	7	40.00%	50.00%	57.14%	71.43%	11.43%	4.46%	31.25%	193.82	23	659
s1423	7	31	100	5	6	7	71.43%	85.71%	71.43%	85.71%	16.13%	6.00%	31.00%	472.3	55	73
s35932	9	12	90	2	3	3	22.22%	44.44%	66.67%	100.00%	16.67%	4.44%	13.33%	63.6	20	50
	10	18	128	4	5	5	40.00%	50.00%	80.00%	100.00%	22.22%	3.91%	14.06%	138	44	50
aes_core	10	90	244	7	8	9	70.00%	80.00%	77.78%	88.89%	7.77%	3.28%	36.89%	521.07	73	380
spi	5	22	65	4	5	5	80.00%	100.00%	80.00%	100.00%	18.18%	7.69%	33.85%	212.33	26	495
	6	36	82	4	5	5	66.67%	83.33%	80.00%	100.00%	11.11%	6.10%	43.91%	291.87	23	495

from 151 to 29 for b18 with $k = 8$ makes it possible to apply physical failure analysis in a case where it cannot be applied with the basic defect diagnosis procedure.

To further illustrate the improvements of the defect diagnosis results using the proposed procedure, Table 3.3 presents a summary of the results of all the cases that are considered. Column *Cnd* provides the total number of candidate faults obtained for all the cases considered. Column *Ovlp* provides the total size of the overlap. Row *Basic* provides the results of the basic defect diagnosis procedure. Row *RA* provides the results of the proposed procedure. Row *Reduction* provides the reduction in the sizes of the candidate fault set and the overlap when the proposed procedure, instead of the basic defect diagnosis procedure, is used. Row *Percent* provides the percentage reduction. It can be observed from Table 3.3 that in total, more than 60% of the candidate faults are removed from C_{B}, while the loss of the overlap is only about 5%. Consequently, the loss of coverage and precision is small, while the increase in resolution is significant.

Table 3.3.: Statistical Analysis

	Cnd	**Ovlp**
Basic	3332	159
RA	1206	151
Reduction	2126	8
Percent	63.81%	5.03%

It can also be seen from column *Tests* that T_{R} typically contains significantly fewer tests than T. This indicates that most of the tests are removed during the test removal phase. Experimental results show that the number of tests in T_{A} for each subset of candidate faults is also small. For instance, the candidate fault set C_{B} in the first case considered for s1423 contains two subsets of candidate faults. The test set T_{A} for each subset of candidate faults contains one and five tests, respectively.

As in [17–19], after the computation of C_{B}, the run time of the procedure described in this chapter can be significantly reduced by providing the defect diagnosis procedure with C_{B} from which to select the candidate faults. However, in our study, we prefer to avoid interfering with the internal operation of the defect diagnosis tool. Therefore, the run time of the procedure described in this chapter is higher than necessary.

4. LOGIC RESYNTHESIS FOR AVOIDING UNDETECTABLE FAULTS BASED ON DFM GUIDELINES IN A CELL-BASED DESIGN

As integrated circuit manufacturing advances, the occurrence of systematic defects is expected to be prominent. A methodology for modeling systematic defects based on DFM guidelines was described earlier. In this chapter we demonstrate that, among the faults obtained based on DFM guidelines, there are undetectable faults, and these faults cluster in certain areas of the circuit. This leaves areas in the circuit uncovered for potential systematic defects. Because the defects are systematic, and they may be detectable even though the faults are undetectable, the test escapes can impact the yield and DPPM significantly. To address this issue, we propose two procedures based on logic and layout resynthesis to eliminate large clusters of undetectable faults related to DFM guidelines, and improve the coverage of the circuit for potential systematic defects. In this chapter, the logic resynthesis procedure is presented. The layout resynthesis procedure is presented in next chapter. The logic resynthesis procedure is applied to benchmark circuits and logic blocks of the OpenSPARC T1 microprocessor. The resynthesized circuit maintains design constraints of critical path delay, power consumption and die area. Experimental results indicate that both the improvement in the coverage of the circuit and the reduction in the sizes of large clusters are significant after applying the proposed logic resynthesis procedure.

4.1 Introduction

Aggressive scaling of IC technologies continues to decrease device size and increase circuit complexity. The continuous shrinking of device sizes increases the gap between the feature size and the lithography wavelength. As a result, certain layout features are more difficult to manufacture than others, and are more likely to lead to defects. Such features can cause repeated or systematic defects to occur when they are present multiple times [20–25]. Because of the systematic nature of these defects, they can impact the yield and DPPM significantly. To address systematic defects, appropriate design interventions are inevitable so as to remedy the potential manufacturing issues.

However, the constraints of die area, layout geometry and the ever-decreasing window of time to market make it impossible to obtain complete information about the potential manufacturing issues in advance. Thus, it is not possible to eliminate all the causes of systematic defects when implementing the physical design. DFM guidelines are layout guidelines that attempt to capture and prevent yield and manufacturability issues. In contrast to design rules, which must be followed by the physical design process, DFM guidelines are applied when possible to improve the yield. The relationship between DFM guidelines and potential defects was noted in [27]. In [27], DFM guidelines were tightened to find the layout locations where DFM guidelines were not applied, or not applied strictly, so as to anticipate potential causes for systematic defects. The affected transistors were identified at the schematic level, and the defect behaviors were translated to gate-level logic faults by using switch-level simulation. A target test set for these logic faults was then generated to close potential test holes and prevent adverse DPPM impact due to systematic defects.

Not all the faults that result from DFM guidelines are detectable. When a translated logic fault is undetectable, it leaves an uncovered site in the circuit. It was shown in [62] and [63] that undetectable faults in general tend to cluster in certain areas, leaving areas of the circuit uncovered. This can lead to test holes that affect more than a single gate or line. If an area of the circuit is uncovered, detectable defects in the area may go undetected, resulting in test escapes. As we demonstrate in Section 4.2, undetectable logic faults that result from DFM guidelines also tend to cluster in certain areas of the circuit. Because these logic faults are likely to be systematic, the test escapes caused by the clustering phenomenon can impact the yield and DPPM significantly.

The solution suggested in [62] and [63] is to target double faults that consist of an undetectable fault and an adjacent detectable fault. Additional tests for double faults were generated so as to improve the coverage of subcircuits containing undetectable faults. For the systematic defects considered in this chapter, the coverage of the circuit needs to be even higher so as to avoid adverse DPPM impact. In addition, experimental results show that the sizes of the clusters of undetectable faults resulting from DFM guidelines are large. Thus, a significant number of additional test patterns is needed so as to achieve an acceptable coverage for theses defects. This may result in an excessive increase in the size of the test set, which leads to an unacceptable tester time.

Motivated by these observations, we propose a procedure that is based on resynthesis to eliminate large clusters of undetectable faults related to DFM guidelines, and improve the coverage of the circuit for potential systematic defects. Resynthesis techniques are usually applied for optimizing the circuit with respect to delay, power and area [64–66]. They are also used for improving the testability of the circuit by reducing the difficulty of test generation [67–72]. The resynthesis procedure described in this chapter is the first one to address clusters of potential systematic defects related to DFM guidelines by resynthesis. A test set for the detectable faults related to DFM guidelines in the resynthesized circuit is also obtained during the proposed resynthesis procedure. Test generation procedures, such as the ones described in [62] and [63], can follow the procedure described in this chapter so as to improve the coverage of the circuit further by generating additional tests based on the remaining undetectable faults. In our experiments, typically one tenth of the original undetectable faults remain undetectable after resynthesis. With significantly fewer undetectable faults, the effect of additional test generation on test set size is reduced.

The resynthesis procedure is developed for a standard cell based design flow. In a cell based design flow, a gate-level netlist and a layout are synthesized from an RTL description of a circuit using a standard cell library. Typically, several iterations of the design process are needed to satisfy design constraints such as area, delay and power. The proposed resynthesis procedure is also iterative, and it can fit within the overall iterative design flow. Specifically, an iteration of the design process can include one or more iterations of the resynthesis procedure to eliminate clusters of undetectable faults.

In this context of cell-based design, we distinguish between faults that are internal to the standard cells (internal faults), and faults that are external to the standard cells (external faults). Every time a gate, or an instance of a standard cell, is used in the circuit, it introduces the same internal faults. The procedure eliminates the undetectable internal faults by resynthesizing the circuit with standard cells containing fewer internal faults. As we demonstrate in Section 4.2, most of the undetectable faults are internal faults. Therefore, the increase in the coverage of the circuit and the decrease in the sizes of large clusters of undetectable faults are significant when undetectable internal faults are eliminated by resynthesis.

The proposed procedure has two phases. The first phase focuses on the largest clusters of undetectable faults so as to reduce the number of undetectable faults that are clustered together. The second phase focuses on the entire circuit so as to improve the coverage of the circuit further. In both phases, the procedure resynthesizes the circuit iteratively. For a large chip, the procedure can be applied to every logic block separately so as to keep the computational effort acceptable. The proposed procedure is applied to logic blocks of the OpenSPARC T1 microprocessor to demonstrate its applicability to such designs.

To guarantee that the proposed procedure maintains design constraints of critical path delay, power consumption and die area, we also develop a backtracking procedure based on the observation that modifying fewer gates implies lower design overheads. The backtracking procedure reduces the design overheads by modifying fewer gates during both phases. In addition, the design overheads are also reduced when gates with fewer internal faults, which are typically smaller, replace larger gates that can cause routing congestion, and thus affect delay and performance adversely.

In this chapter, the die area after resynthesis is kept the same as the original design so as to maintain the original floorplan of the chip. For delay and power, a maximum of five percent increase compared with the original design is assumed to be acceptable. Experimental results indicate that the coverage of the circuit can be improved significantly under these design constraints. The resynthesis procedure can accommodate different design constraints if needed.

4.2 Background

We use the approach described in [27] to translate the violations of DFM guidelines into likely systematic defects, and related logic faults. The violations of DFM guidelines are first translated into likely shorts and opens inside and outside cells. In some cases, the defects are translated into faults that belong to commonly used fault models [73–75]. In other cases, switch-level simulation is carried out, and a fault is represented by input and output patterns of a cell [76]. We denote the resultant fault set by F.

A test generation procedure is applied to F. We denote by T a test set that detects all the detectable faults in F. The fault set $U = \{f_0, f_1, ..., f_{l-1}\}$ consists of all the undetectable faults in F.

We say that a gate *corresponds to a fault* f_i if (1) f_i is an internal fault, and it is inside the gate, or (2) f_i is an external fault, and it is on the inputs or outputs of the gate. Only one gate corresponds to an internal fault. For an external fault, multiple gates may correspond to the fault when it is located on a net that connects multiple gates, or it results in a short between two nets.

To explore the structural relations among the gates corresponding to the undetectable faults, we say that two gates are structurally adjacent if one of the two gates is directly driven by the other gate. For illustration, in Fig. 4.1, gates g1 and g2 are only adjacent in (c). We also define two faults f_{a} and f_{b} to be structurally adjacent if one of the following conditions is satisfied. (1) There exist a gate that corresponds to f_{a} and a gate that corresponds to f_{b}, such that the two gates are adjacent. (2) There exists a gate that corresponds to both f_{a} and f_{b}.

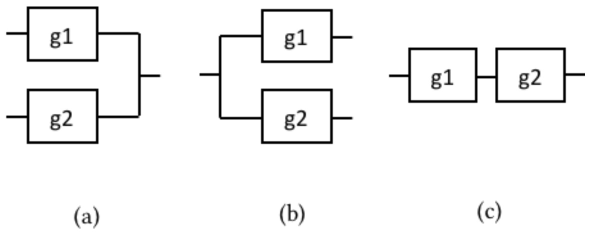

Fig. 4.1.: Adjacent Gates.

We partition U into subsets $S_0, S_1, ...$ of adjacent faults. Initially, we set $S_{\mathrm{i}} = \{f_{\mathrm{i}}\}$ for $0 \leq i < l$. We repeat the following process to merge subsets that contain adjacent faults. The process terminates when additional merging is not possible.

For every pair of subsets of undetectable faults, S_{i1} and S_{i2} such that $i2 > i1$, we check whether S_{i1} and S_{i2} contain faults f_{i1} and f_{i2}, respectively, where f_{i1} and f_{i2} are structurally adjacent. If so, we merge S_{i1} and S_{i2} by adding the faults from S_{i1} to S_{i2}, and removing S_{i1}.

To demonstrate that the undetectable faults resulting from DFM guidelines and the gates corresponding to them tend to cluster in certain areas of the circuit, we computed the subsets of undetectable faults and the gates corresponding to them for ITC-99 benchmark circuits and logic blocks of the OpenSPARC T1 microprocessor. We denote the largest subset of adjacent undetectable faults by S_{max}. The set of gates corresponding to all the faults in S_{max} is denoted by G_{max}. The results are shown in Table 4.1. Column F shows the total number of faults resulting from DFM guidelines. Column U shows the number of undetectable faults. Column G shows the number of gates that correspond to all the undetectable faults. Column G_{max} shows the number of gates in G_{max}. Column S_{max} shows the number of undetectable faults in S_{max}. Column *%Smax_U* shows the percentage of all the undetectable faults that are in S_{max}. Column *Smax_I* shows the number of undetectable internal faults in S_{max}. Column *%Smax_I* shows the percentage of all the faults in S_{max} that are internal faults.

Table 4.1.: Clustered undetectable faults

Circuit	***F***	***U***	***G***	***Gmax***	***Smax***	***%Smax _U***	***Smax _I***	***%Smax _I***
b15	34188	3108	1345	1186	2572	82.75%	2152	83.67%
b20	38337	2552	857	645	1638	64.18%	1472	89.87%
sparc_fpu	234125	15263	4685	2831	8291	54.32%	7515	90.64%
sparc_exu	116525	10753	3661	2771	7072	65.77%	6338	89.62%

From Table 4.1, it can be observed that the circuits have large subsets of adjacent undetectable faults, and large sets of adjacent gates corresponding to them. Although the faults are undetectable, defects in the same areas of the circuit may be detectable, and they will go undetected if the areas are not covered. In addition, the major part of the undetectable faults are internal faults. Such faults can potentially be removed by replacing the gates corresponding to them. These observations motivate us to reduce the sizes of subsets of adjacent undetectable faults and improve the coverage of the circuit by resynthesizing subcircuits that consist of gates corresponding to undetectable faults.

4.3 Resynthesis Procedure

In this section, we describe an iterative resynthesis procedure that eliminates undetectable faults so as to avoid clustering that can lead to poor coverage of certain areas of the circuit. Suppose that the standard cell library consists of m standard cells, denoted by $cell_0, cell_1, ..., cell_{m-1}$. The circuit that is considered by the procedure is denoted by C_{all}. We assume that C_{all} was already optimized by one or more iterations of a standard IC design flow. The resynthesis procedure ensures that C_{all} does not deteriorate in terms of design constraints of delay, power and area, but improves in terms of the clustering of undetectable faults.

The procedure considers different subcircuits in different iterations. The subcircuit as part of C_{all} that is considered for resynthesis in an arbitrary iteration is denoted by C_{sub}. The rest of the circuit, $C_{\text{dont}} = C_{\text{all}} - C_{\text{sub}}$, is not resynthesized. We use G_{zero} to store the gates in C_{sub} that do not contain undetectable internal faults. The gates in G_{zero} are not modified during the resynthesis to avoid unnecessary design changes. In every iteration, logic synthesis is applied to C_{all}, allowing only the gates in $C_{\text{sub}} - G_{\text{zero}}$ to be modified. We denote the logic synthesis tool used in this chapter by *Synthesize*(). The physical design process that follows logic synthesis is denoted by *PDesign*(). The resynthesis procedure reduces the number of undetectable faults by eliminating undetectable internal faults. In addition, the internal faults are only related to the standard cells that are used in the circuit, and do not depend on the placement and routing processes of physical design. Therefore, *PDesign*() is called only when the number of undetectable internal faults decreases in the resynthesized circuit. This avoids unnecessary runtime for physical design. The resynthesis procedure calls *Synthesize*() and *PDesign*() iteratively as described next. The resynthesis repeats until the termination conditions described later in this section are satisfied.

For a subcircuit C_{sub}, the standard cells in the library are considered in the reverse order of the number of internal faults. This is because standard cells with more internal faults tend to introduce more undetectable internal faults to the circuit. Let $cell_0, cell_1, ..., cell_{\text{m}-1}$ be the ordered set of standard cells. For a standard cell $cell_{\text{i}}$, it is eligible to be considered by the procedure when (1) it is used to synthesize C_{sub}, and (2) at least one gate in C_{sub}, of type $cell_{\text{i}}$, contains undetectable internal faults. When $cell_{\text{i}}$ is considered, the procedure resynthesizes C_{sub} without using $cell_0, cell_1, ..., cell_{\text{m}-1}$. This is important to avoid introducing gates with more internal faults. The procedure then calls *Synthesize*(C_{all}), and the resultant circuit is denoted by C_{temp}. A call to *PDesign*(C_{temp}) is carried out if the number of undetectable internal faults decreases. Otherwise, the procedure moves on to consider the next standard cell. After calling *PDesign*(C_{temp}), if the acceptance criteria described later are satisfied, yet the resultant layout violates the design constraints, a backtracking procedure is invoked, denoted by *Backtracking*(). Details of *Backtracking*() will be discussed in Section 4.4.

As described in Section 4.1, the procedure has two phases. During phase one, the procedure targets the largest cluster, which is described by the set of faults $S_{\max}$ and the set of gates $G_{\max}$. In this case, C_{sub} consists of all the gates in $G_{\max}$ and the goal is to reduce the size of $S_{\max}$ without introducing more undetectable faults to the circuit. The resynthesized circuit C_{temp} is accepted in the first phase of the procedure if the following criteria are satisfied. (1) The size of the previous largest subset of adjacent undetectable faults decreases. (2) No more undetectable faults are introduced to the circuit. (3) The design constraints of delay, power and area are satisfied. We use a target p_1 for the percentage of the faults in F that are in $S_{\max}$ after phase one. The first phase of the procedure terminates when (1) the percentage of the faults in F that are in $S_{\max}$ is less than or equal to p_1, or (2) no additional improvements can be achieved during phase one in terms of the number of undetectable faults in the circuit.

During phase two, C_{sub} consists of the gates that correspond to all the undetectable faults in the circuit so as to improve the coverage of the circuit further. We use p_2 to indicate the maximum acceptable percentage of the faults in F that are in $S_{\max}$ after the second phase of the procedure. If the percentage of the faults in F that are in $S_{\max}$ after phase one is less than p_1, then p_2 is set to be p_1. Otherwise, p_2 is set to be the percentage of the faults in F that are in $S_{\max}$ after phase one. The resynthesized circuit C_{temp} is accepted during the second phase of the procedure if the following criteria are satisfied. (1) The total number of undetectable faults in the circuit decreases. (2) The percentage of faults in F that are in $S_{\max}$ does not exceed p_2. (3) The design constraints of delay, power and area are satisfied. The second phase terminates when the number of undetectable faults in the circuit cannot be decreased further.

To speed up the procedure, additional conditions are used to terminate the current phase based on the following observation. In an arbitrary iteration, the procedure eliminates undetectable internal faults in C_{sub} by resynthesizing the circuit with standard cells that contain fewer internal faults. This can potentially increase the number of undetectable external faults, since the nets inside the original gates may be outside the gates in the resynthesized circuit. Therefore, as the standard cells are considered, the gross trend of the number of undetectable faults in the circuit first goes down and then up. We terminate a phase when it appears that the number of undetectable faults is increasing, as follows. Let u_{min} be the minimum number of undetectable faults in the original or any resynthesized circuit that was previously accepted. Let u_{i} be the number of undetectable faults in the circuit when $cell_{\mathrm{i}}$ is considered. The current phase of the procedure terminates if the following conditions are satisfied. (1) $u_{\mathrm{i-1}} > u_{\mathrm{min}}$ and $u_{\mathrm{i}} > u_{\mathrm{i-1}}$, or (2) $u_{\mathrm{i-1}} \leq u_{\mathrm{min}}$ and $u_{\mathrm{i}} > u_{\mathrm{min}}$. In the first case, the consideration of $cell_{\mathrm{i-1}}$ leads to a number of undetectable faults that is larger than u_{min}, and even more undetectable faults are obtained when considering $cell_{\mathrm{i}}$. This implies that the consideration of $cell_{\mathrm{i+1}}, cell_{\mathrm{i+2}}, ..., cell_{\mathrm{m-1}}$ may also result in a number of undetectable faults that is larger than u_{min}. In the second case, the consideration of $cell_{\mathrm{i-1}}$ leads to a number of undetectable faults that is smaller than u_{min}, yet the resynthesized circuit is not accepted as it may violate other acceptance criteria. The current phase does not terminate as long as a number of undetectable faults that is smaller than u_{min} is obtained.

The procedure for phase one is shown next. We use *set_dont_use*() to indicate standard cells that cannot be used during the logic synthesis process. We use *set_dont_touch*() to indicate subsets of gates and subcircuits that are not considered for modification during the logic synthesis process. The procedure for phase two is similar. The main difference is that in step 1 of phase two, C_{sub} consists of all the gates in the circuit that correspond to undetectable faults. In addition, the acceptance criteria for phase two are different from the ones for phase one as described earlier.

Procedure 4.1 Phase one of resynthesis

```
1: $C_{\text{sub}}$ consists of all the gates in $G_{\text{max}}$, $C_{\text{dont}} = C_{\text{all}} - C_{\text{sub}}$
2: **for** every $cell_{\text{i}}$, $0 \le i \le m-1$, if $cell_{\text{i}}$ is eligible **do**
3:    $set\_dont\_use(cell_0, cell_1, , cell_{\text{i}})$
4:    $set\_dont\_touch(G_{\text{zero}}, C_{\text{dont}})$
5:    $C_{\text{temp}} = Synthesize(C_{\text{all}})$
6:    **if** more undetectable internal faults are in the circuit **then**
7:       **continue**
8:    **end if**
9:    $PDesign(C_{\text{temp}})$
10:   **if** $u_{\text{i}-1} > u_{\text{min}}$ and $u_{\text{i}} > u_{\text{i}-1}$, or $u_{\text{i}-1} \le u_{\text{min}}$ and $u_{\text{i}} > u_{\text{min}}$ **then**
11:      $p_2 = max\{p_1, |S_{\text{max}}|/|F|\}$
12:      stop
13:   **end if**
14:   **if** the size of the previous largest cluster decreases and no more undetectable
      faults are introduced to the circuit **then**
15:      **if** design constraints are not satisfied **then**
16:         call $Backtracking()$
17:         **if** a resynthesized circuit is accepted in $Backtracking()$ **then**
18:            go to step 1.
19:         **end if**
20:      **else**
21:         $C_{\text{all}} = C_{\text{temp}}$
22:         **If** $|S_{\text{max}}|/|F| > p_1$, **then** go to step 1.
23:         $p_2 = max\{p_1, |S_{\text{max}}|/|F|\}$
24:         stop
25:      **end if**
26:   **end if**
27: **end for**
28: $p_2 = max\{p_1, |S_{\text{max}}|/|F|\}$
29: stop
```

4.4 Backtracking Procedure

In this part, we describe the backtracking procedure we use to guarantee that the resynthesized circuit does not violate the design constraints of delay, power and area.

The backtracking procedure is called when an attempt to resynthesize the circuit fails because of the design constraints.

Suppose that the standard cell $cell_{\mathrm{i}}$ is currently considered by the resynthesis procedure. Considering the gates in C_{sub} that belong to $cell_0, cell_1, ..., cell_{\mathrm{i}}$, the backtracking procedure includes in a set G_{i} every gate that is not in G_{zero}. The backtracking procedure is based on the observation that modifying fewer gates implies lower design overheads. Therefore, instead of trying to replace all the gates in G_{i} as in phase one or two, the backtracking procedure considers subsets of G_{i}.

The procedure first removes gates from G_{i} in groups of k gates, where k is selected based on the computational complexity as described in Section 4.5. The gates that are removed from G_{i} are placed in a set G_{back}. As the backtracking procedure removes more gates from G_{i}, higher numbers of undetectable faults are obtained. This may result in a resynthesized circuit that does not violate the design constraints, yet does not satisfy the acceptance criteria for the current phase of the resynthesis procedure that are described in Section 4.3. When this occurs, the backtracking procedure returns to G_{i} the last k gates that it added to G_{back} one by one.

When removing gates from G_{i}, the backtracking procedure always considers the gates with fewer undetectable internal faults first. When adding gates back to G_{i}, it considers the gates with more undetectable internal faults first. In every case, the procedure calls $Synthesize(C_{\mathrm{all}})$ to resynthesize C_{sub} without changing the gates in G_{back}, G_{zero} and C_{dont}. As before, the resultant circuit is denoted by C_{temp}. The physical design process $PDesign(C_{\mathrm{temp}})$ is called when the number of undetectable internal faults is smaller than the one before the current iteration of the resynthesis procedure. Otherwise, the acceptance criteria for the current phase of the resynthesis procedure are not satisfied, and there is no need to perform physical design.

The resynthesized circuit C_{temp} is accepted if the acceptance criteria described in Section 4.3 are satisfied and no design constraints are violated. To speed up the proposed resynthesis procedure, the backtracking procedure terminates whenever a resynthesized circuit C_{temp} is accepted. In addition, it terminates if no more gates can be added into or removed from G_{back}.

4.5 Experimental Results

The proposed procedure is applied to ISCAS-89, ITC-99 and OpenCores® benchmark circuits, and to several logic blocks of the OpenSPARC T1 microprocessor. OpenSPARC T1 is a 64-bit open-sourced microprocessor. It has eight cores and each core can support up to four threads for a total of thirty-two threads. Within OpenSPARC T1, we apply the proposed procedure to the logic blocks in a single SPARC core and the floating-point unit (fpu).We run the procedure on a Linux machine with 2.6GHz processors.

We obtained gate-level netlists and layouts from RTL descriptions using the tool kit in the standard cell library developed by OSU [77], which is based on TSMC 0.18um technology. The netlists obtained after logic synthesis for all the circuits considered in this chapter are flattened. Each circuit is treated as one block with respect to floorplanning. The core utilization for the floorplan of the original physical design is set to be 70% for all the circuits. As described in Section I, no increase in die area is allowed in this chapter. In addition, a maximum of five percent increase in critical path delay and power consumption, compared with the original design, is assumed to be acceptable.

To define the set F of target faults, three categories of DFM guidelines from [28] are considered, *Via*, *Metal* and *Density*. The guidelines specify certain dimensions of width and spacing that are recommended for the physical design process. We use 19 guidelines in the *Via* category, 29 guidelines in the *Metal* category, and 11 guidelines in the *Density* category.

We use commercial tools for the logic synthesis and physical design processes. A commercial IC verification and sign-off package is used to find locations of potential systematic defects in the layout. In addition, we use a commercial ATPG tool to generate test patterns for fault detection.

We experimented with different values of p_1, which is the target percentage of all the faults in the circuit that are in S_{max} after the first phase of the procedure. A high value of p_1 may result in a large cluster after resynthesis, while a low value of p_1 may set a bound that is too restrictive for the second phase of the procedure to improve the coverage of the circuit. Experimental results indicate that $p_1 = 1\%$ balances them well.

We select the value of k as follows. Suppose that the computational effort of all the calls to *Synthesize*() and *PDesign*() during the backtracking procedure is approximately the same. Therefore, the goal of the selection of k is to minimize the number of calls to *Synthesize*() and *PDesign*(). Suppose that, initially, G_{i} contains n gates. We partition the calls to *Synthesize*() and *PDesign*() into two parts. In the first part, gates are added into G_{back}. The number of calls for this part is denoted by n_1. In the second part, gates are removed from G_{back}. The number of calls for this part is denoted by n_2. In the worst case, every gate from G_{i} is added to G_{back} in n/k iterations, and then all the newly added k gates are removed from G_{back} one at a time. In this case, the number of calls to *Synthesize*() and *PDesign*() is $n_1 + n_2 = n/k + k$. To minimize $n/k + k$, $k = \sqrt{n}$ is used, where the total number of calls to *Synthesize*() and *PDesign*() is 2.

The experimental results are shown in Table 4.2 as follows. In each case, two rows correspond to a circuit. The first row describes the original design of the circuit. The second row describes the resynthesized circuit obtained using the procedure described in this chapter. Row *average* shows average values considering all the circuits before and after resynthesis.

In every row, column F provides the number of faults in the circuit, which is the number of faults that are translated from the potential systematic defects based on the DFM guidelines. Column U provides the number of undetectable faults in the circuit. Column *Cov* provides the coverage of the circuit, which is defined as

$Cov = 1 - U/F\%$. Column *Smax* provides the number of undetectable faults in $S_{\max}$. Column *%Smax_all* provides the percentage of all the faults that are in $S_{\max}$. Column *Smax_I* provides the number of internal faults in $S_{\max}$. Column *%Smax_I* provides the percentage of all the faults in $S_{\max}$ that are internal faults.

In column *Delay*, we show the ratio for the critical path delay of the resynthesized circuit to the one of the original design. In column *Power*, we show the ratio for the power consumption of the resynthesized circuit to the one of the original design. In column *ntime*, we show the run time for the proposed resynthesis procedure relative to the run time for one iteration of logic synthesis and physical design with test generation for the logic faults resulting from DFM guidelines. The test generation time is included since a test set is also obtained by the proposed resynthesis procedure.

From Table 4.2 it can be seen that the procedure described in this chapter achieves a significant reduction in the number of undetectable faults for all the circuits considered. This can be seen from column U. With a small change to the total number of faults in the circuit, the improvement in the coverage of the circuit is significant.

Because the resynthesis procedure focuses on the largest cluster in the first phase so as to reduce the size of $S_{\max}$, and it is also controlled in the second phase, the size of $S_{\max}$ decreases significantly. This can be seen from column *%Smax_all*. For most of the circuits, the percentage of all the faults that are in $S_{\max}$ after applying the resynthesis procedure is below 1%, which is the target value given by p_1.

The backtracking procedure helps ensure that the improvement in the coverage of the circuit is achieved under the design constraints. This can be observed from columns *Delay* and *Power*. In addition, the layouts for all the resynthesized circuits are achieved with the original floorplans without design rule violation. For some of the circuits, the delay or power reduces compared to the original design. This is because the proposed resynthesis procedure replaces certain larger gates with smaller gates that typically contain fewer internal faults. It thus alleviates the routing congestion caused by larger gates. As a result, the adverse effects in delay and power resulting from the routing congestion are reduced.

Table 4.2.: Experimental results

Circuit	***F***	***U***	***Cov***	***Smax***	***%Smax _all***	***Smax _I***	***%Smax _I***	***Delay***	***Power***	***ntime***
b04	2940	245	91.67%	216	7.35%	139	64.35%	100.93%	103.33%	21.93
	2831	50	98.23%	23	0.81%	1	4.35%			
b14	17413	1104	93.66%	701	4.03%	629	89.73%	92.32%	101.98%	11.94
	16755	127	99.24%	98	0.58%	9	9.18%			
b15	34188	3108	90.91%	2572	7.49%	2152	83.67%	96.04%	96.27%	26.87
	32828	773	97.69%	530	1.59%	68	12.83%			
b20	38337	2552	93.34%	1638	4.27%	1472	89.87%	101.70%	103.42%	21.01
	37292	362	99.03%	311	0.83%	36	11.58%			
s9234	4475	369	91.75%	93	2.08%	82	88.17%	99.01%	101.08%	6.25
	4398	41	99.07%	10	0.23%	0	0%			
s35932	43937	1939	95.59%	1507	3.43%	1316	87.33%	101.35%	100.49%	17.99
	43401	241	99.44%	136	0.31%	0	0%			
DMA	68698	6618	90.37%	1448	2.11%	1353	93.44%	98.26%	103.13%	8.52
	67986	413	99.39%	93	0.14%	0	0%			
tv80	29376	2677	90.89%	1270	4.32%	938	73.86%	93.61%	99.15%	17.77
	28908	465	98.39%	381	1.32%	0	0%			
systemcdes	11645	943	91.90%	663	5.69%	635	95.78%	99.65%	100.75%	5.42
	11258	85	99.24%	36	0.32%	0	0%			
systemcaes	42360	4274	89.91%	2852	6.73%	2694	94.46%	95.13%	102.18%	26.37
	40480	294	99.27%	196	0.48%	1	0.51%			
aes_core	94258	6015	93.62%	1633	1.73%	1267	77.59%	94.39%	103.38%	12.78
	98368	1722	98.25%	259	0.26%	29	11.20%			
des_perf	354562	20897	94.17%	10845	3.02%	10560	97.37%	104.87%	102.11%	12.43
	363609	897	99.75%	59	0.02%	11	18.64%			
wb_conmax	193350	21334	88.97%	5821	3.01%	5571	95.71%	103.61%	104.17%	19.91
	183821	799	99.57%	184	0.10%	1	0.54%			
sparc_fpu	234125	15263	93.48%	8291	3.54%	7515	90.64%	94.58%	99.65%	14.08
	230642	3355	98.55%	2095	0.91%	765	36.52%			
sparc_spu	41939	2598	93.81%	669	1.60%	656	98.06%	98.67%	102.69%	5.46
	40503	260	99.36%	153	0.38%	0	0%			
sparc_ffu	48937	5155	89.47%	3554	7.26%	3232	90.94%	94.49%	100.37%	16.29
	48756	630	98.71%	525	1.29%	22	4.19%			
sparc_exu	116525	10753	90.77%	7072	6.07%	6338	89.62%	92.95%	102.57%	12.22
	116902	768	99.34%	694	0.59%	14	2.02%			
sparc_ifu	149116	10197	93.16%	6619	4.44%	5513	83.29%	96.06%	99.54%	13.99
	147376	1210	99.18%	677	0.46%	7	1.03%			
sparc_tlu	151591	9603	93.67%	5418	3.57%	4555	84.07%	91.19%	98.80%	15.78
	151328	1025	99.32%	756	0.68%	2	0.26%			
sparc_lsu	164658	9357	94.32%	5563	3.38%	4720	84.85%	97.41%	98.01%	10.63
	161196	850	99.47%	594	0.37%	0	0%			
average	92121.5	6750.05	92.27%	3422.25	4.26%	3066.85	87.64%	97.31%	101.15%	14.88
	91431.9	718.35	99.02%	390.5	0.58%	48.3	5.64%			

It can also be observed that in general, the relative runtime does not increase as the complexity of the circuit increases. This is related to the fact that only subcircuits with gates corresponding to undetectable faults are resynthesized. Therefore, the procedure described in this chapter is applicable to complex logic blocks in large chips. As mentioned earlier, the iterative resynthesis procedure can be integrated into an iterative design flow and thus avoid adding new iterations to the process.

5. LAYOUT RESYNTHESIS BY APPLYING DFM GUIDELINES TO AVOID LOW-COVERAGE AREAS OF A CELL-BASED DESIGN

DFM guidelines are recommended layout design practices intended to capture layout features that are difficult to manufacture correctly. Avoiding such features prevents the occurrence of potential systematic defects. Layout features that result in DFM guideline violations may not be avoided completely due to the design constraints of chip area, performance and power consumption. A framework for translating DFM guideline violations into potential systematic defects, and faults, was described earlier. In a cell-based design, the translated faults may be internal or external to cells. In Chapter 4, we focused on the faults that are internal to cells. We use a logic resynthesis procedure to eliminate large clusters of undetectable internal faults related to DFM guidelines. In this chapter we focus on undetectable faults that are external to cells. Using a layout resynthesis procedure that makes fine changes to the layout while maintaining the design constraints, we target areas of the design where large numbers of external faults related to DFM guideline violations are undetectable. By eliminating the corresponding DFM guideline violations, we ensure that the circuit does not suffer from low-coverage areas that may result in detectable systematic defects escaping detection, but failing the circuit in the field. The layout resynthesis procedure is applied to benchmark circuits and logic blocks of the OpenSPARC T1 microprocessor. Experimental results indicate that the improvement in the coverage of potential systematic defects is significant.

5.1 Introduction

The scaling of IC technologies has brought about many benefits including faster devices, lower power consumption, reduced chip sizes and increase in functionality. However, the continuous shrinking of device sizes also increases the gap between the feature size and the lithography wavelength. As a result, for each smaller process technology node, the chips are increasingly impacted by deviations in manufactured patterns from the intended design. Specifically, certain layout features are more difficult to manufacture than others, and are more likely to lead to circuit failures. When such features are present multiple times in a chip, they can result in repeated or systematic defects, which can impact the yield and DPPM significantly [20–25]. Due to modeling errors and algorithmic inaccuracies in removing the resulting systematic variations, process-related corrective actions using OPC/RET techniques are not sufficient for acceptable yield and DPPM [26]. Thus, appropriate interventions during circuit design are inevitable so as to remedy the potential manufacturing issues and address the systematic defects.

Such design interventions are formulated as design rules and DFM guidelines. While design rules are mandatory, and must all be applied to a design, DFM guidelines are taken as recommendations, and they are adhered to when possible within the design constraints of area, performance, and power consumption. When DFM guidelines are not adhered to, potential systematic defects may occur. The relationship between DFM guideline violations and potential systematic defects was discussed in [27–29]. In [27], DFM guidelines related to vias on interconnects, and contacts on p-diffusion, are considered. A more comprehensive set of DFM guidelines is considered in [28]. DFM guidelines related to internal nets of standard cells are considered in [29]. In all these works, the layout sites where DFM guidelines are violated are found, and the affected transistors are identified at the schematic level. The anticipated defect behaviors are then translated into gate-level logic faults using switch-level simulation. Test generation is carried out for the resulting faults to avoid potential test holes.

Among the potential faults resulting from DFM guideline violations, there are undetectable faults. When a large number of undetectable faults related to DFM guideline violations are present in an area of the circuit, the area suffers from a low coverage by tests that target the area. The missing tests may allow detectable defects in the area to escape detection. This can impact the yield and reliability significantly since the defects are likely to be systematic. This chapter demonstrates these issues and addresses them in the context of a cell-based design, and targeting faults that are external to cells. For this discussion we distinguish between faults that are internal and ones that are external to cells. We assume that undetectable faults, which are related to DFM guideline violations, and are internal to cells, can be addressed by proper logic synthesis [78–80]. In [78,79], manufacturability information is integrated into the cost function of logic synthesis, and a DFM extension library that contains yield-optimized cells is used for improving the manufacturability of the circuit. In [80], a logic resynthesis procedure is proposed that replaces cells with large numbers of undetectable internal faults related to DFM guideline violations by smaller cells that contain fewer faults. Overall, the procedure eliminates undetectable faults that are internal to cells, and also reduces the number of undetectable external faults. However, it leaves large numbers of undetectable external faults that cannot be addressed by replacing cells.

For faults that are external to cells, this chapter describes a layout resynthesis procedure that makes fine changes to the layout while maintaining the design constraints in order to improve the coverage of areas with low coverage because of the presence of undetectable external faults. The layout resynthesis procedure in itself (the procedure that makes local changes to the layout) is not the main contribution of this chapter. The contribution is related to the use of DFM guidelines to identify areas of the circuit with low coverage, and improving their coverage by layout resynthesis. From a test point of view, we show which DFM guideline violations need to be considered first so as to improve the layout areas with low coverage. This is the first layout resynthesis procedure to address this issue directly. As part of this solution, we suggest a layout-based coverage metric that can be used for identifying areas with low coverage.

As technologies evolve, many DFM guidelines remain the same and are transferred to the new technology. For example, the interconnect bridge defects shown in [81] to exist in 160nm technology are also believed to be an issue in the latest FinFET technologies [82], and require similar DFM guidelines. The proposed layout resynthesis procedure is independent of the cell library and the DFM guidelines. Therefore, it can work with different DFM guidelines related to different technology nodes even if new DFM guidelines are introduced. For different DFM guidelines, it may require different changes to the layout for fixing DFM guideline violations, but the basic methodology is the same.

The proposed layout resynthesis procedure eliminates undetectable external faults by fixing the DFM guideline violations that lead to them. The procedure prefers to eliminate faults whose effect on the coverage of the circuit is more significant. The DFM guideline violations are fixed by automatically changing the layout with the help of a place and route tool. The procedure does not allow any increase in critical path delay, power consumption or die area when changing the layout.

Other approaches for addressing the testability of a circuit are described in [67, 71, 83–99]. In [83–85, 95, 96], design-for-testability (DFT) methods for improving the transition fault coverage are discussed. The coverage is improved by inserting DFT logic that can provide better control over the state vectors. In [67, 71, 86, 92–94], logic resynthesis techniques are used for improving the testability of the circuit by reducing the difficulty of test generation. In [97–99], scan chain ordering is considered for improving the coverage for transition and path delay faults. With layout information taken into account, the routing penalty and the impact on circuit performance are limited. In [87–91], test point insertion is considered for improving the testability of a circuit, while limiting the deterministic pattern counts. We experimented with circuits into which test points are inserted to improve testability. The results indicate that, even with test points, there are areas with low coverage for faults that result from DFM guideline violations. Such areas require the layout resynthesis procedure described in this chapter. The reason is that test point insertion does not target DFM guideline violations directly, and therefore, does not target the resulting undetectable faults.

The procedure described in this chapter can be embedded into a standard cell based design flow. In a cell based design flow, after the initial design of the layout, several iterations of an incremental physical design process are typically required for satisfying the design constraints of delay, power and area. The proposed procedure is also iterative, and can thus fit within the overall iterative design process. In particular, an iteration of the design process can include one or more iterations of the proposed procedure to eliminate undetectable faults in poorly covered circuit areas, and improve the coverage of potential systematic defects. For a large chip, to maintain an acceptable computational effort, the proposed procedure can be applied to each logic block separately. We applied the proposed procedure to logic blocks of the OpenSPARC T1 microprocessor to demonstrate its applicability to such designs.

This chapter is organized as follows. Section 5.2 demonstrates the existence of undetectable external faults related to DFM guideline violations, and the presence of areas with poor coverage. Section 5.3 describes the proposed layout resynthesis procedure. Experimental results and analysis are presented in Section 5.4.

5.2 Undetectable Faults Related to DFM Guideline Violations

This section discusses the existence of undetectable external faults related to DFM guideline violations, and the presence of areas with poor coverage. A coverage metric is defined based on layout neighborhoods of undetectable faults.

Three categories of DFM guidelines are considered in this chapter. They are *Via*, *Metal* and *Density*. We use 19 guidelines in the *Via* category, 29 guidelines in the *Metal* category, and 11 guidelines in the *Density* category. These guidelines provide recommended layout constraints for dimensions of vias, spacings between exterior-facing edges on polygons, and densities of routing layers.

We translate DFM guideline violations into potential short and open defects that are external to cells, and then translate the potential defects into corresponding logic faults using the approach described in [27–29]. We denote the set of faults by F. A test generation procedure is applied to generate a test set T that detects all the detectable faults in F. We denote by $U = \{f_1, f_2, \ldots, f_n\}$ the set of undetectable faults in F.

5.2.1 Analysis of DFM Guideline Violations

In this section, we discuss the challenges related to DFM guidelines.

In Table 5.1, we show the numbers of DFM guideline violations for several circuits. In column *DFM_total*, we show the total number of DFM guideline violations in the circuit. In column *DFM_undet*, we show the number of DFM guideline violations translated to undetectable faults. It can be observed that the number of DFM guideline violations is typically very large, and it is not possible to fix all of them within the design constraints. The number of DFM guideline violations translated to undetectable faults is small compared to the total number of DFM guideline violations. This makes it possible for the layout resynthesis procedure described in this chapter to address them.

5.2.2 Detectable Defects Modeled by Undetectable Faults

In this section, we consider an example where faults that are translated from DFM guideline violations are undetectable, while potential systematic defects in the same

Table 5.1.: DFM Guideline Violations

Circuit	**DFM_total**	**DFM_undet**
b15	218619	10786
b20	249383	3352
sparc_fpu	1666799	9012
sparc_exu	899740	20328

area are detectable. The presence of such situations motivates the need for layout resynthesis to eliminate undetectable faults and improve the coverage of the area.

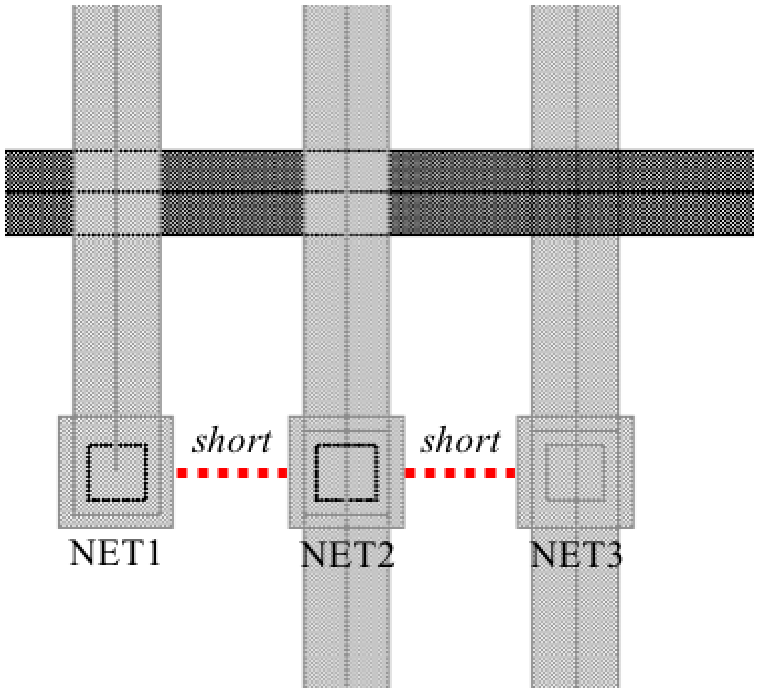

Fig. 5.1.: Potential short defects.

We conducted the following experiment to determine the existence of detectable, potentially systematic defects that may go undetected. The DFM guideline we considered specifies the recommended minimum separation between exterior facing edges of *metal4* polygons. Figure 5.1 shows an example where this DFM guideline is violated for NET1 and NET2, as well as NET2 and NET3. These violations can potentially cause shorts between NET1 and NET2, and between NET2 and NET3. The shorts are modeled by bridging faults. Suppose that both bridging faults are undetectable.

A possible defect that is not covered by the DFM guideline is a short between NET1 and NET3 (i.e. NET1, NET2 and NET3 are shorted). This defect is not modeled by F, and a test for it is not generated directly. Nevertheless, the defect may occur because the site is prone to be defective. Without tests that detect the shorts between NET1 and NET2, and between NET2 and NET3, this defect may remain undetected by T. In this case, the low coverage around NET1, NET2 and NET3, and the missing tests for the two bridging faults, causes the bridge between NET1 and NET3 to go undetected.

We searched for occurrences of this situation in benchmark circuits and logic blocks of the OpenSPARC T1 microprocessor. The test set we used detects all the detectable transition and stuck-at faults, as well as the bridging faults in F. The results for several circuits are shown in Table 5.2. For every circuit, column *Uncov* shows the number of occurrences of undetected defects as illustrated by Figure 5.1.

Although the numbers in Table 5.2 are small, they represent only one example where the presence of undetectable faults may allow detectable defects to go undetected. This motivates the layout resynthesis procedure described in Section 5.3 that eliminates undetectable faults in areas with low coverage.

In general, since sites of DFM guideline violations are more likely to be defective than other sites, and circuits manufactured prior to volume production tend to suffer from multiple defects, it can be expected that multiple DFM violation sites would be defective. In addition to the double fault illustrated above, there can be other types of undetectable faults related to DFM guideline violations that are undetectable alone, but become detectable when two or more faults are present together. One can try to add tests to detect multiple faults [62, 100–102], but the number of faults can be very large, and the number of tests may increase dramatically. For example, in the circuits we considered, we found hundreds and even thousands of undetectable faults related to DFM guideline violations, leading to millions and more multiple faults. The resynthesis procedure we propose in this chapter eliminates or drastically reduces the number of undetectable faults related to DFM guideline violations. If desired, one can add tests for detectable multiple faults that consist of undetectable faults remaining after resynthesis [62].

Table 5.2.: Uncovered Short Defects

Circuit	**Uncov**
b15	42
b20	65
sparc_fpu	75
sparc_exu	76

5.2.3 Circuit Areas with Poor Coverage

In this section, we define a coverage metric, and demonstrate the presence of areas with poor coverage that suffer from the presence of undetectable external faults related to DFM guideline violations.

In Figure 5.2, we show the topological distribution of the undetectable external faults related to DFM guideline violations in the layout of *sparc_fpu*. It can be observed that the undetectable faults tend to cluster in the darker areas, resulting in areas with large numbers of undetectable faults.

We define the coverage of an area as the percentage of detectable faults among all the faults related to DFM guideline violations in this area. The details of this definition are discussed next.

We say that two layout sites are adjacent to each other if the distance between them is less than 20× the minimum feature size. Within such distance, the two layout sites can be affected by similar optical interactions, which are the major causes for systematic defects [103].

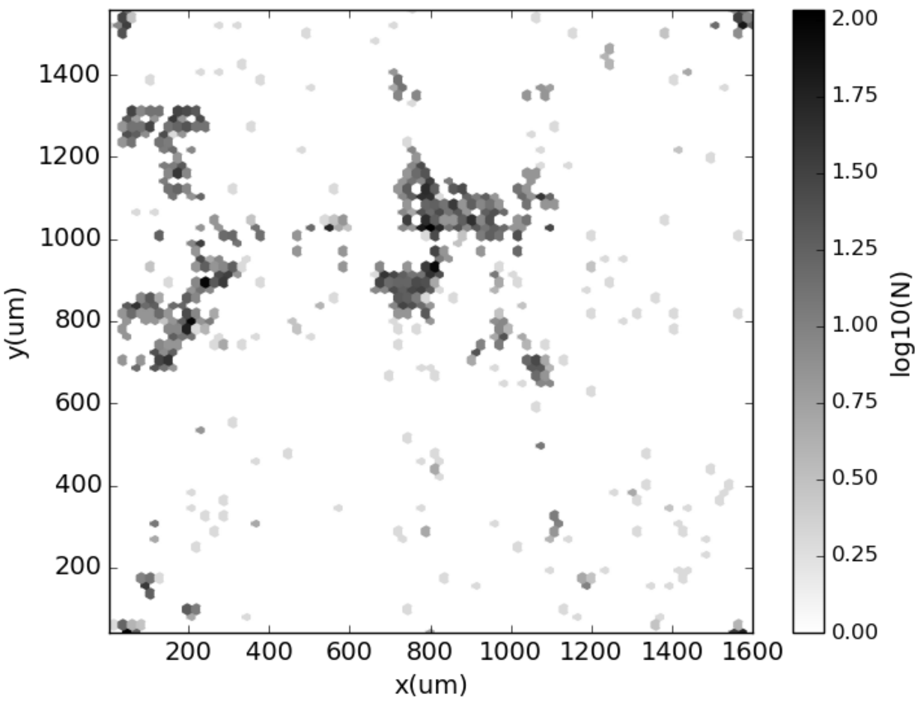

Fig. 5.2.: Undetectable external faults related to DFM guideline violations in *sparc_fpu*.

For a defect d that is obtained from a DFM guideline violation, we define the neighborhood of d to include all the layout sites that are adjacent to the site of d.

A fault f in F may model several different defects. The defects are always at a close proximity to each other. We define the neighborhood of f as the union of the neighborhoods of all the defects that f models.

We say that a fault f' is in the neighborhood of a fault f if the site of a defect d' modeled by f' is in the neighborhood of f. With these definitions, we define the coverage $c(f)$ of the neighborhood of a fault f as follows.

$$c(f) = \frac{\textit{Number of detectable faults in the neighborhood of } f}{\textit{Total number of faults in the neighborhood of } f} \tag{5.1}$$

For illustration, we computed coverages for the neighborhoods of all the undetectable external faults in *sparc_fpu*, and partitioned the faults according to their coverage range. The results are shown in Figure 5.3.

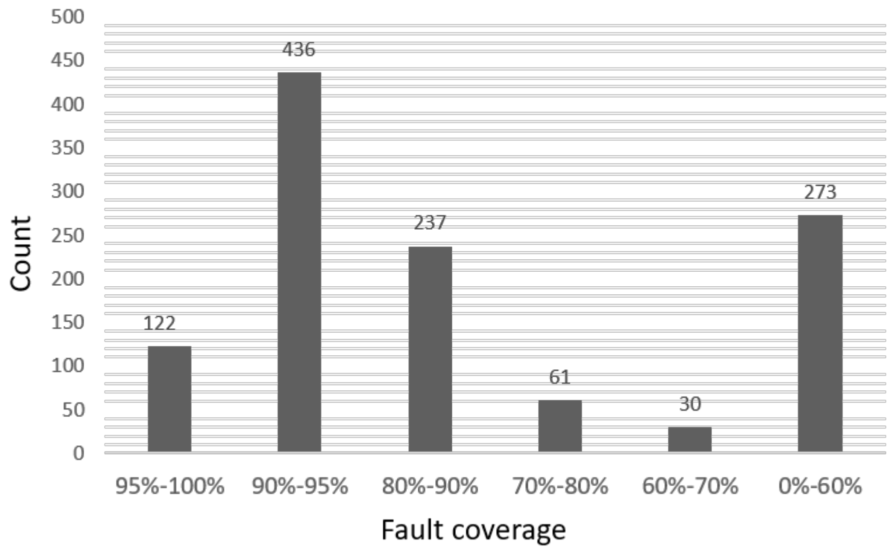

Fig. 5.3.: Coverages for the undetectable external faults in *sparc_fpu*.

It can be seen that the coverages for more than half of the undetectable external faults are below 90%. About a quarter of the faults have coverages below 60%. These observations motivate the need to eliminate undetectable external faults with poorly covered neighborhoods, and thus improve the coverage for potential systematic defects.

5.2.4 Coverage for Faults with Weighted DFM Guidelines

The coverage metric $c(f)$ defined above assumes that all the DFM guidelines are equally important. In this section, we define the coverage of the neighborhood of a fault f when DFM guidelines have different levels of importance.

The evaluation of DFM guidelines was discussed in [104–106]. In [104], test structures are used for evaluating the importance of DFM guidelines. In [105, 106], information extracted from actual failed ICs during volume diagnosis is used for measuring the effectiveness of a DFM guideline.

In this chapter, we define the weight of a DFM guideline as the probability of a defect given a violation of this guideline. A violation of a DFM guideline with a higher weight indicates a higher risk of failure. We also define the weight of a fault related to DFM guideline violations based on the corresponding DFM guideline weights, as follows.

Suppose that a fault f is translated from violations of n different DFM guidelines, $g_1, g_2, \ldots, g_n$, and the weights of $g_1, g_2, \ldots, g_n$ are $p_1, p_2, \ldots, p_n$. The numbers of times that $g_1, g_2, \ldots, g_n$ are violated are denoted by $m_1, m_2, \ldots, m_n$. We assume that all the DFM guideline violations are independent. We define the weight $w(f)$ of f as the probability that f is present in the circuit given all the DFM guideline violations that are translated to f. We have that

$$w(f) = 1 - (1-p_1)^{m_1}(1-p_2)^{m_2}\ldots(1-p_n)^{m_n} \tag{5.2}$$

Next, considering all the faults in the neighborhood of f, we define the coverage $c(f)$ of the neighborhood of f as shown in Equation (5.3). In Equation (5.3), $\sum w(det)$ and $\sum w(undet)$ give the total weights of the detectable and undetectable faults in the neighborhood of f, respectively. In addition, they can be considered as the expected numbers of detectable and undetectable faults in the neighborhood of f. Thus, the subtrahend of the equation can be considered as the probability that f is present in the circuit, while its neighborhood is not covered by the test set that detects all the detectable faults. When this probability is high, the detectable systematic defects in the neighborhood of f may go undetected, causing circuit failure.

$$c(f) = 1 - w(f) \cdot (1 - \frac{\sum w(det)}{\sum w(det) + \sum w(undet)}) \tag{5.3}$$

5.3 Layout Resynthesis

This section describes a methodology for eliminating undetectable faults by fixing the DFM guideline violations leading to them, and the layout resynthesis procedure that is built upon it.

5.3.1 Fixing DFM Guideline Violations

In this section, we use an example to illustrate a methodology for eliminating the undetectable faults related to DFM guideline violations. This methodology is based on the use of a place and route tool to make local changes to the layout for fixing the related DFM guideline violations. For different DFM guideline violations, the concrete modifications to the layout may be different. The connectivity of the circuit is maintained when modifying the layout as described later. The DFM guideline used as an example specifies that the separation between exterior facing edges of *metal1* polygons should be no less than 330nm. In Figure 5.4, this DFM guideline is violated for polygons A and B, resulting in an undetectable bridging fault that involves the two polygons.

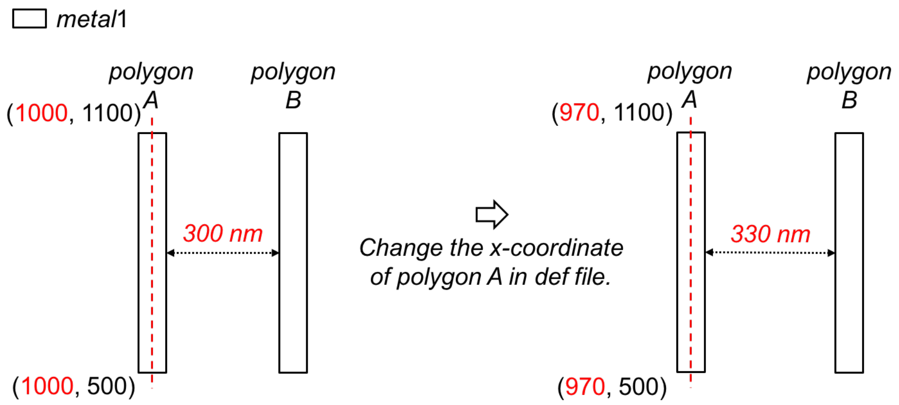

Fig. 5.4.: Fixing a DFM guideline violation.

One way to fix this violation is to move polygon A horizontally, such that the separation between the polygons would be enlarged. We achieve this by first extracting the design exchange format (def) file of the layout. The def file of a layout records the coordinates of all the polygons in the layout. We then change the x-coordinate of polygon A by subtracting 30 units. We also change the x-coordinates of the polygons that are connected to polygon A in the same manner to ensure the connectivity of the circuit. After modifying the def file, we provide the modified def file back to the place and route tool. The tool changes the locations of polygon A and polygons that are connected to polygon A in the layout accordingly. After moving polygon A to the new layout site, the separation between exterior facing edges of polygons A and B is $330nm$, which adheres to the DFM guideline considered. As a result, the DFM guideline violation that results in the undetectable bridging fault is eliminated.

When an undetectable fault is translated from several different DFM guideline violations, we attempt to eliminate the undetectable fault completely so as to improve the coverage of its neighborhood. This is achieved by fixing all the DFM guideline violations leading to the fault, and changing all the polygons involved.

The resynthesis procedure sometimes decides to undo a layout modification. To implement this operation, the procedure stores the original coordinates of the polygons that it moves. By changing the coordinates of a polygon in the def file to its original coordinates, the polygon is moved back to its original position.

5.3.2 Layout Resynthesis Procedure

In this section, we describe an iterative layout resynthesis procedure that makes fine changes to the layout in order to eliminate undetectable faults whose neighborhoods have low coverage. This is achieved by fixing the DFM guideline violations that lead to them. The procedure considers every undetectable fault, and attempts to eliminate the ones whose neighborhoods have the lowest coverages. When an undetectable fault is eliminated from a neighborhood with a low coverage, the coverage of the neighborhood increases.

The proposed procedure starts with the original physical design L_{ayout} of the circuit. We assume that L_{ayout} was already optimized by one or more iterations of a physical design flow, and satisfies the design constraints of delay, power and area. The proposed procedure improves L_{ayout} with respect to the coverage of the circuit without violating the original design constraints.

We use a set U_{done} to store every undetectable fault, for which the procedure has completed the attempt to eliminate it successfully. Initially, U_{done} is empty. The target coverage of the original neighborhood of an undetectable fault is denoted by p. The proposed procedure attempts to ensure that the coverage of the original neighborhood of every undetectable fault initially in the circuit is no less than p. A higher value of p indicates a higher coverage for potential systematic defects after applying the procedure, and more changes to the layout that require a higher runtime. The proposed layout resynthesis procedure can accommodate different values of p. With p=100%, the procedure considers all the undetectable faults.

In every iteration, the procedure computes the set U_{cur} of undetectable faults resulting from DFM guideline violations that are not in U_{done} based on the current layout of the circuit. For every fault it also computes its coverage. A fault with a coverage of p and above is excluded from U_{cur}. The procedure attempts to eliminate the undetectable faults in U_{cur} one by one, considering the faults from low to high coverage of their neighborhoods. This ensures that the faults with the lowest coverages are considered earlier. Such faults are also more important to consider.

When an undetectable fault f from U_{cur} is considered, it is possible that another fault, f', in its neighborhood has already been considered and added to U_{done} in this iteration. In this case, the procedure does not consider f. The reason is that the elimination of f' made a change to its neighborhood that may affect the coverage for f. In the next iteration, the procedure will recompute the coverage for f, and decide whether it still needs to be considered.

After every attempt to eliminate an undetectable fault, the procedure checks whether the design constraints of delay, power and area are satisfied. It also checks whether the modified design has Design-Rule-Check (DRC) or Layout-Versus-Schematic (LVS) violations. If all the design constraints are satisfied and there is no DRC or LVS violation, the undetectable fault is added to U_{done}. Otherwise, the modification to the layout is discarded, and the next eligible undetectable fault in U_{cur} is considered by the procedure.

After considering all the eligible undetectable faults in U_{cur}, and modifying the layout accordingly, the proposed procedure recomputes the faults related to DFM guideline violations based on the modified layout. A test generation procedure for fault detection is carried out only when new faults are obtained. The procedure then computes the coverages of the original neighborhoods of the undetectable faults targeted in this iteration. For every target undetectable fault f, the procedure checks whether the coverage of the original neighborhood of f increased, and the layout modification for eliminating f does not result in more DFM guideline violations leading to undetectable faults. If these conditions are not satisfied, the layout modification for fixing the DFM guideline violations leading to f is discarded.

The procedure described above is shown in Algorithm 5.1. We denote the process for fixing the DFM guideline violations leading to a fault f by $ApplyDFM(f)$, and the reverse process for removing the corresponding layout modification by $UnapplyDFM(f)$. The procedure terminates when (1) the coverages of all the original neighborhoods of the undetectable faults initially in the circuit are at least p, or (2) these coverages cannot be improved further without violating the design constraints of delay, power and area, or introducing more DFM guideline violations leading to undetectable faults.

5.4 Experimental Results

In this section, we describe five experiments to demonstrate the effectiveness and applicability of the proposed layout resynthesis procedure in different scenarios.

Procedure 5.1 Layout Resynthesis Procedure

```
1: $U_{done} = \varnothing$
2: $L_{ayout}$ is the original layout of the circuit
3: while true do
4:    Compute $U_{cur}$ based on $L_{ayout}$, $p$ and $U_{done}$
5:    if $U_{cur} == \varnothing$ then
6:       stop
7:    end if
8:    Sort the faults in $U_{cur}$ in the order of increasing neighborhood coverages
9:    for every fault $f$ in $U_{cur}$, if it is eligible do
10:      $layout = ApplyDFM(f)$
11:      if the design constraints are satisfied and there is no DRC or LVS violation
         then
12:         $L_{ayout} = layout$
13:         Add $f$ into $U_{done}$
14:      end if
15:   end for
16:   if $L_{ayout}$ is not modified then
17:      stop
18:   end if
19:   for every fault $f$ added into $U_{done}$ in this iteration, if the coverage of its origi-
      nal neighborhood does not increase or more DFM guideline violations causing
      undetectable faults are obtained do
20:      $L_{ayout} = UnapplyDFM(f)$
21:   end for
22: end while
```

In all the experiments, we apply the proposed procedure to benchmark circuits, and to logic blocks of the OpenSPARC T1 microprocessor. The proposed procedure is applied to the logic blocks in a single SPARC core, and the floating-point unit (*sparc_fpu*).We run the procedure on a Linux machine with 2.6GHz processors.

We use the tool kit with a standard cell library developed by OSU to synthesize the RTL descriptions of the circuits into gate-level netlists and layouts. The tool kit was developed based on TSMC 180nm technology. For the circuits considered in all the experiments, the netlists obtained after logic synthesis are flattened and treated as one block with respect to floorplanning. In the first four experiments, we set the cell utilization to be 70% for all the circuits. The allocation area, total number of nets and total wire length of the original layout for each circuit are shown in Table 5.3. In the fifth experiment, we set the cell utilization to be 80% and 90% to show the applicability of the proposed procedure to more congested designs.

Table 5.3.: Layouts with 70% Cell Utilization

Circuit	**alloc_area** (mm^2)	**nets**	**wire_length** (um)
b14	0.15	17480	139766
b15	0.28	33640	276498
b20	0.32	38049	291774
aes_core	0.7	119197	1143766
DMA	0.75	78925	915701
tv80	0.25	32988	290335
systemcaes	0.37	46692	517923
s35932	0.67	47543	401998
wb_conmax	1.1	225595	3092420
sparc_spu	0.6	49786	554713
sparc_ffu	0.51	56240	617818
sparc_exu	1.09	140785	1992839
sparc_lsu	1.99	213751	3403376
sparc_tlu	1.94	196218	2743798
sparc_ifu	1.59	187349	2653452
sparc-fpu	2.33	254847	2759603

We use a commercial tool for logic synthesis. We also use a commercial place and route tool for layout synthesis and for applying DFM guidelines. A commercial IC verification and sign-off package is used for finding locations of DFM guideline violations in the layout. We use a commercial ATPG tool to generate test patterns for fault detection.

5.4.1 Layout Resynthesis Procedure

In this experiment, the layout resynthesis procedure as described in Section 5.3 is applied. The results are shown in Tables 5.4 and 5.5 as follows. In each case, three rows correspond to a circuit. The first row describes the original design of the circuit. Row *half* describes the resynthesized circuit when p, the target coverage of the original neighborhood of an undetectable fault, is set to be the median of the coverages of the neighborhoods of all the undetectable faults in the original design. This indicates that only half of the original undetectable faults are considered for elimination. Row *all* describes the resynthesized circuit when p is set to be 100%, which indicates that the procedure attempts to eliminate all the undetectable faults related to DFM guideline violations.

In every row, column F provides the total number of faults related to DFM guideline violations. Column U provides the number of undetectable faults in F. Column *Cov* provides the coverage of the circuit, which is defined as $Cov = (1 - U/F)\%$. Column *Ave_c* provides the average coverage of the original neighborhoods of undetectable faults initially in the circuit. Column *Nchanges* provides the number of changes to polygons during layout resynthesis. In column *Rtime*, we show the run time for the proposed layout resynthesis procedure relative to the run time for one iteration of logic synthesis and physical design with test generation for the logic faults related to DFM guideline violations. We include test generation time since a test set is also obtained by the proposed layout resynthesis procedure.

From Tables 5.4 and 5.5 it can be observed that the procedure described in this chapter achieves a significant reduction in the numbers of undetectable faults for all the circuits considered. This can be seen from column U. As the proposed procedure always attempts to eliminate the undetectable faults with the lowest coverages, and no additional DFM guideline violations causing undetectable faults are allowed to be introduced to the circuit, the coverage of the original neighborhoods of undetectable faults increases significantly. This can be seen from column *Ave_c*.

It can also be observed that when p is set to be the median of the coverages of the neighborhoods of undetectable faults in the original design, it typically requires less than half of the number of polygon changes and runtime relative to the case where p is set to be 100%. This is because eliminating an undetectable fault typically increases the coverages for the undetectable faults in its neighborhood. Thus, fewer than half of the undetectable faults need to be considered.

Table 5.4.: Layout Resynthesis (Benchmarks)

Circuit		**F**	**U**	**Cov**	**Ave_c**	**Nchanges**	**Rtime**
b14	orig	11582	182	98.43%	89.14%	/	1
	half	11518	117	98.98%	94.02%	189	2.02
	all	11404	2	99.98%	99.87%	556	5.7
b15	orig	21103	713	96.62%	85.64%	/	1
	half	20807	417	98.00%	93.07%	631	3.28
	all	20408	10	99.95%	99.80%	2398	7.02
b20	orig	25815	339	98.69%	91.00%	/	1
	half	25704	213	99.17%	95.14%	329	2.12
	all	25492	2	99.99%	99.93%	1037	6.36
aes_core	orig	75092	874	98.84%	93.93%	/	1
	half	74688	455	99.39%	97.49%	575	4.18
	all	74240	11	99.99%	99.90%	1211	9.5
DMA	orig	44589	409	99.08%	91.55%	/	1
	half	44425	231	99.48%	96.23%	286	2.66
	all	44191	2	99.99%	99.95%	821	6.63
tv80	orig	20292	757	96.27%	80.43%	/	1
	half	20048	508	97.47%	87.89%	506	1.74
	all	19542	3	99.99%	99.95%	2618	6.77
systemcaes	orig	26214	267	98.98%	89.43%	/	1
	half	26102	146	99.44%	95.64%	238	1.95
	all	25958	2	99.99%	99.87%	1240	3.52
s35932	orig	32895	227	99.31%	89.46%	/	1
	half	32804	122	99.63%	95.83%	69	2.59
	all	32676	0	100.00%	100.00%	194	5.37
wb_conmax	orig	112351	627	99.44%	93.98%	/	1
	half	112047	319	99.72%	97.82%	340	2.22
	all	111756	19	99.98%	99.82%	785	3.38

From column *Rtime*, we can see that the relative runtime does not increase as the complexity of the circuit increases. This is because the layout modification for applying DFM guidelines is based on the original layout of the circuit. The proposed procedure does not require the layout to be implemented from the gate-level netlist repeatedly.

For further illustration, in Figure 5.5, we show the numbers of the original neighborhoods of undetectable faults initially in the circuit in different coverage ranges for *sparc_fpu*. The three groups of bars correspond to the three rows for *sparc_fpu*

Table 5.5.: Layout Resynthesis (OpenSPARC T1)

Circuit		F	U	Cov	Ave_c	Nchanges	Rtime
sparc_spu	orig	30046	295	99.02%	71.34%	/	1
	half	29926	174	99.42%	88.38%	109	2
	all	29790	24	99.92%	94.64%	562	4.95
sparc_ffu	orig	32605	413	98.73%	74.05%	/	1
	half	32433	238	99.27%	87.12%	127	1.39
	all	32234	37	99.89%	95.43%	563	3.38
sparc_exu	orig	76709	1006	98.69%	84.19%	/	1
	half	76226	514	99.33%	91.76%	417	1.52
	all	75745	39	99.95%	98.51%	1745	2.68
sparc_lsu	orig	114076	1440	98.74%	83.20%	/	1
	half	113427	787	99.31%	91.61%	649	1.18
	all	112715	76	99.93%	98.18%	2114	2.86
sparc_tlu	orig	104694	1553	98.52%	77.67%	/	1
	half	103982	828	99.20%	87.53%	729	2.02
	all	103224	70	99.93%	95.40%	2165	3
sparc_ifu	orig	101074	1858	98.16%	66.93%	/	1
	half	100351	1123	98.88%	82.51%	820	1.28
	all	99271	48	99.95%	91.29%	3581	2.29
sparc_fpu	orig	157728	1159	99.27%	73.72%	/	1
	half	157295	723	99.54%	88.53%	586	2.51
	all	156639	63	99.96%	95.04%	2376	4.04

in Table 5.5. It can be observed that the coverages of the original neighborhoods of undetectable faults increase significantly.

5.4.2 Circuits with Test Points

In this section, we show the experimental results of applying the layout resynthesis procedure to circuits with test points.

The commercial tool we use inserts test points for three purposes, (1) to improve coverage for random pattern resistant faults, (2) to improve coverage for undetected faults during ATPG, and (3) to reduce deterministic test pattern counts. We experimented with all the seven possible combinations of the three groups of test points. Of the seven designs of a circuit, we select the one with the highest coverage for the faults

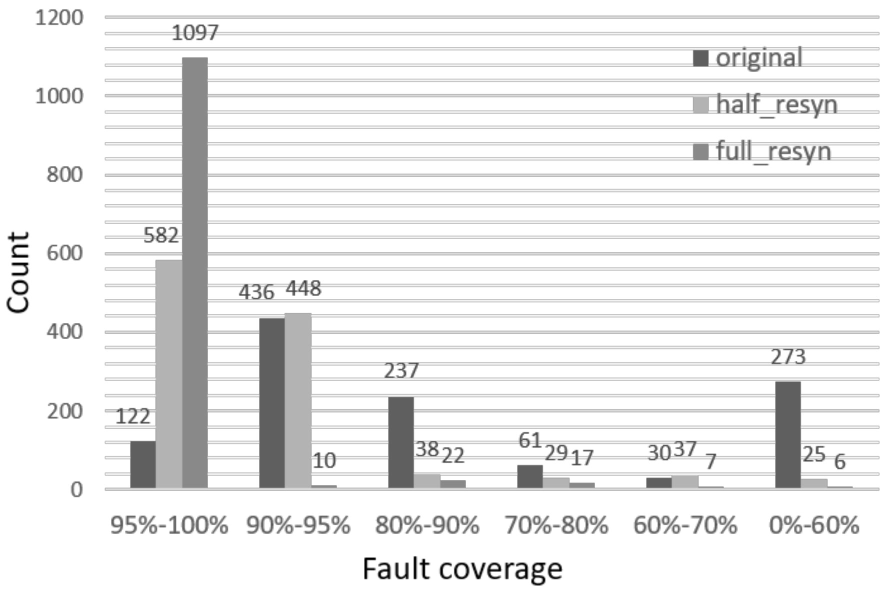

Fig. 5.5.: Coverages of the original neighborhoods of undetectable faults in *sparc_fpu* after resynthesis.

related to DFM guideline violations. We then apply the proposed layout resynthesis procedure to the selected design with p set to be 100%.

Table 5.6.: Circuits with Test Points

Circuit		**F**	**U**	**Cov**	**Ave_c**	**Ntests**	**Nchanges**	**Rtime**	**Delay**	**Power**	**Area**
b15 (4722, 155)	orig_ntp	21103	713	96.62%	85.64%	378	/	1	100.00%	100.00%	100.00%
	resyn_ntp	20408	10	99.95%	99.80%	378	2398	7.02	98.27%	97.40%	100.00%
	orig_tp	21960	184	99.16%	90.18%	256	/	1	108.34%	106.68%	104.16%
	resyn_tp	21789	5	99.98%	99.25%	257	619	3.11	107.78%	104.93%	104.16%
DMA (10696, 331)	orig_ntp	44589	409	99.08%	91.55%	996	/	1	100.00%	100.00%	100.00%
	resyn_ntp	44191	2	99.99%	99.95%	997	821	6.63	99.63%	98.48%	100.00%
	orig_tp	46303	286	99.38%	91.75%	650	/	1	100.70%	100.76%	102.98%
	resyn_tp	46034	4	99.99%	99.89%	651	591	5.38	100.41%	99.32%	102.98%
sparc_ifu (22257, 413)	orig_ntp	101074	1858	98.16%	66.93%	384	/	1	100.00%	100.00%	100.00%
	resyn_ntp	99271	48	99.95%	91.29%	384	3581	2.29	98.77%	99.13%	100.00%
	orig_tp	116836	1539	98.68%	70.11%	201	/	1	105.48%	108.14%	102.82%
	resyn_tp	115331	31	99.97%	93.45%	201	3417	2.23	104.68%	107.46%	102.82%
sparc_fpu (34418, 545)	orig_ntp	157728	1159	99.27%	73.72%	599	/	1	100.00%	100.00%	100.00%
	resyn_ntp	156639	63	99.96%	95.04%	600	2376	4.04	99.62%	98.83%	100.00%
	orig_tp	161059	1073	99.33%	74.13%	232	/	1	103.78%	107.80%	101.79%
	resyn_tp	160131	39	99.98%	95.78%	232	2203	3.99	103.49%	107.13%	101.79%

The results for several circuits are shown in Table 5.6. Under the name of the circuit, we show the number of nets in the circuit, followed by the number of inserted test points. For comparison, in every case, we repeat the results for the circuit without test points in the first two rows. The last two rows describe the results for the circuit with test points. In both scenarios, we present the results of the original design in the first row and the results of the resynthesized design in the second row.

In addition to the columns defined for Tables 5.4 and 5.5, column *Ntests* in Table 5.6 provides the number of test patterns that detect all the detectable faults related to DFM guideline violations. In columns *Delay*, *Power* and *Area*, we show the critical path delay, power consumption and die area of the circuit relative to the ones of the original design without test points. The increases in delay, power and area are due to the insertion of test points. The proposed layout resynthesis procedure does not allow any increases in delay, power or area compared to the original design which it is applied to. It is possible to maintain the original design constraints by limiting the test points inserted to the circuit. However, in this chapter, we prefer not to interfere with the analysis and insertion of test points provided by the commercial tool.

It can be observed from Table 5.6 that the insertion of test points can improve the coverage of the circuit, while reducing the test pattern count. It also improves the coverage of the neighborhoods of undetectable faults. However, this coverage is still low in many cases. After applying the proposed layout resynthesis procedure, the coverage of the original neighborhoods of undetectable faults initially in the circuit increases significantly. Therefore, the application of the proposed procedure can also benefit the coverage for potential systematic defects when the circuit contains test points.

In some cases, column Ave_c of Table 5.6 shows that after applying the proposed procedure, the circuit with test points has an average coverage that is lower than the one of the circuit without test points. This is because the original layout of the circuit with test points is different from the original layout of the circuit without test points. The undetectable faults related to DFM guidelines in the two circuits can be different. Therefore, the unresolved undetectable faults in the two circuits due to design constraints can be different. In some cases, the coverage of the neighborhoods of certain unresolved faults in the circuit with test points may be low. As a result, the average coverage after applying the proposed procedure to the circuit with test points can be lower.

5.4.3 Circuits after Logic Resynthesis

In this section, we show the experimental results of applying the layout resynthesis procedure after applying the logic resynthesis procedure described in [80]. This procedure addresses undetectable faults related to DFM guideline violations that are internal to cells by replacing cells with large numbers of undetectable internal faults by different cells. In this experiment, we focus on the undetectable external faults related to DFM guideline violations that remain in the circuit.

The results for several circuits are shown in Table 5.7. For every circuit, row *orig_inter* describes the original circuit after applying the logic resynthesis procedure in [80]. Row *resyn* describes the circuit that is resynthesized using the layout resynthesis procedure described in this chapter. The columns of Table 5.7 are the same as in Tables 5.4, 5.5 and 5.6.

Table 5.7.: Circuits after Logic Resynthesis

Circuit		**F**	**U**	**Cov**	**Ave_c**	**Ntest**	**Nchanges**	**Rtime**
b15	orig_inter	27265	743	97.27%	83.19%	382	/	1
	resyn	26533	11	99.96%	99.77%	382	2461	7.23
DMA	orig_inter	53567	413	99.23%	91.87%	1001	/	1
	resyn	53156	1	99.99%	99.99%	1001	842	6.32
sparc_ifu	orig_inter	115154	1199	98.96%	73.15%	391	/	1
	resyn	113988	32	99.97%	93.19%	391	2910	2.01
sparc_fpu	orig_inter	185598	1517	99.18%	72.87%	609	/	1
	resyn	184153	67	99.96%	95.18%	610	2895	4.17

From Table 5.7, it can be observed that the circuits after logic resynthesis still contain large numbers of undetectable external faults related to DFM guideline violations, resulting in low-coverage areas in the design. This can be seen from columns *U* and *Ave_c*. The proposed layout resynthesis procedure addresses this issue by eliminating undetectable faults in the neighborhoods with low coverages. As a result, the coverage of the original neighborhoods of undetectable faults initially in the circuit increases significantly.

5.4.4 Weighted DFM Guidelines

In this section, we show the experimental results of applying the layout resynthesis procedure when different DFM guidelines have different weights.

We experimented with two cases for comparison. In the first case, the weights assigned to spacing related DFM guidelines are 50%. In the second case, the weights assigned to via dimension related DFM guidelines are 50%. We select the two types of DFM guidelines since they are related to most of the DFM guideline violations. In both cases, the weights assigned to the rest of the DFM guidelines are 5%, and p is set to be 80%.

Table 5.8.: Weighted DFM Guidelines

Circuit		**F**	**U**	**Cov**	**Ave_c**		**Ntests**	**Nchanges**	**Rtime**
					orig	resyn			
b15	spacing	21092	702	96.67%	93.07%	93.54%	378	99	0.25
	via	20973	582	97.23%	88.03%	93.51%	378	390	0.89
DMA	spacing	44589	409	99.08%	93.12%	95.67%	996	31	0.19
	via	44579	399	99.10%	92.27%	95.03%	996	104	0.77
sparc_ifu	spacing	100998	1782	98.24%	78.63%	90.54%	384	601	0.48
	via	100930	1713	98.30%	75.91%	90.18%	384	826	0.59
sparc_fpu	spacing	157620	1051	99.33%	89.34%	96.23%	599	422	0.31
	via	157563	994	99.37%	87.42%	95.82%	599	524	0.47

The results for several circuits are shown in Table 5.8. For every circuit, row *spacing* describes the case when spacing related DFM guidelines are assigned higher weights, and row *via* describes the case when via dimension related DFM guidelines are assigned higher weights. The columns of Table 5.8 are the same as in Tables 5.4, 5.5, 5.6 and 5.7. In column *Ave_c*, we show the results for the original and the resynthesized circuits. In other columns, we show the result for the resynthesized circuit.

From Table 5.8, it can be seen that the number of changes to polygons in the first case is less than the one in the second case. This is because more DFM guideline violations that are translated to undetectable faults are related to via dimensions. This can also be validated from column *Ave_c*. When via dimension related DFM guidelines are assigned higher weights, the average coverage of the original neighborhoods of undetectable faults is typically lower than the one when spacing related DFM guidelines are assigned higher weights.

5.4.5 Circuits with High Cell Utilization

The cell utilization used thus far is 70% as discussed earlier. In this section, we show the results of applying the layout resynthesis procedure to circuits with higher cell utilization. We experimented with two cases for comparison. In the two cases, the cell utilization is set to be 80% and 90% respectively.

The results for several circuits are shown in Table 5.9. The original layouts of all the circuits considered are obtained without design rule violations in all the cases. For every circuit, we repeat the results with 70% cell utilization, and then show the results with 80% and 90% cell utilization. In all the scenarios, we present the results of the original design in the first row and the results of the resynthesized design in the second row. The columns of Table 5.9 are the same as in Tables 5.4, 5.5 and 5.6.

It can be observed from Table 5.9 that the numbers of unresolved undetectable faults increase compared to the ones in the circuits with 70% cell utilization. This is due to the congestion in the circuit with high cell utilization. However, the proposed layout resynthesis procedure still reduces the number of undetectable faults significantly. Therefore, the coverage of the original neighborhoods of undetectable faults initially in the circuit increases significantly. This can be seen from column *Ave_c*.

Table 5.9.: Circuits with Higher Cell Utilization

Circuit		**F**	**U**	**Cov**	**Ave_c**	**Nchanges**	**Rtime**
b15	orig_70%	21103	713	96.62%	85.64%	/	1
	resyn_70%	20408	10	99.95%	99.80%	2398	7.02
	orig_80%	21489	811	96.23%	81.32%	/	1
	resyn_80%	20702	14	99.93%	99.19%	2491	8.19
	orig_90%	22950	903	96.07%	82.06%	/	1
	resyn_90%	22075	20	99.91%	99.31%	2610	7.79
DMA	orig_70%	44589	409	99.08%	91.55%	/	1
	resyn_70%	44191	2	99.99%	99.95%	821	6.63
	orig_80%	45033	524	98.84%	89.58%	/	1
	resyn_80%	44530	7	99.98%	99.91%	1037	7.18
	orig_90%	46194	629	98.64%	88.37%	/	1
	resyn_90%	45607	25	99.95%	99.36%	1153	7.68
sparc_ifu	orig_70%	101074	1858	98.16%	66.93%	/	1
	resyn_70%	99271	48	99.95%	91.29%	3581	2.29
	orig_80%	102019	2105	97.94%	63.19%	/	1
	resyn_80%	100029	97	99.90%	89.93%	3710	2.96
	orig_90%	104986	2973	97.17%	62.64%	/	1
	resyn_90%	102173	156	99.85%	87.56%	4002	3.59
sparc_fpu	orig_70%	157728	1159	99.27%	73.72%	/	1
	resyn_70%	156639	63	99.96%	95.04%	2376	4.04
	orig_80%	163579	1490	99.09%	74.56%	/	1
	resyn_80%	162196	102	99.94%	91.63%	2865	4.68
	orig_90%	170479	1817	98.93%	72.10%	/	1
	resyn_90%	168854	177	99.90%	90.29%	3309	4.81

6. SUMMARY

This dissertation addressed three important issues in test and diagnosis that are related to delay fault testing, defect diagnosis and systematic defects based on DFM guidelines. The solutions to these issues were implemented using commercial EDA tools. No modification to these commercial tools is required. Thus, they can be easily applied to complex designs with state-of-the-art features.

First, an implementation of a functional broadside test generation procedure using a commercial ATPG tool was presented. Functional broadside tests use only reachable states as scan-in states to address overtesting of delay faults. The procedure starts with a initial state and iteratively generates more reachable states and more tests. Features that already exist in the commercial tool allowed us to implement this procedure without modifying the tool. This allows functional broadside tests to be generated for state-of-the-art circuits with features that are not handled by academic tool. The experimental results demonstrated that the generated test set achieved a fault coverage that is typically close to the one achieved by non-functional broadside tests. The difference in fault coverage is caused by the functional constraints that are necessary to avoid overtesting. The need to satisfy functional constraints also explains the increased number of tests. The experimental results also showed that generating multi-cycle functional broadside tests can reduce the need to find additional reachable states.

Next, a new procedure was presented to improve the resolution of multi-defect diagnosis by considering fewer tests. The procedure is implemented on top of a defect diagnosis tool that computes a partitioned set of candidate faults and takes advantage of the partition to avoid losing precision. The procedure has two phases. The first phase is based on the removal of subsets of tests. The second phase is based on the selection of subsets of tests. A subset-by-subset strategy is applied in the second phase to avoid a loss of precision while increasing the resolution. Experimental results indicated that the procedure results in reduced candidate fault sets. In addition, the procedure rarely loses the candidate faults that match the defects.

Last, we demonstrated that, among all the faults translated from DFM guideline violations, the undetectablle faults tend to cluster together and can result in areas of the circuit with poor coverage for potential systematic defects. In order to eliminate large clusters of undetectable faults, and improve the coverage of the circuit for potential systematic defects, two procedures based on logic and layout resynthesis were presented.

We first presented a procedure based on logic resynthesis followed by physical design to eliminate large clusters of undetectable internal faults caused by DFM guideline violations. The proposed logic resynthesis procedure has two phases. The first phase focuses on the largest clusters of undetectable faults. The second phase considers the entire circuit. A backtracking procedure was used to guarantee that the resynthesized circuit maintains design constraints of critical path delay, power consumption and die area. Next, we presented a layout resynthesis procedure to address the issue of undetectable external faults caused by DFM guideline violations. The layout resynthesis procedure uses a layout-based coverage metric, and makes fine changes to the layout while maintaining the design constraints. The procedure is iterative. In every iteration, it prefers to eliminate the undetectable faults whose neighborhoods have the lowest coverages. The undetectable faults are eliminated by fixing the DFM guideline violations that lead to them. The experimental results for benchmark circuits and logic blocks of the OpenSparc T1 microprocessor showed that both the improvement in the coverage of the circuit areas with low coverage and the reduction in the sizes of large clusters of undetectable faults are significant. Therefore, the coverage for potential systematic defects can be improved significantly.

REFERENCES

REFERENCES

[1] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition fault simulation," *IEEE Design Test of Computers*, vol. 4, no. 2, pp. 32–38, 1987.

[2] E. B. Eichelberger and T. W. Williams, "A logic design structure for lsi testability," in *Proceedings of the Design Automation Conference*, Jun 1977, pp. 462–468.

[3] J. Savir and S. Patil, "Broad-side delay test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1057–1064, 1994.

[4] J. Rearick, "Too much delay fault coverage is a bad thing," in *Proceedings of the International Test Conference*, Nov 2001, pp. 624–633.

[5] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger, "A case study of ir-drop in structured at-speed testing," in *Proceedings of the International Test Conference*, vol. 1, Sep. 2003, pp. 1098–1104.

[6] S. Sde-Paz and E. Salomon, "Frequency and power correlation between at-speed scan and functional tests," in *Proceedings of the International Test Conference*, Oct 2008, pp. 1–9.

[7] X. Liu and M. S. Hsiao, "Constrained atpg for broadside transition testing," in *Proceedings of the Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov 2003, pp. 175–182.

[8] I. Pomeranz and S. M. Reddy, "On application of output masking to undetectable faults in synchronous sequential circuits with design-for-testability logic," in *Proceedings of the International Conference on Computer Aided Design*, Nov 2003, pp. 867–872.

[9] I. Pomeranz, "On the generation of scan-based test sets with reachable states for testing under functional operation conditions," in *Proceedings of the Design Automation Conference*, July 2004, pp. 928–933.

[10] I. Pomeranz and S. M. Reddy, "Generation of functional broadside tests for transition faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2207–2218, 2006.

[11] ——, "On reset based functional broadside tests," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2010, pp. 1438–1443.

[12] Y. C. Lin, F. Lu, K. Yang, and K. T. Cheng, "Constraint extraction for pseudo-functional scan-based delay testing," in *Proceedings of the Asia and South Pacific Design Automation Conference*, vol. 1, Jan 2005, pp. 166–171 Vol. 1.

[13] Z. Zhang, S. M. Reddy, and I. Pomeranz, "On generating pseudo-functional delay fault tests for scan designs," in *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct 2005, pp. 398–405.

[14] M. Syal, K. Chandrasekar, V. Vimjam, M. S. Hsiao, Y. Chang, and S. Chakravarty, "A study of implication based pseudo functional testing," in *Proceedings of the International Test Conference*, Oct 2006, pp. 1–10.

[15] T. Zhang and D. M. Hank Walker, "Power supply noise control in pseudo functional test," in *Proceedings of the VLSI Test Symposium*, April 2013, pp. 1–6.

[16] J. A. Waicukauski and E. Lindbloom, "Failure diagnosis of structured vlsi," *IEEE Design Test of Computers*, vol. 6, no. 4, pp. 49–60, 1989.

[17] I. Pomeranz, "Improving the accuracy of defect diagnosis by considering fewer tests," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 2010–2014, 2014.

[18] ——, "Improving the accuracy of defect diagnosis with multiple sets of candidate faults," *IEEE Transactions on Computers*, vol. 65, no. 7, pp. 2332–2338, 2016.

[19] ——, "A test selection procedure for improving the accuracy of defect diagnosis," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 8, pp. 2759–2767, 2016.

[20] B. Kruseman, A. Majhi, C. Hora, S. Eichenberger, and J. Meirlevede, "Systematic defects in deep sub-micron technologies," in *Proceedings of the International Test Conference*, Oct 2004, pp. 290–299.

[21] C. Schuermyer, K. Cota, R. Madge, and B. Benware, "Identification of systematic yield limiters in complex asics through volume structural test fail data visualization and analysis," in *Proceedings of the International Test Conference*, Nov 2005, pp. 9 pp.–145.

[22] R. Turakhia, M. Ward, S. K. Goel, and B. Benware, "Bridging dfm analysis and volume diagnostics for yield learning - a case study," in *Proceedings of the VLSI Test Symposium*, May 2009, pp. 167–172.

[23] R. Desineni, L. Pastel, M. Kassab, M. F. Fayaz, and J. Lee, "Identifying design systematics using learning based diagnostic analysis," in *Proceedings of the Advanced Semiconductor Manufacturing Conference*, July 2010, pp. 317–321.

[24] S. Kundu and A. Sreedhar, "Modeling manufacturing process variation for design and test," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2011, pp. 1–6.

[25] B. Seshadri, P. Gupta, Y. T. Lin, and B. Cory, "Systematic defect screening in controlled experiments using volume diagnosis," in *Proceedings of the International Test Conference*, Nov 2012, pp. 1–7.

[26] M. Brodsky, S. Halle, V. Jophlin-Gut, L. Liebmann, D. Samuels, G. Crispo, K. Nafisi, V. Ramani, and I. Peterson, "Process-window sensitive full-chip inspection for design-tosilicon optimization in the sub-wavelength era," in *Proceedings of the IEEE/SEMI Conference and Workshop on Advanced Semiconductor Manufacturing*, April 2005, pp. 64–71.

[27] D. Kim, M. E. Amyeen, S. Venkataraman, I. Pomeranz, S. Basumallick, and B. Landau, "Testing for systematic defects based on dfm guidelines," in *Proceedings of the International Test Conference*, Oct 2007, pp. 1–10.

[28] D. Kim, I. Pomeranz, M. E. Amyeen, and S. Venkataraman, "Defect diagnosis based on dfm guidelines," in *Proceedings of the VLSI Test Symposium*, April 2010, pp. 206–211.

[29] A. Sinha, S. Pandey, A. Singhal, A. Sanyal, and A. Schmaltz, "Dfm-aware fault model and atpg for intra-cell and inter-cell defects," in *Proceedings of the International Test Conference*, Oct 2017, pp. 1–10.

[30] H. Zheng, K. K. Saluja, and R. Jain, "Test application time reduction for scan based sequential circuits," in *Proceedings of the Great Lakes Symposium on VLSI*, March 1995, pp. 188–191.

[31] I. Park and E. J. McCluskey, "Launch-on-shift-capture transition tests," in *Proceedings of the International Test Conference*, Oct 2008, pp. 1–9.

[32] I. Pomeranz, "Generation of multi-cycle broadside tests," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1253–1257, Aug 2011.

[33] X. Lin and R. Thompson, "Test generation for designs with multiple clocks," in *Proceedings of the Design Automation Conference*, June 2003, pp. 662–667.

[34] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli, "High-frequency, at-speed scan testing," *IEEE Design Test of Computers*, vol. 20, no. 5, pp. 17–25, 2003.

[35] [Online]. Available: http://www.opencores.org/

[36] [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/index.html

[37] G. Chen, S. M. Reddy, and I. Pomeranz, "Procedures for identifying untestable and redundant transition faults in synchronous sequential circuits," in *Proceedings of the International Conference on Computer Design*, Oct 2003, pp. 36–41.

[38] *Digital Systems Testing and Testable Design.*

[39] D. B. Lavo, B. Chess, T. Larrabee, and I. Hartanto, "Probabilistic mixed-model fault diagnosis," in *Proceedings of the International Test Conference*, Oct 1998, pp. 1084–1093.

[40] J. Ghosh-Dastidar and N. A. Touba, "Adaptive techniques for improving delay fault diagnosis," in *Proceedings of the VLSI Test Symposium*, April 1999, pp. 168–172.

[41] S. Venkataraman and S. B. Drummonds, "Poirot: a logic fault diagnosis tool and its applications," in *Proceedings of the International Test Conference*, Oct 2000, pp. 253–262.

[42] S. Y. Huang, "On improving the accuracy of multiple defect diagnosis," in *Proceedings of the VLSI Test Symposium*, April 2001, pp. 34–39.

[43] T. Bartenstein, D. Heaberlin, L. Huisman, and D. Sliwinski, "Diagnosing combinational logic designs using the single location at-a-time (slat) paradigm," in *Proceedings of the International Test Conference*, Nov 2001, pp. 287–296.

[44] D. B. Lavo, I. Hartanto, and T. Larrabee, "Multiplets, models, and the search for meaning: improving per-test fault diagnosis," in *Proceedings of the International Test Conference*, Oct 2002, pp. 250–259.

[45] Z. Wang, M. Marek-Sadowska, K. H. Tsai, and J. Rajski, "Multiple fault diagnosis using n-detection tests," in *Proceedings of the International Conference on Computer Design*, Oct 2003, pp. 198–201.

[46] I. Pomeranz, S. Venkataraman, S. M. Reddy, and E. Amyeen, "Defect diagnosis based on pattern-dependent stuck-at faults," in *Proceedings of the International Conference on VLSI Design*, Jan 2004, pp. 475–480.

[47] J. B. Liu and A. Veneris, "Incremental fault diagnosis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 240–251, 2005.

[48] R. Desineni and R. D. Blanton, "Diagnosis of arbitrary defects using neighborhood function extraction," in *Proceedings of the VLSI Test Symposium*, May 2005, pp. 366–373.

[49] C. Liu, "Improve the quality of per-test fault diagnosis using output information," *Journal of Electronic Testing*, vol. 23, no. 1, pp. 11–24, 2007.

[50] R. Adapa, S. Tragoudas, and M. K. Michael, "Accelerating diagnosis via dominance relations between sets of faults," in *Proceedings of the VLSI Test Symposium*, May 2007, pp. 219–224.

[51] S. Holst and H. Wunderlich, "Adaptive debug and diagnosis without fault dictionaries," in *Proceedings of the European Test Symposium*, May 2007, pp. 7–12.

[52] W. C. Tam, O. Poku, and R. D. Blanton, "Precise failure localization using automated layout analysis of diagnosis candidates," in *Proceedings of the Design Automation Conference*, June 2008, pp. 367–372.

[53] X. Yu and R. D. Blanton, "An effective and flexible multiple defect diagnosis methodology using error propagation analysis," in *Proceedings of the International Test Conference*, Oct 2008, pp. 1–9.

[54] W. Cheng, B. Benware, R. Guo, K. Tsai, T. Kobayashi, K. Maruo, M. Nakao, Y. Fukui, and H. Otake, "Enhancing transition fault model for delay defect diagnosis," in *Proceedings of the Asian Test Symposium*, Nov 2008, pp. 179–184.

[55] V. J. Mehta, M. Marek-Sadowska, K. Tsai, and J. Rajski, "Timing-aware multiple-delay-fault diagnosis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 245–258, 2009.

[56] X. Tang, W. Cheng, R. Guo, and S. M. Reddy, "Diagnosis of multiple physical defects using logic fault models," in *Proceedings of the Asian Test Symposium*, Dec 2010, pp. 94–99.

[57] I. Pomeranz, "Obo: An output-by-output scoring algorithm for fault diagnosis," in *Proceedings of the Computer Society Annual Symposium on VLSI*, July 2014, pp. 314–319.

[58] P. G. Ryan, W. K. Fuchs, and I. Pomeranz, "Fault dictionary compression and equivalence class computation for sequential circuits," in *Proceedings of the International Conference on Computer Aided Design*, Nov 1993, pp. 508–511.

[59] H. Wang, O. Poku, X. Yu, S. Liu, I. Komara, and R. D. Blanton, "Test-data volume optimization for diagnosis," in *Proceedings of the DAC Design Automation Conference*, June 2012, pp. 567–572.

[60] X. Fan, H. Tang, Y. Huang, W. Cheng, S. M. Reddy, and B. Benware, "Improved volume diagnosis throughput using dynamic design partitioning," in *Proceedings of the International Test Conference*, Nov 2012, pp. 1–10.

[61] S. Kundu, P. Bhattacharya, and R. Kapur, "Fault diagnosis in designs with extreme low pin test data compressors," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2015, pp. 1285–1288.

[62] I. Pomeranz and S. M. Reddy, "On clustering of undetectable single stuck-at faults and test quality in full-scan circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1135–1140, 2010.

[63] I. Pomeranz, "On clustering of undetectable transition faults in standard-scan circuits," in *Proceedings of the VLSI Test Symposium*, May 2011, pp. 128–133.

[64] P. Pan, "Performance-driven integration of retiming and resynthesis," in *Proceedings of the Design Automation Conference*, June 1999, pp. 243–246.

[65] V. N. Kravets and K. A. Sakallah, "Resynthesis of multi-level circuits under tight constraints using symbolic optimization," in *Proceedings of the International Conference on Computer Aided Design*, Nov 2002, pp. 687–693.

[66] A. Saifhashemi, D. Hand, P. A. Beerel, W. Koven, and H. Wang, "Performance and area optimization of a bundled-data intel processor through resynthesis," in *Proceedings of the International Symposium on Asynchronous Circuits and Systems*, May 2014, pp. 110–111.

[67] S. Chiu and C. A. Papachristou, "A design for testability scheme with applications to data path synthesis," in *Proceedings of the Design Automation Conference*, June 1991, pp. 271–277.

[68] T. C. Lee, W. H. Wolf, and N. K. Jha, "Behavioral synthesis for easy testability in data path scheduling," in *Proceedings of the International Conference on Computer-Aided Design*, Nov 1992, pp. 616–619.

[69] S. Kanjilal, S. T. Chakradhar, and V. D. Agrawal, "A partition and resynthesis approach to testable design of large circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 10, pp. 1268–1276, 1995.

[70] I. Pomeranz and S. M. Reddy, "On the number of tests to detect all path delay faults in combinational logic circuits," *IEEE Transactions on Computers*, vol. 45, no. 1, pp. 50–62, 1996.

[71] A. Krstic and K. Cheng, "Resynthesis of combinational circuits for path count reduction and for path delay fault testability," in *Proceedings of the European Design and Test Conference*, March 1996, pp. 486–490.

[72] K. D. Wagner and S. Dey, "High-level synthesis for testability: a survey and perspective," in *Proceedings of the Design Automation Conference*, June 1996, pp. 131–136.

[73] J. C. M. Li, Chao-Wen Tseng, and E. J. McCluskey, "Testing for resistive opens and stuck opens," in *Proceedings of the International Test Conference*, Nov 2001, pp. 1049–1058.

[74] Z. Li, X. Lu, W. Qiu, W. Shi, and D. M. H. Walker, "A circuit level fault model for resistive opens and bridges," in *Proceedings of the VLSI Test Symposium*, May 2003, pp. 379–384.

[75] V. Krishnaswamy, A. B. Ma, and P. Vishakantaiah, "A study of bridging defect probabilities on a pentium (tm) 4 cpu," in *Proceedings of the International Test Conference*, Nov 2001, pp. 688–695.

[76] A. Salz and M. Horowitz, "Irsim: An incremental mos switch-level simulator," in *Proceedings of the Design Automation Conference*, June 1989, pp. 173–178.

[77] J. E. Stine, J. Grad, I. Castellanos, J. Blank, V. Dave, M. Prakash, N. Iliev, and N. Jachimiec, "A framework for high-level synthesis of system on chip designs," in *Proceedings of the International Conference on Microelectronic Systems Education*, June 2005, pp. 67–68.

[78] C. Guardiani, N. Dragone, and P. McNamara, "Proactive design for manufacturing (dfm) for nanometer soc designs," in *Proceedings of the Custom Integrated Circuits Conference*, Oct 2004, pp. 309–316.

[79] A. Nardi and A. L. Sangiovanni-Vincentelli, "Synthesis for manufacturability: a sanity check," in *Proceedings of the Design, Automation and Test in Europe Conference*, Feb 2004, pp. 796–801.

[80] N. Wang, I. Pomeranz, S. M. Reddy, A. Sinha, and S. Venkataraman, "Resynthesis for avoiding undetectable faults based on design-for-manufacturability guidelines," in *Proceedings of the Design, Automation and Test in Europe Conference*, March 2019, pp. 1022–1027.

[81] P. Maxwell, F. Hapke, M. Ryynnen, and P. Weseloh, "Bridge over troubled waters: Critical area based pattern generation," in *Proceedings of the European Test Symposium*, May 2017, pp. 1–6.

[82] W. Howell, F. Hapke, E. Brazil, S. Venkataraman, R. Datta, A. Glowatz, W. Redemund, J. Schmerberg, A. Fast, and J. Rajski, "Dppm reduction methods and new defect oriented test methods applied to advanced finfet technologies," in *Proceedings of the International Test Conference*, Oct 2018, pp. 1–10.

[83] I. Pomeranz and S. M. Reddy, "On achieving complete coverage of delay faults in full scan circuits using locally available lines," in *Proceedings of the International Test Conference*, Sept 1999, pp. 923–931.

[84] S. Wang, X. Liu, and S. T. Chakradhar, "Hybrid delay scan: a low hardware overhead scan-based delay test technique for high fault coverage and compact test sets," in *Proceedings of the Design, Automation and Test in Europe Conference*, Feb 2004, pp. 1296–1301.

[85] N. Devtaprasanna, A. Gunda, P. Krishnamurthy, S. M. Reddy, and I. Pomeranz, "Methods for improving transition delay fault coverage using broadside tests," in *Proceedings of the International Test Conference*, Nov 2005, pp. 10 pp.–265.

[86] S. Wang and T. Yeh, "High-level test synthesis for delay fault testability," in *Proceedings of the Design, Automation and Test in Europe Conference*, April 2007, pp. 1–6.

[87] B. Krishnamurthy, "A dynamic programming approach to the test point insertion problem," in *Proceedings of the Design Automation Conference*, June 1987, pp. 695–705.

[88] J. Yang, N. A. Touba, and B. Nadeau-Dostie, "Test point insertion with control points driven by existing functional flip-flops," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1473–1483, 2012.

[89] E. Moghaddam, N. Mukherjee, J. Rajski, J. Tyszer, and J. Zawada, "Test point insertion in hybrid test compression/lbist architectures," in *Proceedings of the International Test Conference*, Nov 2016, pp. 1–10.

[90] C. Acero, D. Feltham, F. Hapke, E. Moghaddam, N. Mukherjee, V. Neerkundar, M. Patyra, J. Rajski, J. Tyszer, and J. Zawada, "Embedded deterministic test points for compact cell-aware tests," in *Proceedings of the International Test Conference*, Oct 2015, pp. 1–8.

[91] Y. Liu, E. Moghaddam, N. Mukherjee, S. M. Reddy, J. Rajski, and J. Tyszer, "Minimal area test points for deterministic patterns," in *Proceedings of the International Test Conference*, Nov 2016, pp. 1–7.

[92] S. Ravi and M. Joseph, "High-level test synthesis: A survey from synthesis process flow perspective," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 4, pp. 38:1–38:27, 2014.

[93] C. A. Papachristou, S. Chiu, and H. Harmanani, "A data path synthesis method for self-testable designs," in *Proceedings of the Design Automation Conference*, 1991, pp. 378–384.

[94] I. Pomeranz and S. M. Reddy, "On synthesis-for-testability of combinational logic circuits," in *Proceedings of the Design Automation Conference*, 1995, pp. 126–132.

[95] I. Pomeranz, "Design-for-testability for multi-cycle broadside tests by holding of state variables," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, pp. 19:1–19:20, 2014.

[96] ——, "Enhanced test compaction for multicycle broadside tests by using state complementation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 1, pp. 13:1–13:20, 2015.

[97] S. Wang, K. Peng, K. Hsiao, and K. S. Li, "Layout-aware scan chain reorder for launch-off-shift transition test coverage," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 4, pp. 64:1–64:16, 2008.

[98] P. Gupta, A. B. Kahng, I. Mandoiu, and P. Sharma, "Layout-aware scan chain synthesis for improved path delay fault coverage," in *Proceedings of the International Conference on Computer-Aided Design*, Nov 2003, pp. 754–759.

[99] S. Wang, K. Peng, and K. S. Li, "Layout-aware scan chain reorder for skewed-load transition test coverage," in *Proceedings of the Asian Test Symposium*, Nov 2006, pp. 169–174.

[100] S. Kajihara, T. Sumioka, and K. Kinoshita, "Test generation for multiple faults based on parallel vector pair analysis," in *Proceedings of the International Conference on Computer-Aided Design*, Nov 1993, pp. 436–439.

[101] M. Fujita and A. Mishchenko, "Efficient sat-based atpg techniques for all multiple stuck-at faults," in *Proceedings of the International Test Conference*, Oct 2014, pp. 1–10.

[102] I. Pomeranz and S. M. Reddy, "On multiple bridging faults," in *Proceedings of the VLSI Test Symposium*, April 2010, pp. 221–226.

[103] T. Jhaveri, A. Strojwas, L. Pileggi, and V. Rovner, "Enabling technology scaling with "in production" lithography processes," in *Proceedings of the SPIE*, vol. 6924, 2008, pp. 6924 – 6924 – 10.

[104] C. Tabery, M. Craig, G. Burbach, B. Wagner, S. McGowan, P. Etter, S. Roling, C. Haidinyak, and E. Ehrichs, "Process window and device variations evaluation using array-based characterization circuits," in *Proceedings of the International Symposium on Quality Electronic Design*, March 2006, pp. 6 pp.–265.

[105] W. C. Tam and S. Blanton, "To dfm or not to dfm?" in *Proceedings of the Design Automation Conference*, June 2011, pp. 65–70.

[106] R. D. Blanton, F. Wang, C. Xue, P. K. Nag, Y. Xue, and X. Li, "Dreams: Dfm rule evaluation using manufactured silicon," in *Proceedings of the International Conference on Computer-Aided Design*, Nov 2013, pp. 99–106.

VITA

VITA

Naixing Wang received his B.Eng. degree in Electrical & Electronics Engineering from Beijing Institute of Technology, Beijing, China in 2014. Since 2014, he has been at the school of Electrical and Computer Engineering, Purdue University, West Lafayette, USA, pursuing a Ph.D. degree.

His current research interests include Design-for-testability (DFT), test generation, defect diagnosis, logic synthesis, design-for-manufacturability (DFM) and EDA tool development.

PUBLICATIONS

1. N. Wang, B. Yao, X. Lin and I. Pomeranz, "Functional Broadside Test Generation Using a Commercial ATPG Tool," in *Proceedings of the IEEE Computer Society Annual Symp. on VLSI*, Jul 2017, pp. 308-313.
2. N. Wang, I. Pomeranz, B. Benware, M. E. Amyeen and S. Venkataraman, "Improving the Resolution of Multiple Defect Diagnosis by Removing and Selecting Tests," in *Proceedings of the Defect and Reliability Symp.*, Oct 2018, pp. 1-6. (*Best paper award nominee*)
3. N. Wang, I. Pomeranz, S. M. Reddy, A. Sinha and S. Venkataraman, "Resynthesis for Avoiding Undetectable Faults Based on Design-for-Manufacturability Guidelines," in *Proceedings of the Design, Automation and Test in Europe Conference*, Mar 2019, pp. 1022-1027.
4. N. Wang, I. Pomeranz, S. M. Reddy, A. Sinha and S. Venkataraman, "Layout Resynthesis by Applying Design-for-Manufacturability Guidelines to Avoid Low-Coverage Areas of a Cell-Based Design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 4, pp. 42:1-42:19, 2019.