REDUCING DIGITAL TEST VOLUME USING TEST POINT INSERTION

BY RAJAMANI SETHURAM

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Prof. Michael L. Bushnell

and approved by

____

____

____

____

____

____

New Brunswick, New Jersey

January, 2008

ABSTRACT OF THE DISSERTATION

Reducing Digital Test Volume Using Test Point Insertion

by Rajamani Sethuram

Dissertation Director: Prof. Michael L. Bushnell

Test cost accounts for more than 40% of the entire cost for making a chip. This figure is expected to grow even higher in the future. Two major factors that determine test cost are a) test volume and b) test application time. Several techniques such as compaction and compression have been proposed in the past to keep the test cost under an acceptable limit. However, due to the ever increasing size of today's digital very large scale integrated circuits, all prior known test cost reduction techniques are unable to keep the test cost under control.

In this dissertation, we present a new *test point insertion* (TPI) technique for regular cell-based *application specific integrated chips* (ASICs) and structured ASIC designs. The proposed technique can drastically reduce the test volume, the test application time and the test generation time. The TPI scheme facilitates the compression and the compaction algorithm to reduce test volume and test application time. By facilitating the *automatic test pattern generation* (ATPG) algorithm, we also reduce the test generation time. Test points are inserted using timing information, so they do not degrade performance. We present novel gain functions that quantify the reduction in test volume and ATPG time due to TPI

and are used as heuristics to guide the selection of signal lines for inserting test points. We, then, show how test point insertion can be used to enhance the performance of a broadcast scan-based compressor. To further improve its performance, we use a new scan chain re-ordering algorithm to break the correlation that exists among different signal lines in the circuit due to a particular scan chain order. Experiments conducted with ISCAS '89, ITC '99, and few industrial benchmarks clearly demonstrate the effectiveness and scalability of the proposed technique. By using very little extra hardware for implementing test points and very little extra run time for the TPI step, we show that the test volume and test application can be reduced by up to 64.5% and test generation time can be reduced by up to 63.1% for structured ASIC designs. For the cell-based ASICs with broadcast scan compressors, experiments indicate that the proposed technique improves the compression by up to 46.6% and also reduces the overall ATPG CPU time by up to 49.3%.

Acknowledgements

First, I thank my advisor Prof. Michael L. Bushnell for his support and guidance throughout my stay at Rutgers University. I wish to thank NEC Laboratories, America for providing us financial support and the software tools for conducting this research. I especially thank Dr. Srimat T. Chakradhar of NEC Laboratories for providing us this research topic. I thank Dr. Seongmoon Wang of NEC Laboratories for several suggestions provided that tremendously improved the quality of this work. His critical reviews about out work are greatly acknowledged. I also thank the committee members Prof. Peter Meer, Prof. Manish Parashar, and Prof. Lawrence Rabiner, of Rutgers University, Dr. Srimat T. Chakradhar of NEC Laboratories America, and Dr. Tapan J. Chakroborty of Bell Laboratories, America for taking their precious time to be on my committee. I thank all of my friends, past and present, who constantly encouraged me throughout the course of this research.

Last, but not the least, I wish to thank my family members for their encouragement, understanding, and love that helped me complete this research work successfully.

Dedication

To my late father Mr. S. Sethuramalingam.

Table of Contents

Abstract ii

Acknowledgements iv

Dedication v

List of Tables x

List of Figures xi

1. Introduction 1

1.1. The Structured ASIC Methodology 2

1.2. The Cell-based ASIC Methodology 4

1.3. Motivation 4

1.4. Contributions of the Dissertation 6

1.5. Organization of the Dissertation 7

2. Prior Work 9

2.1. Test Data Compaction 9

2.2. Built-in Self Testing 10

2.3. Input Test Data Compression 11

2.3.1. Fixed-to-fixed Code 12

2.3.2. Fixed-to-variable Code 13

2.3.3. Variable-to-fixed Code 15

2.3.4. Variable-to-variable Code 17

2.4. Scan Architectures for Test Volume and Test Application Time Reduction 19

2.4.1. Illinois Scan Architecture 20
2.5. Test Point Insertion 23
2.5.1. Test Point Insertion for BIST 24
2.5.2. Test Point Insertion for Non-BIST Designs 28
2.5.2.1. Test Point Insertion to Reduce Test Volume . . . 29

3. Observation Points 32
3.1. Introduction 32
3.2. The Algorithm to Insert Observation Points 33
3.2.1. The Constraints 34
3.2.2. Gain Function 36
3.2.2.1. Computing $N_i(l)$ 38
3.2.2.2. Computing N_f 42
3.2.3. Mergeability 43
3.2.4. Updating Gain Values 45
3.3. Algorithm Outline 46
3.4. Complexity Analysis 47
3.5. Summary 48

4. Control Points 49
4.1. Introduction 49
4.2. Control Points and their Implementation 50
4.2.1. Conventional Control Point (CP) 50
4.2.2. Complete Test Point (CTP) 52
4.2.3. Pseudo Control Point (PCP) 53
4.2.4. Inversion Test Point (ITP) 55
4.3. Algorithm 57
4.3.1. Controllability and Observability Cost 57
4.3.2. Computing N_f (l), F_C^v (l), and F_O^v (l) 58
4.3.2.1. Computing N_f (l) 59

4.3.2.2. Computing $F_C^v(l)$ and $F_O^v(l)$ 59
4.3.3. Updating Testability Measures 59
4.3.4. Overall Algorithm 60
4.4. Complexity Analysis 61
4.5. Summary 62

5. Test Points for a Hardware Compressor 63
5.1. Introduction 63
5.2. The Proposed Design-for-Test Scheme 64
5.3. Our Test Point Insertion Procedure 67
5.3.1. The Test Point Hardware 69
5.3.2. Identifying Correlated Flip-flops 69
5.3.3. Controllability and Observability Cost 72
5.3.4. Computing $N_f(l)$, $F_C^v(l)$, and $F_O^v(l)$ 74
5.3.4.1. Computing $N_f(l)$ 74
5.3.4.2. Computing $F_C^v(l)$ and $F_O^v(l)$ 74
5.3.5. Updating Testability Measures 75
5.3.6. The Overall TPI Algorithm 75
5.4. Scan Chain Re-ordering Technique 76
5.4.1. Build Neighborhood Information 78
5.4.2. Slice Correlation 79
5.4.3. Computing the Gain Value 79
5.4.4. Updating Gain Values 80
5.4.5. The Overall Algorithm 81
5.5. Complexity Analysis 81
5.6. Summary 82

6. Results 84
6.1. Structured ASICs 84
6.1.1. Inserting Observation Points 84

6.1.2. Inserting Control Points 87
6.1.2.1. Comparison with Prior Work 89
6.2. Cell-based ASICs 90
6.2.1. Test Point Insertion 90
6.2.2. Scan Chain Re-ordering 92
6.2.3. The Overall DFT Results 92
6.3. Summary 94
7. Conclusions and Future Work 95
7.1. Conclusion 95
7.2. Future Work 96
Appendix A. User's Guide 98
A.1. Directory 98
A.2. Command Synopsis 98
References 101
Vita 107

List of Tables

2.1. Statistical Coding Based on Symbol Frequencies for Test Set in Figure 2.2 14
2.2. Example of Run length Code 16
2.3. Example of Golomb Code 18
2.4. Example of *Frequency Directed Run Length* (FDR) Code 19
3.1. Bitwise ORing with SCOAP-like Measure Comparison 42
3.2. Example to Describe Mergeability 44
6.1. Design Characteristics 85
6.2. Experimental Results 85
6.3. Test Vector Length Reduction and CPU Time with TPI with Near 100% FE 88
6.4. Test Vector Length Reduction during ATPG as a Function of # and Type of Test Points Inserted 89
6.5. Comparison with Previous Work (All Achieve Near 100% FE) . . 90
6.6. Results for Inserting Complete Test Points in Broadcast Scan . . 91
6.7. Results for the Broadcast Scan Chain Re-ordering Operation . . . 92
6.8. The Overall DFT Results 93
6.9. CPU Time and Peak Memory of our Tool 94

List of Figures

1.1. Block Diagram of Structured ASIC: a) The ISSP Architecture, b) Complex Multi-gate, and c) NAND Gate Implementation in ISSP 3
2.1. Block Diagram of a BIST System 11
2.2. Example of Test Data 14
2.3. Huffman Tree for the Code Shown in Table 2.1 15
2.4. Huffman Tree for the Three Highest Frequency Symbols in Table 2.1 15
2.5. Cyclical Scan Chain: a) Decompression Architecture and b) Using Boundary Scan 17
2.6. Parallel Serial Full Scan Architecture 21
2.7. Broadcast Scan: a) Serial Mode and b) Parallel Mode 22
2.8. Example Illustrating Test Points: a) Observation Point, b) 1-Control Point, and c) 0-Control Point 23
2.9. Control Points Driven by Pattern Decoding Logic 26
2.10. BIST with the Multi-phase Test Point Scheme 27
2.11. Example to Illustrate Test Counts 29
2.12. Breaking Fanout Free Regions (FFRs): a) A Large FFR and b) Smaller FFRs 31
3.1. Used and Unused CMG Cells 34
3.2. Example of Circuit with Three Logic Cones C_1, C_2, and C_3 36
3.3. Accurate Computation for Reconverging Signals 38
3.4. Dividing the Circuit into a Fanout Free Region (FFR) Network . 41
3.5. Pseudo-code for Accurate Computation of $N_i(l)$ 41
3.6. Illustration for Mergeability 44

3.7. Updating the Gain Value: a) Fanin Cone and Fanout Cone and b) Overlapping Fanin Cones 45
4.1. Conventional Control Point 50
4.2. Complete Test Point 52
4.3. Pseudo Control Point 53
4.4. A 3-input LUT Implementing $F{=}A\overline{B}C + AB\overline{C}$ 55
4.5. Inversion Test Point 56
4.6. Pseudo-code for the Overall Algorithm 61
5.1. Artificially Untestable Faults 65
5.2. Example Illustrating Correlated SFFs 68
5.3. An Optimized Complete Test Point Highlighting the Operation for: a) Test Mode and b) Functional Mode 70
5.4. Algorithm to Identify Correlated Flip-flops 71
5.5. The Overall TPI Algorithm 76
5.6. Swap Operation: a) Old and b) New Order 77
5.7. Building Neighborhood Information 78
5.8. The Algorithm for Updating the Gain Values 80
5.9. The Overall Re-ordering Algorithm 81
6.1. Results for Inserting 100, 500, and 1000 TPs 86
6.2. ATPG vs. Compaction Time 87
6.3. Proposed Technique vs. SCOAP-like Measure 87
6.4. The ATPG Backtrack Count 93
6.5. The Compactor Performance 94
A.1. Scan Group and Scan Chain Information File 99

Chapter 1
Introduction

We live in a very exciting era of electronic design and automation. The electronics and computer revolution that took place in the 80's and the 90's has changed the computing device from an artifact in a scientific research laboratory into an integral and indispensable part of a common man's life. Such is the impact of this revolution that today, teenagers find it difficult to imagine a life without their cell phone that comes with an integrated digital video camera, internet browsing capability, *global positioning system* (GPS), data communication features, a gamut of entertainment features such as streaming radio and television channels, and advanced gaming along with its basic ability to place phone calls. The tremendous advancement in electronic systems is attributed to the successful research and development work conducted in the last few decades. Looking at the pace and intensity of the ongoing research activities in this field, it leaves little doubt that the electronic industry is going to provide us even more returns in the future, taking the quality of a common man's life to new heights.

To meet the requirements for building such complex electronic system while also keeping the price of these hi-tech devices affordable, designers have proposed a spectrum of design methodologies. On the one end of this spectrum lies the cell-based *application specific integrated chip* (ASIC) design method used to manufacture large volume chip production for high speed and low power devices. However, the up-front *non-recurring engineering* (NRE) cost for cell-based ASICs is very high because they use several custom metal layers and making masks for all layers costs over 1 million dollars. Another major disadvantage of this methodology is the enormous *turn-around time* (typically more than a year)

requirement, which is defined as the total time required to convert the design specification into a manufactured device. The other end of the spectrum is the *field programmable gate array* (FPGA) technology used for low volume chip production. The FPGA technique solves the *turn-around time* (TAT) problem but the disadvantages of the FPGAs are the increased power consumption and the much lower operating clock frequency. Recently, a new design methodology called *structured ASICs* (SAs) has emerged as a sweet spot between the cell-based ASIC methodology and the FPGA technology. In this dissertation, we present novel *design-for-test* (DFT) techniques to reduce the test cost for the structured ASIC (see Chapters 3 and 4) and the regular cell-based ASIC design methodology (see Chapter 5). These two design methodologies are described in detail below.

1.1 The Structured ASIC Methodology

As mentioned earlier, SAs have emerged as a new design paradigm that borrows a few concepts from the cell-based ASIC methodology and a few concepts from the FPGA methodology. This enables SAs to consume less power than FPGAs and incur less *non-recurring engineering* (NRE) cost with shorter TAT compared to cell-based ASICs. Many SAs have been proposed recently both in the industry [44] as well as in academia [19, 30, 42, 46].

SAs use a pre-defined array of macro cells organized in a 2-dimensional grid fashion. These pre-defined macro cells are normally realized using a limited number of metal or via layers (typically 2-3), which are not field-programmable. A thoroughly verified clock tree, DFT hardware and other generic *intellectual property* (IP) cores such as *serializer-deserializer* (SerDes), Memory, *analog phase locked loops* (APLLs), and *delay locked loops* (DLL), are pre-defined and need not be designed by users. Hence, using SAs eliminates the design time involved in clock tree synthesis, testability related issues and obtaining IP cores. Another advantage of using these pre-defined layers is that the physical design of these layers is fully verified for timing and signal integrity, which in turn, makes the

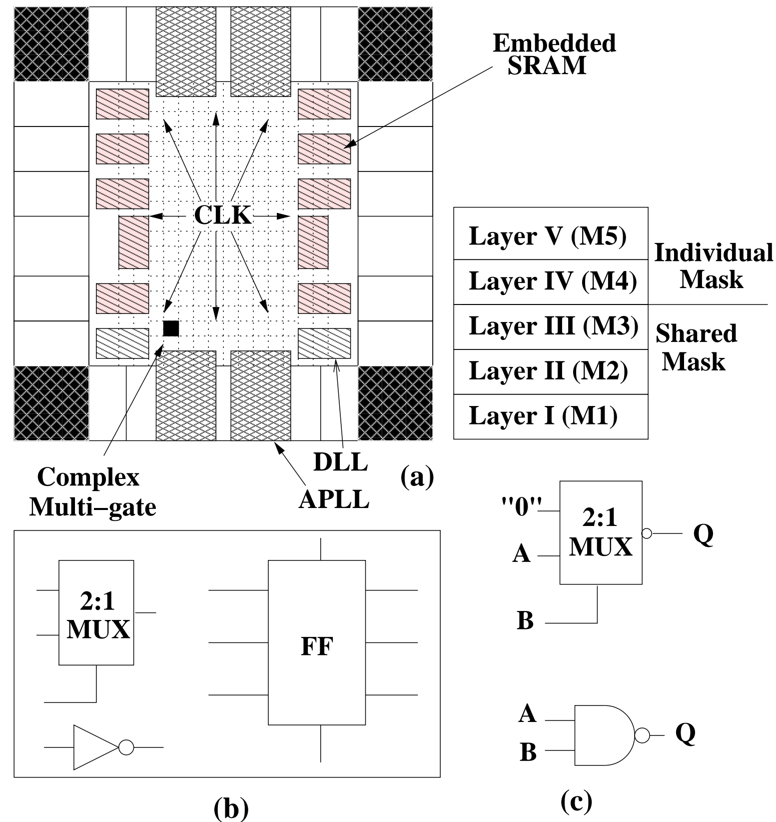


Figure 1.1: Block Diagram of Structured ASIC: a) The ISSP Architecture, b) Complex Multi-gate, and c) NAND Gate Implementation in ISSP

total design cycle time of an SA significantly shorter than that of a cell-based ASIC. The size and complexity of the pre-defined cells vary with the architecture employed by the SAs [69, 74]. As an example, the block diagram of NEC's structured ASIC called the *Instant Silicon Solution Platform* (ISSP) is given in Figure 1.1(a). Each pre-defined macro cell is called a *complex multi-gate* (CMG) in the ISSP, and comprises a multiplexor, inverters, and a flip-flop as shown in Figure 1.1(b). User functions are realized using the components in a CMG cell by programming the interconnects between them, as in FPGAs. For example, Figure 1.1(c) shows how to realize a NAND gate using the constituent multiplexor in the CMG cell of the ISSP. The number of programmable inter-connect layers is limited to one or two. Therefore, only a few custom masks are required and the NRE cost can be drastically reduced.

1.2 The Cell-based ASIC Methodology

In a cell-based ASIC design flow, a large number of *cells* (commonly used hardware blocks such as primitive gates, full adders, scan flip-flops, etc.) are custom designed and are saved as a library called the *technology library*. The technology library is then used to build the entire chip. From the DFT viewpoint, there are two important considerations. First, ASICs are used for high performance and high volume chip production. Since the test volume is typically very large for these designs, hardware test compression schemes (see Chapter 2) are used. In Chapter 5, we describe a novel DFT scheme to reduce the test volume in the presence of a hardware compressor. Second, additional hardware will have to be added to build any new ad hoc DFT structure. Hence, we should consider the extra cost incurred (e.g., increase in scan chain length) due to the DFT structure while quantifying the benefits associated with any DFT solution for cell-based ASICs.

1.3 Motivation

Designers are presenting different design techniques and methodologies to address the growing demands of high levels of integration while keeping the TAT low. However, on the negative side, the test engineers are finding it extremely difficult to test the entire chip in a cost effective way. Today, test cost accounts for more than 40% of the entire cost for making a chip [13]. This figure is expected to grow even higher in the future. Test engineers have to construct large numbers of test inputs that cover all possible defects that could possibly manifest themselves in the device. Constructing these test inputs consumes enormous CPU time and can delay the *time-to-market* (TTM). Large test inputs also means storing a large amount of test data in the tester memory and transferring this data from the tester to the *circuit under test* (CUT). Present-day testers work at a much slower clock rate than the CUT, so a significant amount of time is also required to test the CUT. This time, also known as the *test application time* (TAT), adds

to the product turn-around time.

The high demands of ultra large scale integration are fueling a new *deep submicron* (DSM) phenomenon that is further aggravating the testing problem. DSM refers to the continuous shrinking of the transistor size that has now become a fraction of a micrometer ($\times 10^{-6}$ meters). To integrate a large number of applications into one system, designers are continuously trying to reduce the size of the transistor to keep the silicon area constant. For example, Samsung Electronics has started production in the 35 nanometer ($\times 10^{-9}$ meters) node to roll out their recent flash memory products and NEC electronics has successfully fabricated transistors with a 5 nanometer wide channel and is planning mass volume production of these in the future. However, these DSM devices also come with an array of new types of defects such as crosstalk noise, small delay defects, voltage drops on power rails, and substrate noise. Hence, novel defect-oriented fault models are required to detect these defects. To identify these defects, test engineers are required to use more test inputs to thoroughly test the chip. This explains how DSM is severely affecting test volume and test application time.

The *International Technology Roadmap for Semiconductors* (ITRS) has predicted that major breakthroughs in the design-for-test area are required in the next two decades to address the growing test problem. Hence, we present a new test point insertion technique that can alleviate the growing test cost by reducing test volume, test application time, and the test generation time. This doctoral dissertation describes some of the recent techniques used for test volume reduction such as compaction and compression (described in Chapter 2) and how test point insertion can facilitate these technologies to drastically reduce test volume. We will now summarize the key motivation for our research work.

1. As mentioned earlier, test cost is sky rocketing due to the increased test volume and test application time requirement for testing these very large designs. Currently, test set compaction and hardware test compression are used to reduce the test volume to keep the test cost under acceptable limits.

2. Several prior works have presented test point insertion algorithms in conjunction with the *built-in self test* [43, 59, 62, 65] hardware to improve the random pattern testability of the design, but not much work have been reported in the literature that specifically focuses on reducing test volume by inserting test points into the design. Our TPI technique understands the compaction tool and the hardware compression algorithm to effectively reduce the test volume in the presence of these techniques.

3. To the best of our knowledge, no prior work has studied the impact of the test point insertion step on the overall design cycle. We present TPI algorithms that also minimizes the overall design TAT. We also present efficient algorithms to calculate our testability measures that can drastically reduce test volume and the test application time.

1.4 Contributions of the Dissertation

Following are the major contributions of this dissertation.

1. This dissertation describes the first accurate testability measure that facilitates the ATPG and the compaction algorithms to reduce the test volume. All prior work uses SCOAP [21] or COP-based [11] testability measures to insert test points for improving the fault coverage but these measures are inaccurate because of correlation of digital signals. However, in this dissertation we have developed a theory that describes how TPI can reduce test volume and test generation time by facilitating ATPG and compaction algorithms.

2. We present the first zero-cost TPI technique for SAs. We show how the unused SFFs in a SA design can be implemented as test points to reduce the test volume and the test generation time.

3. For regular cell-based ASICs, we present the first TPI technique that facilitates a hardware decompressor to drastically reduce the test volume and

the test generation time. Since *parallel/serial full scan* (PSFS) (see Chapter 2) is the most widely used scan architecture for test set compression, in this dissertation, we have presented test point insertion along with a scan chain re-ordering algorithm for designs with PSFS as well.

4. We also present algorithms to compute the proposed testability measures whose computational complexity is just $O(n)$, where n is the number of signal lines in the circuit. Hence, the proposed testability measures are scalable.

Experiments conducted with ISCAS '89, ITC '99, and a few industrial benchmarks clearly demonstrate the effectiveness and scalability of the proposed technique. By using very little extra hardware for implementing test points and very little extra run time for the TPI step, we show that the test volume and test application can be reduced by up to 64.5% and test generation time can be reduced by up to 63.1%. With broadcast scan compressors, experiments indicate that the proposed technique improves the compression by up to 46.6% and also reduces the overall ATPG CPU time by up to 49.3%.

1.5 Organization of the Dissertation

We survey the published literature on test volume and test application reduction in Chapter 2. Readers familiar with the prior work can directly read Chapter 3. Chapters 3, 4 and 5 describe the concepts of the proposed test point insertion scheme. In Chapter 3, we describe how to insert observe points for structured ASICs design without incurring any hardware overhead. We also describe, how we can insert test points directly into the physical design without affecting the design timing and the design placement and routing. Chapter 4 presents different types of test points that can improve the controllability and the observability of the structured ASIC design. Chapter 5 describes how test points can be inserted into the design employing compression technique. We conducted several experiments to identify the efficacy of all of the proposed algorithms. The results of these

experiments along with analyses are presented in Chapter 6. Finally, Chapter 7 concludes the dissertation and suggests possible future work.

Chapter 2
Prior Work

Due to ever increasing design sizes, the cost for testing large *application specific integrated circuit* (ASIC) designs has sky rocketed. The major factors governing this cost are: a) test volume, b) test application time, and c) test hardware cost. Test volume refers to the total number of test vectors that are required to be stored in the tester for testing a chip. Test application time is the total time required to test one chip in a tester. Test hardware cost refers to the total extra hardware that is inserted into the design for the purpose of testing. Researchers have presented several exciting *design-for-test* (DFT) techniques to reduce test cost that are widely used. They are: a) *built-in self test* (BIST), b) test data compression, c) new scan architectures, and d) test point insertion. However, all of these techniques comes with an extra hardware cost because one has to include special circuitry into the design for implementing the DFT hardware. In this chapter, we shall describe several test cost reduction techniques along with the hardware cost associated with them.

2.1 Test Data Compaction

Test data compaction [8, 18, 47, 48, 49, 56] is an algorithmic technique that modifies a given test cube, T, to reduce the test volume without reducing the fault coverage obtained by T. Today, all industrial *automatic test pattern generator* (ATPG) tools use test compaction to reduce test volume. Conventionally, compaction is classified into *dynamic* [47] and *static* [18] compaction according

to when compaction of test patterns is performed. Dynamic compaction is performed during test generation. The ATPG picks a fault f_P (the *primary fault*) and generates a test pattern T for f_P, typically, with several don't cares (defined as the unspecified bits in a test vector and denoted by X). The ATPG tool then tries to detect several *secondary faults* by specifying Boolean values for the Xs in T. After test pattern generation, static compaction is performed in which multiple compatible patterns are collapsed into one single pattern. Fault simulation-based [58] techniques are used for static compaction. Hamzaoglu and Patel [26] later presented more advanced techniques such as *redundant vector removal* and *essential fault pruning* to further compact a given test set. They also presented a new heuristic for estimating the minimum single stuck-at fault test set size. Recently, Kajihara [36] presented a technique to generate large numbers of Xs in a given test set without affecting the fault coverage achieved by that test set. They also describe how these Xs can be used to perform better compaction and compression of a given test set.

2.2 Built-in Self Testing

Built-in self-test (BIST) uses on-chip hardware for giving test inputs and for comparing the output response to test the device. Hence, it completely eliminates the need to use an *automatic test equipment* (ATE). Hardware test pattern generators can be classified into following major groups: exhaustive testing [71], pseudo-random testing [32], weighted random testing [10, 70], hardware pattern generators of deterministic tests [1, 20], and mixed-mode test pattern generation [28, 37]. These techniques offer different tradeoffs between the test data storage, test application time, hardware overhead, fault coverage, and programmability. Their properties have been studied extensively in the literature [5]. A *multiple input shift register* (MISR) is used to compact the response that is then compared against the actual compacted response stored in a ROM inside the chip. Hence, test cost is drastically reduced because the total time required to use ATE for

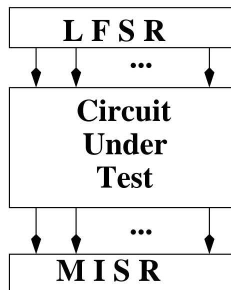


Figure 2.1: Block Diagram of a BIST System

testing is reduced. Since BIST cannot test the part of the circuit from the pins to the input multiplexor of the boundary scan register or from the outputs to the output pins, a low-cost ATE is still needed for parametric test of the pins, but this ATE is not needed for digital functional test. Figure 2.1 shows the block diagram of a BIST system in which a *linear feedback shift register* (LFSR) is used as a test generator and a MISR is used for response compaction.

However for pre-designed logic cores, this is only practical if the core was originally designed to achieve high fault coverage with BIST. Many legacy designs cannot be efficiently tested with BIST without significant re-design effort. Currently there are few commercially available cores that include BIST features. Usually only a set of test vectors for the core is available. The amount of BIST hardware required to apply a large set of deterministic test vectors is generally prohibitive [50, 64].

2.3 Input Test Data Compression

Today, test data compression is the most widely used technique to reduce the test data volume for *system-on-chips* (SoCs). In this approach, a precomputed test set T_D for an IP core is compressed (encoded) into a much smaller test set, T_E, which is stored in the ATE memory. An on-chip decoder is used for pattern decompression to obtain T_D from T_E during test application. A disadvantage of the compression technique is that additional hardware is required to decode the test pattern. Balakrishnan *et al.* [4] used the *entropy theory* to describe

the maximum achievable compression ratio for a given test vector set using any compression technique. They also classify the compression schemes into four main categories: a) *fixed-to-fixed code*, b) *fixed-to-variable*, c) *variable-to-fixed*, and d) *variable-to-variable*, which will be described in detail below.

2.3.1 Fixed-to-fixed Code

Techniques that use combinational expanders with more outputs than inputs to fill more scan chains with fewer tester channels at each clock cycle fall into this category. These techniques include using linear combinational expanders such as broadcast networks [25] and XOR networks [6], as well as non-linear combinational expanders [40, 54]. Section 2.4.1 describes the broadcast network of Hamzaoglu and Patel in detail. The LFSR reseeding technique proposed by Koenemann [37] also falls into this category because each fixed size test vector is encoded as a smaller fixed size LFSR seed and is stored in the tester memory. During the test application, the seeds are loaded into an on-chip LFSR that expands each seed into a test cube, which is then applied to the device under test. However, a major disadvantage of this scheme is that additional procedures are required to characterize and solve a system of linear equations to obtain the seeds from the test set. The total number of stages of the LFSR should also be carefully chosen as linear dependencies in the LFSR sequence may sometimes lead to an unsolvable system of linear equations.

If the size of the original blocks in the test vector to be encoded is n bits and the size of the encoded blocks is b bits, then there are 2^n possible symbols (original block combinations) and 2^b possible codewords (encoded block combinations). Since $b < n$, all possible symbols cannot be encoded using a fixed-to-fixed code. If $S_{dictionary}$ is the set of symbols that can be encoded (i.e., are in the "dictionary") and S_{data} is the set of symbols that occur in the original data, then if $S_{data} \subseteq S_{dictionary}$, it is a complete encoding, otherwise it is a partial encoding. For LFSR reseeding, it has been shown [37] that if b is chosen to be s+20 where

s is the maximum number of specified bits in any n-bit block of the original data, then the probability of not having a complete encoding is less than 10^{-6}. The *embedded deterministic test* (EDT) tool presented by Rajski [51, 52] uses a continuous flow decompressor that processes test data on the fly. Compared to the reseeding-based techniques [38], EDT reduces the area overhead and improves the encoding capabilities.

Reddy *et al.*'s [54] technique constructs the non-linear combinational expander so that it will implement a complete encoding. Several researchers have used partial encoding scheme to compress test vector. This can be done by adding additional constraints to the ATPG tool such that it only generates test data that is contained in the dictionary [6, 25]. Li *et al.* [40] use a bypassing technique for symbols that are not contained in the dictionary. Bypassing the dictionary requires adding an extra bit to each codeword to indicate whether it is coded data or not.

2.3.2 Fixed-to-variable Code

These codes encode fixed size blocks of data using a variable number of bits in the encoded data. Huffman codes [31] belong to this category. In Huffman coding, symbols that occur more frequently are encoded using shorter codewords and symbols that occur less frequently are encoded with longer codewords. This encoding scheme minimizes the average length of a codeword. To understand Huffman codes, consider the test data shown in Figure 2.2 in which each test pattern is organized as 12 blocks where each block contains four test bits. Table 2.1 shows the frequency of occurrence of each of the possible blocks (referred to as symbols). There are a total of 60 4-bit blocks in the example in Figure 2.2. Figure 2.3 shows the Huffman tree for this frequency distribution and the corresponding codewords are shown in Table 2.1. The column labeled **Huffman Code** represents the Huffman code for that particular symbol.

Jas *et al.* [33, 34] presented a selective Huffman code in which partial coding

0010 0100 0010 0110 0000 0010 1011 0100 0010 0100 0110 0010
0010 0100 0010 0110 0000 0110 0010 0100 0110 0010 0010 0000
0010 0110 0010 0010 0010 0100 0100 0110 0010 0010 1000 0101
0001 0100 0010 0111 0010 0010 0111 0111 0100 0100 1000 0101
1100 0100 0100 0111 0010 0010 0111 1101 0010 0100 1111 0011

Figure 2.2: Example of Test Data

Table 2.1: Statistical Coding Based on Symbol Frequencies for Test Set in Figure 2.2

Symbol	**Frequency**	**Block**	**Huffman Code**	**Selective Code**
S_0	22	0010	10	10
S_1	13	0100	00	110
S_2	7	0110	110	111
S_3	5	0111	010	00111
S_4	3	0000	0110	00000
S_5	2	1000	0111	01000
S_6	2	0101	11100	00101
S_7	1	1011	111010	01011
S_8	1	1100	111011	01100
S_9	1	0001	111100	00001
S_{10}	1	1101	111101	01101
S_{11}	1	1111	111110	01111
S_{12}	1	0011	111111	00011
S_{13}	0	1110	–	–
S_{14}	0	1010	–	–
S_{15}	0	1001	–	–

is used and the dictionary is selectively bypassed. The column labeled **Selective Code** in Table 2.1 represents the selective Huffman code. In this example, only the first three symbols (S_0, S_1, and S_2) with the highest frequency of occurrence are encoded. Figure 2.4 shows the Huffman tree for these three symbols. If the *most significant bit* (msb) of the selective Huffman code is 1, then it means the symbol is encoded; otherwise it is not encoded. Since the larger Huffman codes are not encoded, the size of the encoded test data is further reduced.

An important property of Huffman codes is that they are prefix-tree. No codeword is a prefix of another codeword. This greatly simplifies the coding process. However, the major disadvantage with Huffman codes is that the size of the decoder for decoding the codeword grows exponentially as the block size is

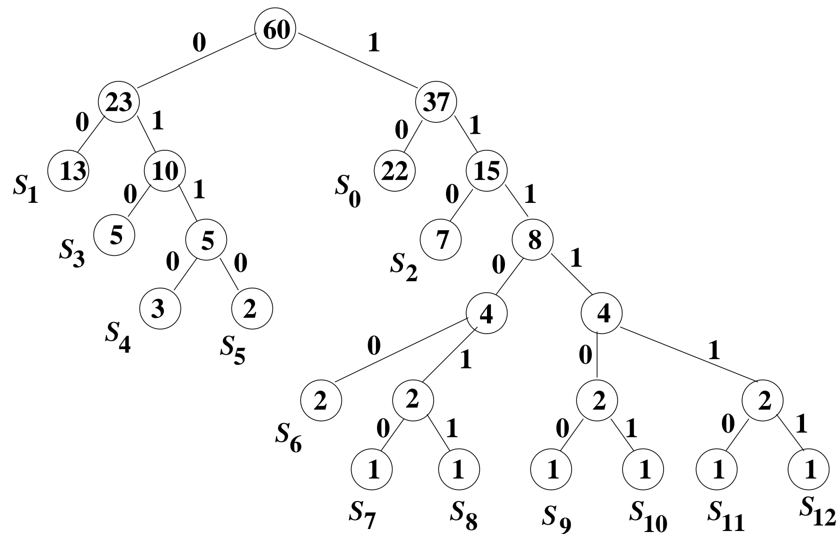


Figure 2.3: Huffman Tree for the Code Shown in Table 2.1

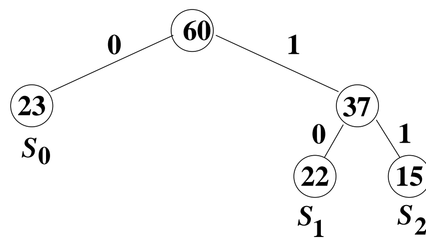


Figure 2.4: Huffman Tree for the Three Highest Frequency Symbols in Table 2.1

increased.

2.3.3 Variable-to-fixed Code

These codes encode a variable number of bits using fixed size blocks of encoded data. Conventional run-length codes belong to this category. An example of run-length code is given in Table 2.2. If the test vector is 000001 0000001 1 00001, then the encoded vector will be 101 110 000 100. Jas *et al.* [35] proposed a method for encoding variable length runs of 0s using fixed size blocks of encoded data. They use two scan chains as shown in Figure 2.5. One is called the *test scan chain* where the test vector is applied to the *circuit-under-test* (CUT). The other is called the *cyclical scan chain* that has the same number of scan elements as the test scan chain and compresses the test data. The serial output of the

Table 2.2: Example of Run length Code

Run Length Code	**Block**
000	1
001	01
010	001
011	0001
100	00001
101	000001
110	0000001
111	0000000

cyclical scan chain feeds the serial input of the test scan chain and also loops back and is XORed in with the serial input of the cyclical scan chain. Hence, the cyclical scan chain has the property that if it contains test vector t, then the next test vector that is generated in the cyclical scan chain will be the XOR of t and the difference vector that is shifted in. So, generating a test set consisting of n test vectors, t_1, t_2, ..., t_n, in a cyclical scan chain would involve first initializing the scan chain to all 0's, and then shifting t_1 into the scan chain followed by the difference vector $t_1 \oplus t_2$, followed by $t_2 \oplus t_3$, and up to $t_{n-1} \oplus t_n$. They use a heuristic approach to carefully order the test set such that the resulting difference vectors have large numbers of 0's. This is based on their observation that test vectors are highly correlated. Faults in the CUT that are structurally related require similar input value assignments in order to be excited and sensitized to an output. Thus, many pairs in the test set will have similar input combinations such that the difference vectors have a lot of 0's. Ordering the test vectors in the test set such that correlated test vectors follow each other results in difference vectors with many 0's than 1's. The LZ77 based coding [68] presented by Wolff and Papachristou also falls in this category. They use a partial encoding scheme with a bypass mode for compressing the test data.

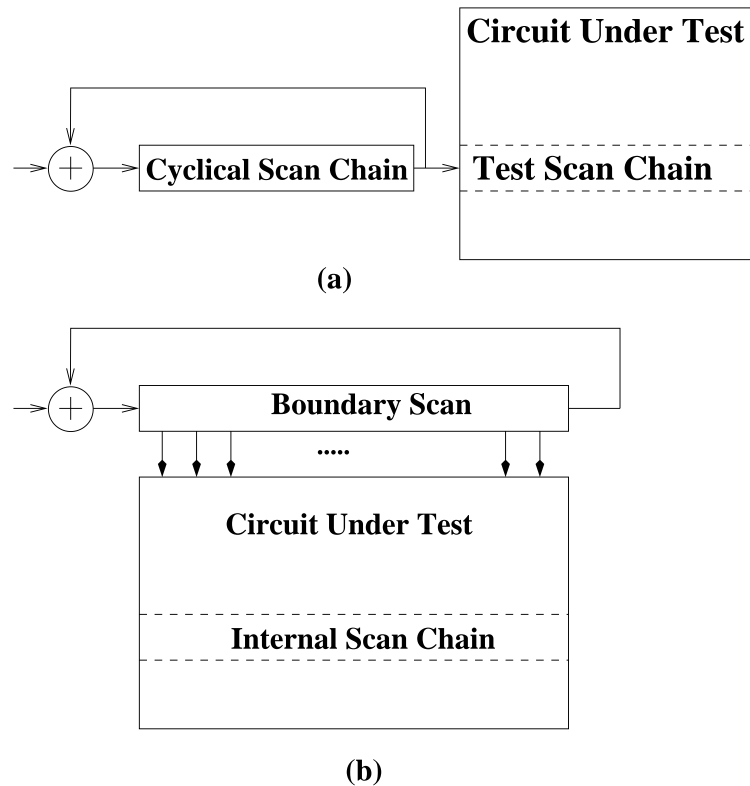


Figure 2.5: Cyclical Scan Chain: a) Decompression Architecture and b) Using Boundary Scan

2.3.4 Variable-to-variable Code

These codes encode a variable number of bits in the original data using a variable number of bits in the encoded data. Several techniques that use run-length codes with a variable number of bits per code word have been proposed. They are: a) Golomb codes [17], b) *frequency directed codes* (FDR) [16], and c) *variable input Huffman codes* (VIHC) [22].

The first step of a Golomb coding scheme is to carefully order the given vector set T_D and generate the difference vector set T_{diff}. As explained in Section 2.3.3, the difference vector is given by $T_{diff} = \{t_1, t_1 \oplus t_2, \ldots, t_{n-1} \oplus t_n\}$. The next step in the encoding procedure is to select the Golomb code parameter m, referred to as the group size. Once the group size m is selected, the runs of zeros in T_{diff} are mapped to groups of size m (each group corresponding to a run length). The number of such groups is determined by the length of the longest run of zeros

in the T_{diff}. The set of run lengths {0, 1, 2, ..., m-1} forms group A_1; the set {m, m+1, ..., 2m-1}, forms group A_2 and so on. In general, the set of run lengths {(k-1)m, (k-1)m+1, ..., km-1} comprises group A_k. To each group A_k, they assign a group prefix of (k-1) ones followed by a zero, which is denoted by $1^{k-1}0$. If m is chosen to be a power of two, i.e., m=2^N, each group contains 2^N members and a $\log_2 m$-bit sequence (tail) uniquely identifies each member within the group. Thus, the final code word for a run length L that belongs to group A_k is composed of two parts – a group prefix and a tail. The prefix is $1^{(k-1)}0$ and the tail is a sequence of $\log_2 m$ bits. They showed that for a run of length l, $k = \lfloor l/m \rfloor$. The Golomb codes for the first three groups, A_1, A_2, and A_3, are illustrated in Table 2.3 for m=4.

Table 2.3: Example of Golomb Code

Group	**Run Length**	**Group Prefix**	**Tail**	**Code Word**
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011

Later, Chandra and Chakrabarty [16] observed that the difference test set T_{diff} contains a large number of short runs of 0s. If a Golomb coding with a code parameter m is used, a run of l 0s is mapped to a codeword with $\lfloor l/m \rfloor + \log_2 m$ bits. Hence, test data compression is more efficient if the runs of 0s that occur more frequently are mapped to shorter codewords. Hence, they proposed the *frequency directed run length* (FDR) codes. In FDR codes, the runs of 0s are divided into groups A_1, A_2, ..., A_k, where k is determined by the length l_{max} of the longest run ($2^k - 3 < l_{max} \leq 2^{k+1} - 3$). Note also that a run of length l

is mapped to group A_j where j=$\lceil log_2(l+1)-1\rceil$. The size of the ith group is equal to 2^i, i.e., A_i contains 2^i members. Each codeword consists of two parts − a group prefix and a tail. The group prefix is used to identify the group to which the run belongs and the tail is used to identify the members within the group. An example of FDR codes for the first three groups, A_1, A_2, and A_3, is shown in Table 2.4.

Table 2.4: Example of *Frequency Directed Run Length* (FDR) Code

Group	**Run Length**	**Group Prefix**	**Tail**	**Code Word**
	0		0	00
A_1	1	0	1	01
	2		00	1000
A_2	3	10	01	1001
	4		10	1010
	5		11	1011
	6		000	110000
A_3	7	110	001	110001
	8		010	110010
	9		011	110111
	10		110	110100
	11		111	110101
	12		110	110110
	13		111	110111

One of the difficulties with variable-to-variable codes is synchronizing the transfer of data from the tester. All of the techniques that have been proposed in this category require the use of a synchronizing signal going from the on-chip decoder back to the tester to tell the tester to stop sending data at certain times while the decoder is busy. Fixed-to-fixed codes do not have this issue because the data transfer rate from the tester to the decoder is constant.

2.4 Scan Architectures for Test Volume and Test Application Time Reduction

Several novel scan architectures are also available in the literature and can significantly reduce test volume and the test application time. Here scan architecture

refers to the way in which various *scan flip-flops* (SFFs) are organized and connected with other SFFs in the design. Baik *et al.* [2] presented a *random access scan* (RAS) architecture to reduce test application and test volume. In this technique, the scan function is implemented like a *random-access memory* (RAM) [13]. Here all SFFs form a RAM in the scan mode. In general a subset of SFFs can be included in the RAM if partial-scan is desired. However, due to the high hardware overhead, RAS is not very widely used.

Broadcast scan [39] is a popular scan architecture that is widely used for reducing the test data volume. In the broadcast scan scheme, a single scan chain is divided into multiple small scan chains such that each of these small scan chains are driven by a single scan input. Additional constraints are added to the ATPG tool to generate the broadcast scan test data. A disadvantage of this scheme is that since the same test data is "broadcasted" to multiple scan chains, several useful test vectors cannot be applied to the *device under test* (DUT). Due to this artificial constraint, the compression ratio achieved by the broadcast scan scheme is generally inferior compared to the LFSR reseeding technique. However, the major advantage of the broadcast scan scheme is its simplicity and the ease of implementation.

2.4.1 Illinois Scan Architecture

Recently, several variants of the broadcast scan architecture implementations have been proposed. The *Illinois scan architecture* (ILS) proposed by Hamzaoglu and Patel [25] is an implementation of the broadcast scan concept and is also referred as *parallel/serial full scan* (PSFS). This is achieved by dividing the scan chain into multiple partitions and shifting the same vector to each scan chain through a single scan input.

Figure 2.6 shows the organization of SFFs in the parallel serial full scan architecture. SFFs are partitioned into n scan chains and are labeled *Scan Chain 1*, 2, ... n, respectively. The block labeled $MISR$ represents an n-input *multiple*

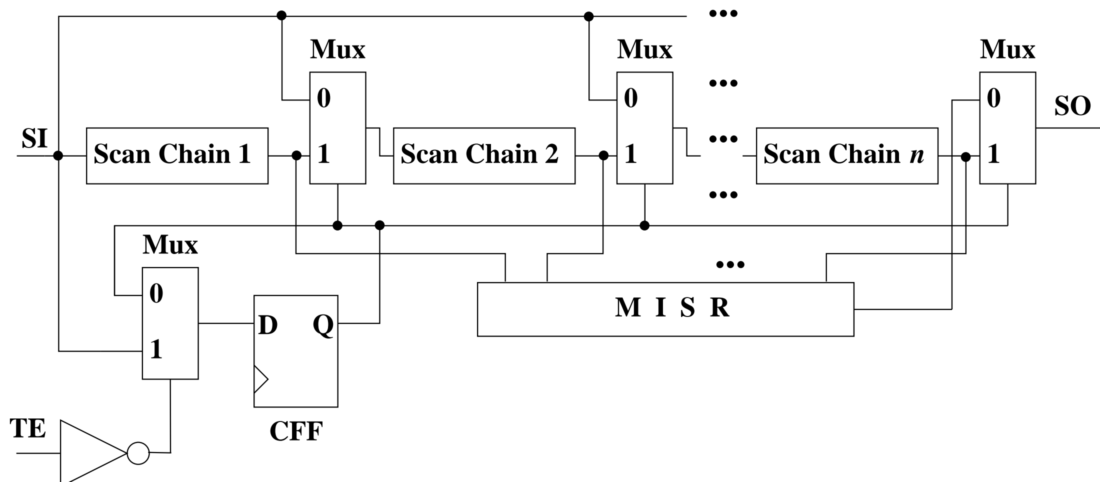


Figure 2.6: Parallel Serial Full Scan Architecture

input shift register and is used for response compaction. The TE input controls the operation of the PSFS cores. When TE is 0, the core operates in the normal mode. When TE is 1, the core operates in the test mode. The operation of the PSFS in the test mode is controlled by the control flip-flop, CFF. When CFF=1, the PSFS core operates in the *serial test mode*. The operation of the PSFS core in the serial test mode is same as the operation of the full scan cores in the test mode, i.e., the new values are shifted into each flip-flop serially through the SI input and the previous values of all SFFs are shifted out serially through the SO output. When CFF=0, the PSFS core operates in *parallel test mode*. In the parallel test mode, shifting out test responses from all scan chains is done in parallel using the MISR. The MISR collects the bit streams coming out from n scan chains, compacts them and serially outputs the compacted response through SO output. This is illustrated in Figure 2.7. Here, the block labeled $CMPT$ is a response compactor. This design has two scan inputs (SI_1 and SI_2) and two scan outputs (SO_1 and SO_2).

The advantage of the parallel mode is that a much shorter test sequence will now be shifted into the scan chain. This reduces both test volume and test application time. But note that the same bit is shifted into different scan chains. Due to this artificial constraint, several faults remain untestable in parallel mode.

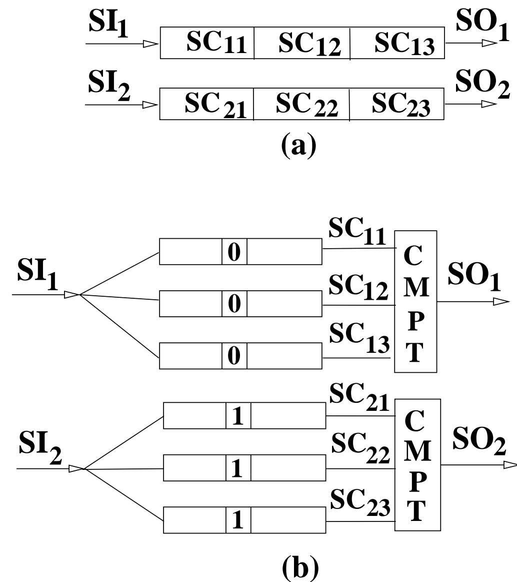


Figure 2.7: Broadcast Scan: a) Serial Mode and b) Parallel Mode

Thus, the effectiveness of the PSFS technique also relies on the scan chain partitioning and ordering algorithm. Hence, they also proposed a near optimal scan chain partitioning and ordering scheme to improve the performance of the Illinois scan architecture. But, a disadvantage of their approach is that implementing the optimal scan ordering may require changing the existing placement and routing of the design, thereby adding extra turn-around time into the design flow.

Hsu *et al.* [29] presented a case study on the effectiveness of the Illinois scan architecture on large industrial benchmark designs. Pandey and Patel [45] presented a reconfiguration technique that supports two parallel modes and a serial mode of operation for applying the test vector so that a large number of faults are detected in the parallel mode itself. This idea has been further extended in the *adaptive acan* [55] architecture of Synopsys and the *virtual scan* [67] architecture of Syntest. They used a more elaborate "broadcast" logic that supports multiple parallel modes of operation so that all of the faults in the circuit are detected in the parallel modes. A disadvantage of this scheme is the large hardware overhead because additional control logic is required to support multiple parallel modes of operation. Also, one has to modify the ATPG tool so that it can effectively utilize the multiple parallel modes.

2.5 Test Point Insertion

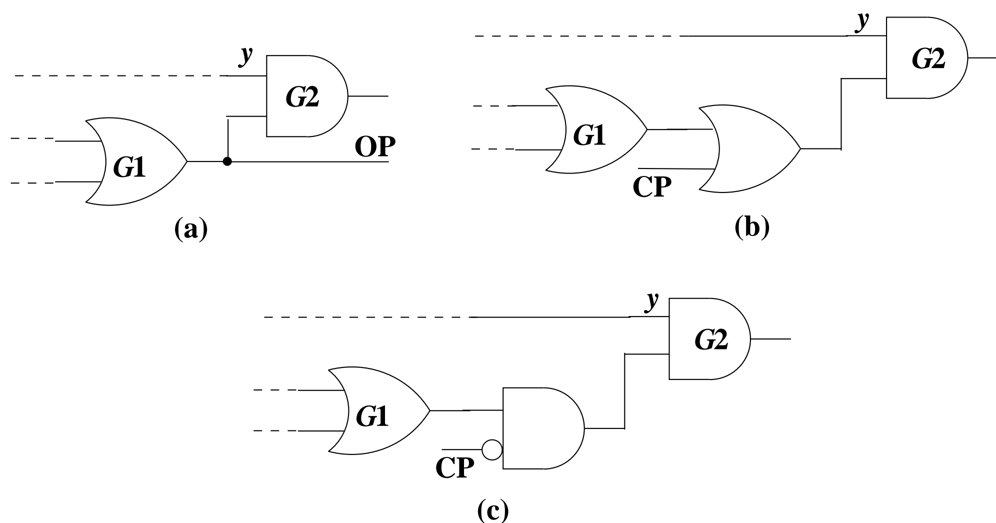


Figure 2.8: Example Illustrating Test Points: a) Observation Point, b) 1-Control Point, and c) 0-Control Point

The *test point insertion* (TPI) problem was originally proposed by Hayes and Friedman [27]. The goal of their TPI scheme [3, 27] is to select t signal lines in a combinational circuit as candidates for inserting *observation points* (OPs), so as to minimize the number of test vectors needed to detect all single stuck-at faults in the circuit. An observation point is an additional primary output that is inserted in the circuit to increase the observability of faults in the circuit. In the example in Figure 2.8(a), an observation point is inserted at the output of gate $G1$ such that faults are observable regardless of the logic value at node y. A control point is inserted in the circuit such that when it is activated, it fixes the logic value at a particular node to increase the controllability of some faults in the circuit. A control point can also affect the observability of some faults in the circuit because it can change the propagation paths in the circuit. In the example in Figure 2.8(b), a control point is inserted to fix the logic value at the output of gate $G1$ to a 1 when the control point is activated (this is called a control-1 point). This is accomplished by placing an OR gate at the output of gate $G1$. In the example in Figure 2.8(c), a control point is inserted to fix the logic value at

the output of gate G1 to a 0 when the control point is activated (this is called a control-0 point). This is accomplished by placing an AND gate at the output of gate $G1$. During system operation, the control points are not activated and hence, they do not affect the system function. However, control points do add an extra level of logic to some paths in the circuit. If a control point is placed on a critical timing path, it can increase the cycle time of the circuit. Since test points add both area and performance overhead, it is important to try to minimize the number of test points that are inserted to achieve the desired fault coverage. Optimal test point placement for circuits with reconvergent fan-out has been shown to be NP-complete [3].

TPI has become a popular *design-for-test* (DFT) technique to improve fault coverage for designs using random pattern *built-in self-test* (BIST) (see Section 2.2). Control points and observation points are inserted to improve random pattern testability of the circuit. Simulation-based [12, 32] and testability measure-based [43] approaches are often used to choose signal lines to insert test points. Another interesting advantage of test points is that the SFFs used for test points can also be used for silicon debug as described by Carbine and Feltham [14].

2.5.1 Test Point Insertion for BIST

As mentioned earlier in Section 2.2, BIST uses pseudo-random hardware test generators to generate patterns to detect all of the faults in the circuit. However, BIST fails to achieve the desired fault coverage with a reasonable test sequence length because of the presence of large numbers of random-pattern resistant faults. Hence, control points and observation points are inserted to improve the fault coverage [43, 59, 62, 65]. Briers and Totton [12] were the first to present a systematic algorithm to insert test points to improve the random pattern testability of BIST. They use simulation statistics to identify correlation between different signal lines and inserted test points to break these correlation. Iyengar and Brand [32] proposed a fault simulation technique to identify gates that block fault propagation.

Many TPI methods [9, 57, 59, 66, 73] use the COP [11] testability analysis to extract the testability information. For each line n, the COP measurements gives the controllability estimates $CZ(n)$, and $CO(n)$, expressing the probability that n will be 0 or 1, and an observability estimate $W(n)$, expressing the probability that a fault effect at n will be observable at a primary output. Combining these controllability and observability estimates results in the testability (or detection probability) estimates for a fault.

Savaria *et al.* [57] and Youssef *et al.* [73] used the COP measure to guide the test point selection process. They identify regions of hard-to-detect faults and insert test points at the origins of these sectors. Seiss *et al.* [59] presented a cost function based on the COP testability measure and then computed, in linear time, the gradient of the function with respect to each possible test point. The gradients were used to approximate the global testability impact for inserting a particular test point. Based on these approximations, a test point was inserted and the COP testability measure was recomputed. This process iterated until the testability of the design became satisfactory.

Touba and McCluskey [65] presented a path tracing-based algorithm to insert test points to improve the fault coverage using BIST. They used fault simulation to identify the set of faults that were not detected by the given test set. For each undetected fault, a path tracing procedure was used to identify the set of test points that would enable that fault to be detected. Given a set of test points for all of the undetected faults in the circuit, they then used a set covering procedure to reduce the number of test points to be inserted. They also presented a new technique for driving the control points. Rather than adding extra scan elements to drive the control points, a few of the existing primary inputs of the circuit were ANDed together to form signals to drive the control points. This is shown in Figure 2.9. Hence, their approach comprised four steps and is summarized below:

1. Perform Fault Simulation.

   In this step, fault simulation is performed for the set of test patterns to be

applied to the CUT to identify which faults are already detected and which faults need test points to be detected.

2. Compute Test Point Set.

   In this step, they compute the set of test points for each undetected fault such that if one test point from this set is inserted then the fault is detected.

3. Minimizing the Number of Test Points.

   Given a set of test point solutions for each undetected fault, a set covering procedure is used to identify the minimal test point set that enables all of the faults to be detected.

4. Synthesize Logic to Activate the Control Points.

   Pattern decoding logic is synthesized to activate control points for certain patterns.

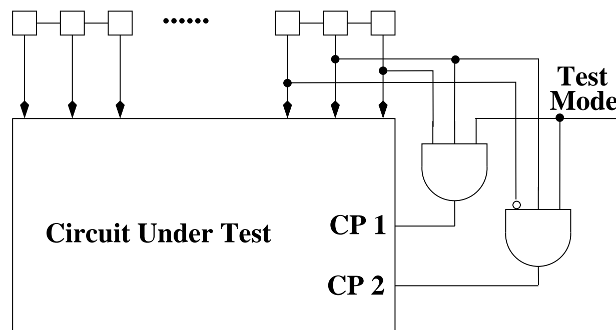


Figure 2.9: Control Points Driven by Pattern Decoding Logic

Tamarapalli and Rajski [62] presented another approach for inserting test points for BIST. They observed that when a large number of *control points* (CPs) are inserted into a design, sometimes the inserted CPs begin to work in a destructive fashion, resulting in a loss of fault coverage. This happens in very large and complex designs. To address this issue, they partitioned the test application into several constructive phases. Each phase contributes to the results achieved so far, and hence, improving the fault coverage. The selection of test points for each phase is guided by progressively reducing the set of undetected faults. Within

each phase, a set of control points maximally contributing to the fault coverage achieved so far is identified using a probabilistic fault simulation technique, which accurately computes the impact of a new control point in the presence of the control points selected so far. In this manner, in each phase a group of control points, driven by fixed values and operating synergistically, is enabled. In addition, observation points maximally enhancing the fault coverage are selected by a covering technique that utilizes the probabilistic fault simulation information. Figure 2.10 shows the block diagram of a BIST system that incorporates their proposed test point scheme. The phase decoder block generates three signals $P1$, $P2$, and $P3$ that supports three different phases. In the first phase, the signal $P1$ is 1 due to which only the CPs corresponding to phase 1 are enabled. Similarly, the phases corresponding to signals $P2$ and $P3$ work constructively to detect all of the faults in the circuit. The advantages of their method are its ability to converge, due to

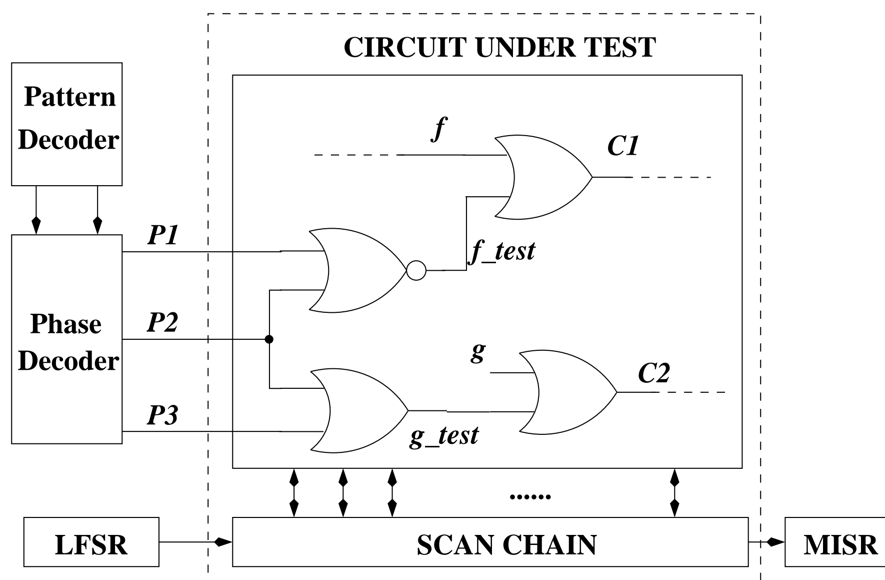


Figure 2.10: BIST with the Multi-phase Test Point Scheme

partitioning, and increased accuracy in predicting the impact of multiple control points, due to the usage of fixed values. In addition, the number of ineffective control points is reduced, since conflicting control points are enabled in different phases. These features, combined with inherent sharing of logic driving the control points, lead to a technique with low area overhead. They also showed that

the power dissipation during test mode is potentially reduced due to the usage of fixed values to drive control points and due to the distribution of control points across multiple phases.

2.5.2 Test Point Insertion for Non-BIST Designs

A *non-BIST* design means a design that uses externally stored test vectors in the *automatic test equipment* (ATE) for testing. As mentioned earlier, Hayes and Friedman [27] originally proposed the TPI problem for non-BIST designs. They also proposed a labeling scheme and modeled the test point insertion problem as an integer programming problem. They only considered combinational circuits without any fanout signals in them. Berger and Kohavi [7] showed how to use their labeling scheme to generate minimal test sets. Later, Balakrishnan [3] presented a dynamic programming approach for the TPI problem. Balakrishnan's work [3] considered circuits with fanout signals but does not guarantee optimal results for circuits with fanouts. Both, Balakrishnan's work and Hayes and Friedman's work used a testability measure called *test count* (TC), which will be shortly described. The TC measure proposed by them, later, became the basis of Geuzebroek *et al.*'s [23, 24] work for reducing test volume using TPI. Test counts for each signal line, l, in the circuit comprise the following four values:

1. The essential zeros (ones), $E0$ ($E1$). This is the minimum number of times a line must become 0 (1) (during the application of a test set) and be observable on an output, such that all corresponding faults in its fanin-cone can be covered.

2. The total zeros (ones), $T0$ ($T1$). This is the minimum total number of times a line must become 0 (l) (during the application of a test set) (either observable or making other lines observable).

Figure 2.11 shows a circuit to illustrate the computation of TC. In order to cover the stuck-at 0 (1) faults on a, b, and c, these lines must be l (0) and

observable at least once. The stuck-at 1 faults at a and b have to be tested independently, therefore d must be 0 at least twice ($E0_d$=2). The stuck-at 0 faults at a and b can be tested at the same time such that d only has to be 1 once ($E1_d$=l). Likewise, the stuck-at 1 faults at c and d have to be tested independently while the stuck-at 0 faults can be tested at the same time. In this example, e is the circuit output and hence it is always observable, ($T0_e$=$E0_e$ and $T1_e$=$E1_e$). Line d must be 0 three times: a) when e is 0 twice and b) when c is observable (c is only observable when d has the non-controlling value of 0), which means $T0_d$=3. The same applies for $T0_c$. The lower bound on the test set size found is 4, which happens to be the actual minimal test set size in this simple example circuit, because it contains no reconvergent fanout. The TC method is exact only for fanout-free circuits, because it cannot be known in advance how the essential zeros/ones will divide over the different branches of the fanout stem during the actual ATPG.

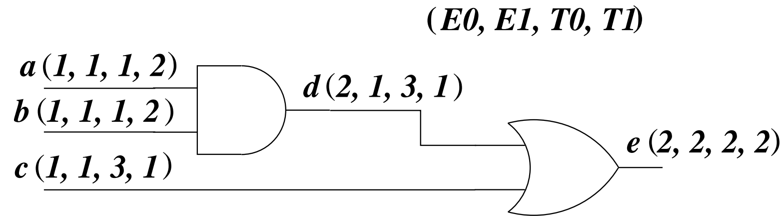


Figure 2.11: Example to Illustrate Test Counts

2.5.2.1 Test Point Insertion to Reduce Test Volume

Geuzebroek *et al.* [23, 24] were the first to show that a TPI scheme can also be used for non-BIST designs for reducing the test volume. In their work, they used the COP measure and the test counting measure to construct a cost function as shown in Equation 2.1. They inserted test points to minimize this cost function.

$$K_l = \frac{E0_l}{C0_l \cdot W_l} + \frac{T0_l - E0_l}{C0_l} + \frac{E1_l}{C1_l \cdot W_l} + \frac{T1_l - E1_l}{C1_l} \tag{2.1}$$

This equation consists of four parts:

1. The minimum number of times the line should be driven to 0 and be observable at the same time, divided by the probability that the line is 0 and observable at the same time.

2. The remaining minimum number of times the line should be driven to 0, divided by the probability that the line is 0.

3. The minimum number of times the line should be driven to 1 and be observable at the same time, divided by the probability that the line is 1 and observable.

4. The remaining minimum number of times the line should be driven to 1, divided by the probability that the line is 1 at the same time.

Later [24], they analyzed the testability of the design using the SCOAP [21], the COP [11], and the TC measure by assigning priorities to them. A high priority for a particular testability measure indicates that the test problem corresponding to that testability measure is high. For the different testability measures, the following priority schemes are used:

- COP priority. The COP priority is determined by the number of faults with a COP detectability (denoted by P_d) below a given threshold value. The more faults with a P_d below this threshold, the higher the priority. The priority is also higher, when there are faults with extremely low P_d values ($< 1^{-10}$) to make sure that the TPI algorithm will target these very hard-to-test faults, even when there are only a few of them.

- SCOAP priority. The number of faults with SCOAP values higher than a given threshold determine the SCOAP priority. They showed that relative SCOAP values (SCOAP values of a given signal line compared to SCOAP values of other lines) are more useful than absolute SCOAP values. Therefore, they determined the threshold by the distribution of the SCOAP values and it is set at $SCOAP + 3 \cdot \sigma_{SCOAP}$ (standard deviation).

- Test Counting priority. For the test counting, only two priority values are used: 0 and 1. Again they used a threshold value to differentiate between these priorities. When there are TC values higher than this threshold, the priority is 1, otherwise it is 0.

When the COP priorities are higher than the SCOAP priority, a COP-based cost function is selected. If the SCOAP priority is higher, a SCOAP based cost function is selected. If COP and SCOAP priorities are equally high, a COP and SCOAP based cost function is selected. The cost function was always TC based when the TC-priority is 1. They also showed that the presence of large *fanout-free regions* (FFRs) in a design increases the test volume. Hence, their scheme inserted test points into the large FFRs in the design to break them into smaller FFRs. This is illustrated in Figure 2.12 where inserting a transparent flip-flop (a test point) at signal line a breaks the large FFR into two small FFRs. Yoshimura

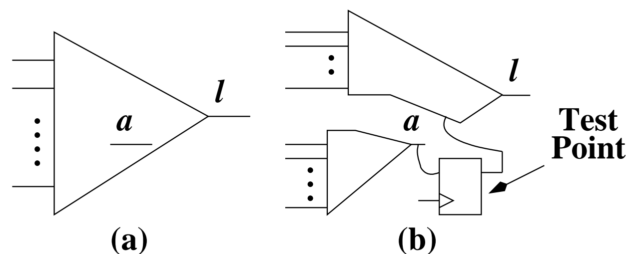


Figure 2.12: Breaking Fanout Free Regions (FFRs): a) A Large FFR and b) Smaller FFRs

et al. [72] described a test point insertion technique that reduces the test volume by improving the fault detection probability.

Chapter 3
Observation Points

This chapter describes a zero cost technique to insert observation points into the SA designs to reduce test set sizes and ATPG run time. Since structured ASIC products require very short turn around time, long ATPG run time is undesirable. Large structured ASICs often require a large number of test patterns to achieve the desired fault coverage. We use the unused flip-flops in the structured ASIC design to implement observation points. Hence, the proposed technique does not incur any hardware overhead. Since test points are inserted during a post-layout step, considering both timing and layout information, hence test points can be inserted without changing the existing layout or routing. In Section 3.2.1, we describe how we identify the unused hardware directly from the physical design and take care of both the timing constraints and the routing constraints for inserting the observation points. In Sections 3.2.2 and 3.2.3, we introduce the novel gain functions that specifically quantify the reduction in test volume and test time to select the best signal lines for inserting observation points.

3.1 Introduction

As mentioned in Chapter 1, there is a big gap in performance, area, and power consumption between standard cell-based ASICs and *field programmable gate arrays* (FPGAs). *Structured ASICs* (SAs) have emerged as a new technology that can fill this gap. SAs consume less power than FPGAs and incur less *non-recurring engineering* (NRE) cost with shorter *turn-around-time* (TAT) compared to cell-based ASICs [74]. In this chapter, we will describe how to insert the observation points

using NEC's *Instant Silicon Solution Platform* (ISSP) products (see Chapter 1 for details on ISSP). However, the technique can be extended to other SA products without loss of generality.

Recent structured ASIC chips accommodate designs comprising several million gates. Achieving near 100% fault coverage for such large designs requires enormous CPU time. Since TAT is a critical factor for SAs, long test generation time is undesirable. The total test set size that needs to be stored in the tester is another important criterion that determines test cost. Test data compression [34] and test point insertion [27] are two different DFT techniques that can reduce the test data volume. To achieve the highest reduction in test data volume, both an on-chip decompressor and test points are used.

We propose a zero cost test point insertion technique that can reduce both test volume and ATPG CPU time significantly. Later, in Chapter 5, we will extend the proposed TPI technique and show how it can be used in conjunction with any compression technique to further reduce the test size. The proposed test point insertion technique identifies useful unused flip-flops (see Section 3.2.1) that can be used for observation points without affecting existing placement and routing. To eliminate any increase in routing area due to inserting test points and minimize impact on the overall TAT, test points are inserted after placement and routing is completed. To keep the run time for large industrial designs as low as possible, efficient algorithms using *cone based partitioning* and *divide-and-conquer based* approaches (see Section 3.2) are used.

3.2 The Algorithm to Insert Observation Points

Our TPI technique exploits the fact that test patterns that have many don't cares (defined as inputs that are not assigned Boolean values by ATPG and denoted by X's) are more amenable to test compaction. Test compaction techniques can be classified into dynamic [47] and static compaction [18] according to when compaction of test patterns is performed. Dynamic compaction is performed

during test generation. The ATPG picks a fault f_P (called the *primary fault*) and generates a test pattern T for f_P, typically, with several don't cares. The ATPG tool then tries to detect several *secondary faults* by specifying Boolean values for the don't cares in T. Hence, if the ATPG generates test patterns that have more don't cares, then more secondary faults can be detected by each test pattern resulting in a smaller test set. Static compaction merges multiple compatible test patterns into a single test pattern to reduce the number of patterns in the final test pattern set. Test patterns that have many don't cares can be easily merged with other test patterns.

The proposed TPI technique inserts test points directly into the physical design. Hence, we make sure that inserting any test points does not require changing the existing placement and routing of the user design. By using layout and timing information during the test point insertion process, we minimize any possible increase in turn-around time due to the test point insertion step. Inserting an observation point into a signal line l can increase the load capacitance of the driver that drives l. In order to prevent any possible performance impact due to this, we do not insert any test points into signal lines that are on a timing critical path. Since inserting a test point requires adding extra wires, it may not be possible to insert test points in an area that is highly congested without changing the existing placement and routing. We cannot add any additional hardware into a SA design. These constraints are described in detail in the following section.

3.2.1 The Constraints

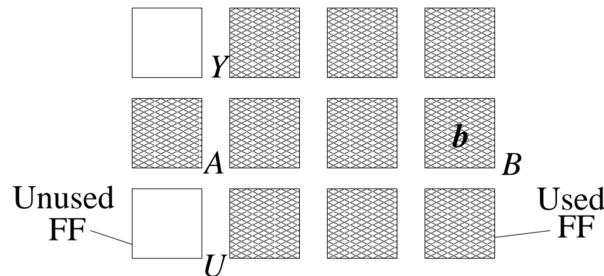


Figure 3.1: Used and Unused CMG Cells

Our TPI technique uses the unused scan *flip-flops* (FFs) in the CMG cell to implement observation points. Thus, no extra chip area is used to implement observation points. Although all unused FFs can be used for test points without incurring any additional hardware, we use only a few selected FFs for the following reasons: 1) Inserting a test point requires additional wires connecting CMG cells. This increases the chances of having physical defects on these wires and CMG cells, thereby reducing the yield, and 2) In ISSP, the clock signals for all unused flip-flops are clamped to VDD/GND. However, using an unused FF to implement an observation point requires changing the clock signal of the unused FF to be driven by the system clock. This increases power consumption in functional mode. Hence, in our experiments, we limit the number of inserted observation points to less than 1% of the total number of flip-flops in the entire design, even if the design has more unused FFs available.

Figure 3.1 shows a part of CMG array. Highlighted squares denote CMG cells whose FFs are used and blank squares represent CMG cells whose FFs are not used. Suppose that a signal line b in the cell B is the best place to insert a test point to reduce test set size and test generation time. Signal b can be observed using the unused FF in the cell Y or U. However, since the cells Y and U are located far away from b, connecting b to the unused FF in either of the CMG cells will require a long wire. Hence, observing b by the FF in U or Y may not be possible without changing the routing in that area, unless that part of the CMG array is very sparsely routed. A long wire also increases the signal propagation delay of the paths with signal b. Hence, we do not insert a test point into signal line b. Signal lines in CMG cell A can be connected to a FF either in U or Y, which is adjacent to A, with a very short wire. Since routing is local, major changes in placement and routing are not necessary. In our experiments, a signal line l is considered as a potential test point candidate signal line only if there is at least one unused FF in the neighborhood of l.

An observation point at line l is inserted by adding a wire that connects the signal line l to an unused FF adjacent to l in the layout. However, if line l lies on

a timing critical path, then inserting a test point into l may degrade performance. If l is in a highly congested area, it may not be possible to add even a short wire to connect l to an adjacent unused FF. In the experiments (see Chapter 6), we used a commercial physical design tool and a static timing analysis tool to identify all signal lines that belong to a routing congested area or lie on a critical path. We exclude these signal lines from the potential test point candidates.

3.2.2 Gain Function

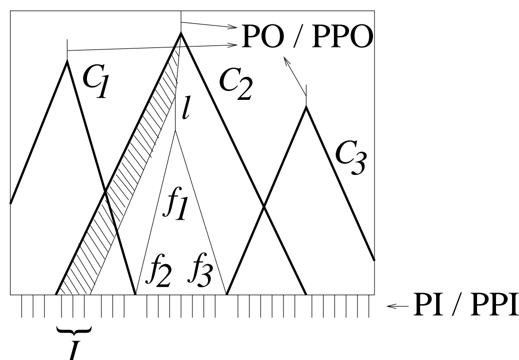


Figure 3.2: Example of Circuit with Three Logic Cones C_1, C_2, and C_3

Figure 3.2 shows a circuit that comprises three logic cones, C_1, C_2, and C_3, where each of the logic cones drives a primary output or a pseudo-primary output. Assume that a set of inputs I has to be specified to propagate any fault effect on line l to the nearest observation point. If we insert an observation point at l, then none of the inputs in I need to be specified to detect the fault effect at signal line l. Now, consider the faults f_1, f_2, and f_3 in the fanin cone of l. If they are all independent faults, then inserting an observation point into l will increase don't cares in the test patterns for f_1, f_2, and f_3 (two faults f_a and f_b are said to be *independent* if they cannot be simultaneously detected by any single pattern T). These don't cares created by inserting the observation point can then be specified to detect more faults and reduce test set sizes. Note that since fault effects for f_1, f_2, and f_3 are observed at l without propagating them further, this can reduce ATPG run time.

Since we limit the number of observation points inserted, we select only the best signal lines to insert test points that can reduce the test generation time and test data volume as much as possible. Using a gain function, we quantify how desirable it is to insert a test point at line l. Our gain function $G(l)$ for a signal line l represents the total number of X's that will be additionally generated when an observation point is inserted at l and the savings in test generation time required to propagate fault effects at signal line l to *primary outputs* (POs) / *pseudo-primary outputs* (PPOs). The gain function for signal line l is mathematically defined as:

$$G(l) = N_f(l) \times N_i(l), \tag{3.1}$$

where $N_f(l)$ is the total number of independent faults in the fanin cone of the signal line l and $N_i(l)$ is the minimum number of primary or pseudo primary inputs that need to be specified to observe the fault effect at the signal line l.

Theorem 3.2.1 *$G(l)$ is the lower bound of the total number of X's additionally generated in the resulting test set by inserting an observation point at signal line l.*

Proof. To detect faults in the fanin cone of l, ATPG has to generate at least $N_f(l)$ vectors (because there are $N_f(l)$ independent faults). Now, if we insert an observation point at l, then ATPG need not specify $N_i(l)$ inputs that are necessary to propagate the fault effect at l to a PO/PPO in $N_f(l)$ or generate more test patterns. Hence, the test patterns for $N_f(l)$ faults will have at least $N_i(l)$ additional X's. Therefore, the lower bound of the number of additional X's generated by inserting an observation point at l in the resulting test set is $N_f(l) \times N_i(l)$. □

According to Theorem 3.2.1, if the gain function of a signal line l is large, then inserting an observation point into l will greatly increase don't cares in the resulting test set. This explains why we use $G(l)$ as the gain function for selecting signal lines for test point insertion. We will now describe efficient and accurate algorithms to compute the gain function $G(l)$.

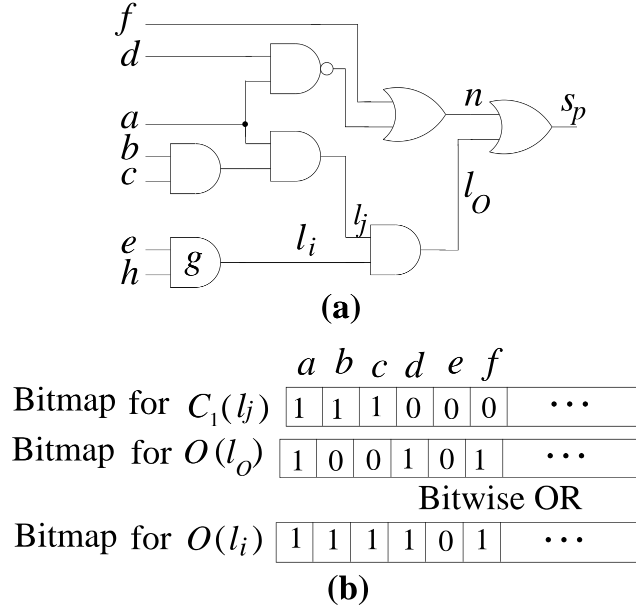


Figure 3.3: Accurate Computation for Reconverging Signals

3.2.2.1 Computing $N_i(l)$

If the design has reconvergent fanouts, then calculating accurate $N_i(l)$ for signal l is difficult. As a simple estimation of $N_i(l)$, we can use a SCOAP-like [21] measure where the observability cost of the line l_i is obtained by simple additions of controllability costs of all its other inputs and the observability cost of l_O where signal line l_i is the ith input of gate g and l_O is g's output. This is defined as:

$$O(l_i) = \begin{cases} 0, & \text{if } l_i \text{ is PO/PPO} \\ \sum_{j \neq i} C_{\overline{c}}(l_j) + O(l_O), & \text{otherwise} \end{cases} \tag{3.2}$$

where $C_{\overline{c}}(l_j)$ reflects the number of inputs required to set the line l_j to its non-controlling Boolean value $\overline{c}$.

The controllability cost $C_v(l_i)$ of signal line l_i that is driven by gate g reflects the number of inputs to be specified to set line l_i to Boolean value v and is defined as:

$$C_v(l_i) = \begin{cases} 1, & \text{if } l_i \text{ is PI/PPI} \\ \sum_{\forall l_j \in Fanin(g)} C_{\overline{c}}(l_j), & \text{if } v = \overline{c} \oplus inv \\ \min_{\forall l_j \in Fanin(g)} \{C_c(l_j)\}, & \text{otherwise} \end{cases} \tag{3.3}$$

where $\oplus$ represents Boolean XOR operator, $Fanin(g)$ represents the fanin of gate g, and inv represents the Boolean inversion parity of gate g (Boolean 0 for AND/OR/BUF gates and Boolean 1 for NAND/NOR/NOT gates). To understand Equation 3.3, consider a NAND gate (the output is labeled l_i) for which inv=1 and c=0. To compute $C_0(l_i)$ (v= 1 $\oplus$ 1 = 0), we have to sum the 1-controllability of all of the fanin gates. However, to compute $C_1(l_i)$, we need to compute the minimum 0-controllability of all of its fanin gates. Since the time complexity of calculating $O(l)$ for signal line l is linear in the number of signal lines in the design, it can be calculated very fast. However, the SCOAP-like measure is fairly inaccurate due to the presence of reconvergent signals. This inaccuracy is illustrated using the circuit shown in Figure 3.3(a). To propagate a fault effect on l_O to s_p, line n should be set to 0 by specifying a=1, d=1, and f=0. Hence, $O(l_O)$=3. The observability of l is given by $O(l)$=$C_1(l_j)$+ $O(l_O)$. Since, l_j can be set to 1 by specifying a, b, and c to 1, $C_1(l_j)$=3. Hence $O(l)$ (= $C_1(l_j)$+ $O(l_O)$) is 6. However, since input a overlaps in $C_1(l_j)$ and $O(l_O)$, the real observability cost is 5. Hence, we present an improvement that computes more accurate estimations of $N_i(l)$ for any signal line l.

We use bitmaps to represent controllability and observability of signal lines as shown in Figure 3.3(b). Since a fault effect in l_O can be observed by specifying a, d, and f to 1, 1, and 0 respectively, the bits corresponding to a, d, and f are assigned 1's in the bitmap for $O(l_O)$ and all other inputs are assigned 0's. Inputs a, b, and c should be set to 1 to set l_j to 1. Hence, the bits corresponding to a, b, and c are assigned 1's in the bitmap for $C_1(l_j)$. The observability cost of line l is obtained by conducting a bitwise *ORing* of the bitmap for $C_1(l_j)$ with that for $O(l_O)$. Note that the bitmap for $O(l)$ shown in Figure 3.3(b) has exactly five bits that are assigned 1's. $N_i(l)$ is obtained by simply counting the number of 1's in the bitmap for $O(l)$.

A disadvantage of bitwise the ORing approach is that it requires a large memory space to store intermediate bitwise operation results. To reduce memory, we partition the circuit into a large number of circuit cones, as shown in Figure 3.2.

We consider one circuit cone C_i at a time and compute the gain values, $N_i(l)$ and $N_f(l)$ for each signal l in this cone. Memory allocated for the current circuit cone C_i is freed before we compute gain values for the next cone C_{i+1}. Hence, the memory complexity of our algorithm is bounded by the number of signal lines in the largest cone of the circuit and the number of inputs that drive the cone.

We use a two-pass algorithm to compute the value of $N_i(l)$ for each signal line l in cone C. All of the bitmaps associated with all of the signal lines l in C are initialized to 0's. In the first pass, we start the traversal from *primary inputs* (PIs) / *pseudo primary inputs* (PPIs) in a level order fashion towards the PO/PPO of cone C. In this pass we compute two bitmap sets for each signal line l where 1's in one bitmap set represent all of the PIs/PPIs that have to be specified to control the signal l to Boolean 1 and 1's in the other bitmap set represent all of the PIs/PPIs that have to be specified to control the signal l to 0. In the second pass, we start our traversal from the PO/PPO of cone C towards PIs/PPIs. In this pass, we compute the bitmap set for $O(l)$ for observing the fault effect in the signal line l at the PO/PPO of cone C. We obtain $N_i(l)$ for signal line l by simply counting the number of 1's in the bitmap set for $O(l)$.

To reduce run time to compute gain functions, we use a few inexpensive criteria to skip computing gain values for many signal lines that do not satisfy the criteria. This is referred to as *screening* and significantly reduces CPU time required to identify test points. This is based upon our observation that only a few signal lines in the entire circuit have very poor testability values. If the level order of the PO/PPO of cone C is below a threshold value or the total number of independent faults in C is below certain threshold, then we skip computing gain functions of all signal lines in cone C (as cone C is very small).

We use a divide-and-conquer approach for computing our gain functions to further reduce run time. We decompose the entire circuit into a network of *fanout free regions* (FFR) as shown in Figure 3.4. To compute the observability cost N_i for all signal lines in a particular FFR f_r, we first determine the set of inputs I_f that needs to be specified to observe a fault effect at the FFR output. *FFR*

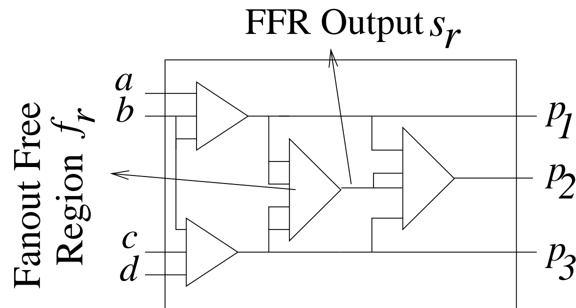


Figure 3.4: Dividing the Circuit into a Fanout Free Region (FFR) Network

output, s_r, is defined as the output signal line of that particular FFR f_r (see Figure 3.4). To compute the observability cost of a signal line l within the FFR f_r, we first compute the set of inputs I_l that has to be specified to observe the fault effect on l at FFR output s_r. Then, we conduct a bitwise ORing of the set I_l with the set I_f to compute the actual cost of $N_i(l)$. Note that we reuse the bitmap for $O(s_r)$ to compute observability costs for all signal lines in f_r. This reduces run time significantly because computational cost to observe a fault effect at line l is significantly less than the computational cost of observing it at a PO/PPO.

```
computeAccurateNi ( l)
1. Obtain the list of all the inputs D that drive signal lines
       in the off-path input of a path connecting l and PO/PPO
2. for i <- 1 to 3^|D|
3.    Generate a new input assignment by specifying the inputs
           in D to a new combination using 0, 1, X
4.    Perform 3-valued event driven logic simulation
5.    if( l is observable && # unspecified inputs in D < Ni(l) )
6.          Ni(l ) := # unspecified inputs in D;
```

Figure 3.5: Pseudo-code for Accurate Computation of $N_i(l)$

Experiments conducted with ISCAS '89 benchmark circuits clearly show that conducting bitwise ORing instead of simple additions improves accuracy of the observability measure $N_i(l)$. We computed accurate $N_i(l)$ to observe each signal line l by completely enumerating all possible input combinations and using a three-valued logic simulation to observe that signal l. Pseudo-code used for

accurate computation of $N_i(l)$ is given in Figure 3.5. This value is then compared against estimations computed by the bitwise ORing technique and the SCOAP-like measure. To compare the estimated values against the accurate values, we compute E_{rms} which is the *root mean square* of the errors of the estimates of all signal lines and is defined as:

$$E_{rms} = \sqrt{\frac{\sum_{\forall l}(\tilde{N}_i(l) - N_i(l))^2}{N_{total}}} \tag{3.4}$$

where $\tilde{N}_i(l)$ is an estimation of $N_i(l)$ and N_{total} is the total number of signal lines. In Table 3.1, the column labeled *Bitwise ORing* represents the E_{rms} for the estimation using bitwise ORing. The column labeled *SCOAP-like* shows the E_{rms} of the estimation using Equation 3.2. Our experiments with the observation points also show that this inaccuracy can have adverse impact on the final results (see Chapter 6).

Table 3.1: Bitwise ORing with SCOAP-like Measure Comparison

Benchmark	*Bitwise ORing*	*SCOAP-like*
$s9234$	1.5	6.3
$s13207$	2.1	4.8
$s15850$	1.6	4.9
$s35932$	2.5	4.2
$s38417$	1.3	3.6

3.2.2.2 Computing N_f

To compute N_f for all of the signal lines in a cone C, we first compute the total number of independent faults in each of the individual FFRs f_r in the circuit cone C. Since no reconvergent fanouts exist in a FFR, exact values for the number of independent faults in a FFR can be computed by simple additions, thereby reducing run time. We then go into each of these FFRs and compute $N_f(l)$ for each signal line l within the FFR.

Also, as identifying independent faults in a circuit is expensive, we approximate the total number of independent faults by the total number of collapsed

faults. A highly collapsed fault set, F_m, for the circuit is constructed using an algorithm described below. The algorithm begins with an initial collapsed fault set F_0, which comprises all of the stuck-at 0 and stuck-at 1 faults in the circuit. Thus, initially the size of F_0 is $2 \times n$, where n is the number of signal lines in the circuit. Starting from the PIs/PPIs, we consider a fault $f \in F_0$ and remove all the faults f_c from the set F_0 that are equivalent to f or that dominate f to get a reduced fault set F_1. Note that two faults f and f_c are said to be equivalent if every pattern that detects fault f_c also detects f and vice-versa. A fault f dominates fault f_c if every pattern that detects f also detects f_c [13]. We continue reducing the fault set F_j, where j=0, 1, 2, ..., m (m is the last iteration count) until we are unable to reduce it any further.

3.2.3 Mergeability

It is important to generate large numbers of X's in test patterns. These X's should be placed at the appropriate locations in the test pattern. Only then will these X's facilitate the test generator to merge many patterns to obtain compact test sets. To achieve this, we perform a topology based mergeability analysis. Consider the case when all of the generated X's are placed in one test pattern. In that case, the generated X's obviously do not contribute to any reduction in the test length. Hence, we need to spread the X's across different locations in test patterns to obtain compact test sets.

Consider three signal lines l_1, l_2, and l_3. N_f and N_i values for each of these signal lines are given in Table 3.2. Note that all three signals have the same gain value. However, signal line l_1 is located very close to one of the primary outputs driven by a large fanin cone comprising 625 faults and requires only setting one PI or one PPI to be observed. Inserting an observation point at this location will generate only one X at each test pattern that detects these 625 faults. Since these X's are all aligned in one column of the resulting test vectors, the X's generated cannot be effectively used for merging.

Table 3.2: Example to Describe Mergeability

Signal Line	N_f	N_i	$G(l) = N_f \times N_i$
l_1	625	1	625
l_2	1	625	625
l_3	25	25	625

It can be seen that signal line l_2 is located very close to one of the primary inputs (only one fault in the fanin cone of l_2). However, to observe this signal we have to specify 625 inputs. Inserting an observation point into l_2 will not be very beneficial either, because all generated X's will belong to the same test pattern. Thus, even though it can be merged with another test pattern, we can only reduce the test pattern count by at most one.

The third signal line l_3 represents a desirable location for test point insertion. Inserting an observation point into l_3 will generate 25 X's which will be divided among 25 test patterns (assuming all 625 faults are independent). Thus, in the case of l_3 there is a higher chance that test patterns generated are merged with other patterns, thereby reducing the pattern count significantly. To facilitate the test generator to merge many patterns, we need to spread the generated X's across different locations in test patterns. To do this, we maintain a Boolean priority flag called $MERGEABILITY$ for each signal line l. While selecting signal lines for test point insertion, we give preference to signal lines whose $MERGEABILITY$ flags are high. If either the $N_f(l)$ or $N_i(l)$ value of a signal line l is below a certain threshold value, then its $MERGEABILITY$ flag is set to low. Otherwise it remains high.

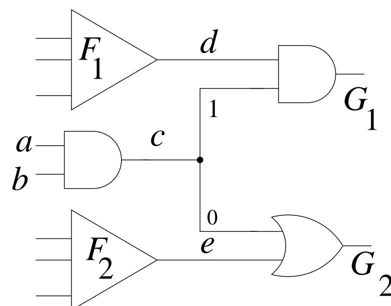


Figure 3.6: Illustration for Mergeability

Consider Figure 3.6 where a stem signal c branches to two conjugate gates G_1 and G_2 (AND/NAND gates are conjugates of OR/NOR, respectively) as shown in Figure 3.6. Let gates G_1 and G_2 have large numbers of independent faults in their fanin cones F_1 and F_2, respectively. Test patterns generated for faults within the cones F_1 and F_2 can never be merged because signal lines d and e cannot be simultaneously observed (Setting c to 1 to observe line d at the output of G_1 makes signal line e unobservable). However, inserting an observation point at d or e can make test patterns that detect faults in cones F_1 and F_2 merged together, thereby reducing the test length further. Thus, $MERGEABILITY$ flags for signal lines such as d and e are also set to high.

3.2.4 Updating Gain Values

Once a test point is inserted into a signal line in cone C, the gain values for all of the signal lines in C are updated to reflect the extra observation point that has been added into the circuit. If a test point is inserted at a signal line l, then we would not want to insert another test point in the neighborhood of l. But recomputing the gain values for all signals in the cone C after each test point insertion is also very expensive. Hence, we use an inexpensive approach that consumes very little CPU time to update the gain values after inserting a test point.

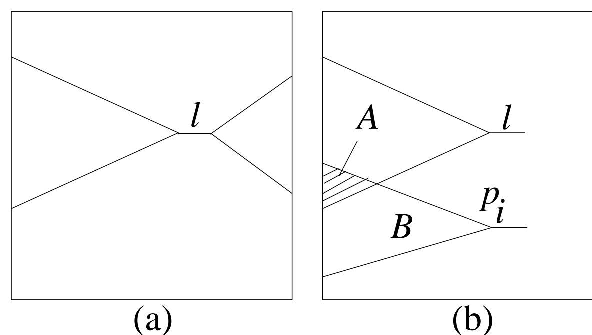


Figure 3.7: Updating the Gain Value: a) Fanin Cone and Fanout Cone and b) Overlapping Fanin Cones

When an observation point is inserted at signal line l, all signals in the fanin

and fanout cones of l are not considered as candidate test points. This is because the test point inserted at l will observe most of the faults in the fanin cone of l (see Figure 3.7(a)). Any test point inserted in the fanin cone or the fanout cone of l may not reduce the test set significantly. If the fanin cone of a candidate test point p overlaps with the fanin cone of l where a test point is already inserted, then we need to update N_f for all signals in the fanin cone of p (since the faults in the overlapping area of the two fanin cones can be detected at the observation point at l as a test point is already inserted at l). Figure 3.7(b) shows overlapping area A of the fanin cones of l and p_i. Hence, we update $N_f(p_i)$ of every candidate test point p_i (i = 1, 2, ...) whose fanin cone overlaps with the fanin cone of l where a test point is already inserted. The updated number of independent faults in the cone of p_i, i.e., the new $N_f(p_i)$, is obtained by subtracting the number of independent faults in A from the original $N_f(p_i)$.

3.3 Algorithm Outline

We describe the outline of the procedure of our TPI tool.

1. Identify signal lines that belong to the timing critical and layout congested areas and exclude these signals from the list of candidate test points.

2. Consider a circuit cone C of the circuit. Let the PO/PPOs of the cone C be s_p.

3. If the level order of s_p is smaller than a certain threshold value (see Section 3.2.2.1), then go to Step 2. Five was used as the threshold for our experiments.

4. Compute $N_f(l)$ for each signal $l \in C$ as described in Section 3.2.2.1.

5. Compute $N_i(l)$ for each signal line $l \in C$.

6. Sort the list of signal lines based upon their gain values.

7. Mark the $MERGEABILITY$ flag for each signal $l \in C$ as described in Section 3.2.3. When two or more signals have the same gain function, then their $MERGEABILITY$ flags are used to select a candidate signal line for inserting a test point.

8. Insert a test point at a signal line that has the highest gain value as described in Section 3.2.

9. If the desired number of TPs is successfully inserted then exit, else update N_f of signal lines in cones that overlap with C (described in Section 3.2.4) and go to Step 8.

3.4 Complexity Analysis

The computation of our testability measures (see Section 3.2.2) involves a depth-first traversal from each of the PIs/PPIs to POs/PPOs (forward pass) to compute the controllability measures and another depth-first traversal from each of the POs/PPOs to PIs/PPIs (backward pass) to compute the observability measures. The time complexity for this operation is $O(k \times n)$, where n is the total number of signals in the circuit and k is the upper bound of the size of the bitmap used for each of the each signal. The value of k is governed by the number of PIs/PPIs in the largest circuit cone (see Section 3.3) of the design. Since $k << n$, k can be assumed as a constant. Hence, the cost is $\approx O(n)$. The complexity of fault collapsing is also $O(n)$ because it involves a depth-first traversal where each signal is visited only once. The cost for the mergeability analysis is $O(c \times n)$ where c is the largest fanout size. Since c is typically a small number, the overall time complexity of the proposed algorithm is $O(n)$. The results of the run-times presented in Chapter 6 for the TPI tool confirm this. The memory complexity of our algorithm is $O(k \times n_c)$ where n_c is the upper bound of the total number of signals in the largest circuit cone of the design.

3.5 Summary

In this chapter, we presented a new technique to insert observation points into *structured ASIC* (SA) designs. The proposed TPI technique can reduce test generation time and test data volume for structured ASICs. The observation points are implemented using the unused flip-flops, which exist in any structured ASICs in a large quantity. This is described in detail in Section 3.2.1. Since only very limited numbers of observation points are inserted to minimize any negative impact on yield and performance, a gain function is computed for every potential signal line to select the best places to insert observation points. The gain function for a line reflects the number of independent faults in the fanin cone of the line and the number of inputs that should be specified to propagate a fault effect at that line for observation. This was described in detail in Section 3.2.2. In Section 3.2.3, we introduced another testability criterion called Mergeability. Mergeability helps the ATPG tool to generate test vectors that can be easily compacted by a static compactor tool. Section 3.2.4 describes how inserting an observation point affects the gain values of other signal lines in the circuit and how these gain values can be corrected. Finally, Section 3.3 presented the overall algorithm for inserting the requested number of observation points into the SA design.

Chapter 4
Control Points

In this chapter, we describe different types of control points that can be implemented using the unused *multiplexers* (MUXes) and SFFs of a SA design. We convert unused hardware in SAs into: a) conventional control points, b) complete test points, c) pseudo-control points, or d) inversion test points. Since only unused hardware is used, the proposed TPI technique does not entail any extra hardware overhead. Test points are inserted using timing information, so they do not degrade performance. We also present novel gain functions that quantify the reduction in test volume and ATPG time due to TPI and are used as heuristics to guide the selection of signal lines for inserting test points.

4.1 Introduction

While several prior efforts [3, 24, 27] have used CPs to improve digital circuit testability, the method presented here differs significantly from all other prior work. We present different types of TPs that can be implemented by re-configuring the unused MUXes and scan *flip-flops* (flops) that already exist in an SA design to drastically reduce test data volume and test generation time. We show how scan flops can be used to enable/disable different types of TPs without using a test enable signal, while also minimizing any routing overhead. We present a novel gain function that quantifies the reduction in test data volume and ATPG time due to inserting a particular TP at a signal line l. We also present efficient algorithms to compute the gain function and update the gain functions of all other signal lines after incorporating a particular TP at l. Later, we describe various

types of TPs and their impact on test data volume and test generation time.

Typically, 30-40% of the hardware in SAs is unused. The proposed TPI technique re-configures this unused hardware into test points to reduce test volume and ATPG CPU time. Since, in an SA design the *test enable* signal is inaccessible from metal layer(s) that implement the user design, we use unused scan flops to enable/disable TPs. We also present gain functions that mathematically quantify the benefits obtained by inserting a particular type of test point into the design. In Chapter 3, we showed that maximizing don't cares in test patterns can reduce the test volume. This is because don't cares facilitate both the static compaction and the dynamic compaction process to compact the test data. The proposed gain functions are used to select signal lines in which inserting test points maximizes the reduction in test data volume and ATPG time.

4.2 Control Points and their Implementation

In this section, we present four different TP types: a) a *conventional 0/1 CP*, b) a *complete test point* (CTP), c) a *pseudo-control point* (PCP), and d) an *inversion test point* (ITP). We now describe these TPs and a gain function for each TP.

4.2.1 Conventional Control Point (CP)

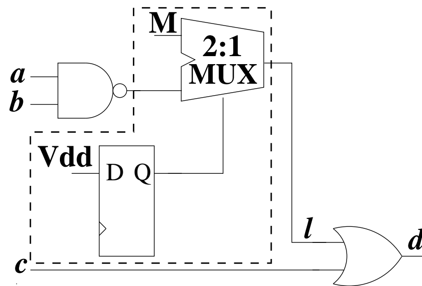


Figure 4.1: Conventional Control Point

Figure 4.1 shows an implementation of a *conventional 0/1 control point* (CP) using an unused scan flop and a MUX. Line M is tied to V_{dd} (V_{ss}) to implement a 1 (0)-CP. In functional mode, the scan flop is initialized to logic 1 and hence,

line l is driven by the NAND gate output. However, in test mode, the scan flop can be loaded with either logic 1 or 0 using the scan path. If logic 0 is loaded into the flop, then M drives l to 1/0 depending on whether M is tied to V_{dd} or V_{ss}. However, if logic 1 is loaded into the scan flop during test mode, then the CP is switched off.

Let us consider the gain in test volume reduction due to inserting a conventional CP at l. Let $N_C^1(l)$ ($N_C^0(l)$) represent the number of PIs/PPIs that must be specified to control l to 1 (0), let $F_C^1(l)$ ($F_C^0(l)$) represent the number of independent faults that are excited due to the CP at l and $F_O^1(l)$ ($F_O^0(l)$) represent the number of independent faults that pass through the AND/NAND/XOR/XNOR (OR/NOR/XOR/XNOR) gate g where l is an input of g. The gain function G_{CP}^1 (G_{CP}^0) for the 1 (0)-CP is expressed as this theorem:

Theorem 4.2.1 *The lower bound of the total number of X's additionally generated in the resulting test set by inserting a conventional 1 and 0 CP at signal line l is:*

$$G_{CP}^1(l) = (F_C^1(l) + F_O^1(l)) \times N_C^1(l) \tag{4.1}$$

$$G_{CP}^0(l) = (F_C^0(l) + F_O^0(l)) \times N_C^0(l) \tag{4.2}$$

Proof. First consider inserting a 1 CP at l. ATPG has to generate at least $F_C^1(l)$ vectors to excite the faults in the fanout of l and at least $F_O^1(l)$ vectors to observe the faults in the fanin cone of the off-path signals of l (because all of the $F_C^1(l) + F_O^1(l)$ faults are independent). Now, if we insert a 1 CP at l, then ATPG need not specify $N_C^1(l)$ inputs that are necessary to set l to 1. Hence, the test patterns for $F_C^1(l) + F_O^1(l)$ faults will have at least $N_C^1(l)$ additional X's. Therefore, the lower bound of the number of additional X's generated by inserting a 1 CP at l in the resulting test set is $(F_C^1(l) + F_O^1(l)) \times N_C^1(l)$. A similar argument shows that $G_{CP}^0(l)$ in Eqn. 4.2 represents the lower bound of the number of additional X's generated by inserting a 0 CP at l in the resulting test set. □

A disadvantage of the conventional CP is that it can only improve either 1- or 0-controllability of a signal line, but not both. Next, we describe a *complete test point* (CTP) that combines an OP and a CP.

4.2.2 Complete Test Point (CTP)

A CTP is a test structure that combines a CP and an OP. Hence, it improves both both 1- and 0-controllability and also the observability of another signal line. It is implemented using two scan flops and a MUX. The dotted boxes in Figure 4.2 show two implementations of a CTP. In functional mode, scan flop $FF2$ should be set to Boolean 1. In test mode, scan flop $FF2$ is set to Boolean 0 to turn on the TP and to Boolean 1 to turn off the TP.

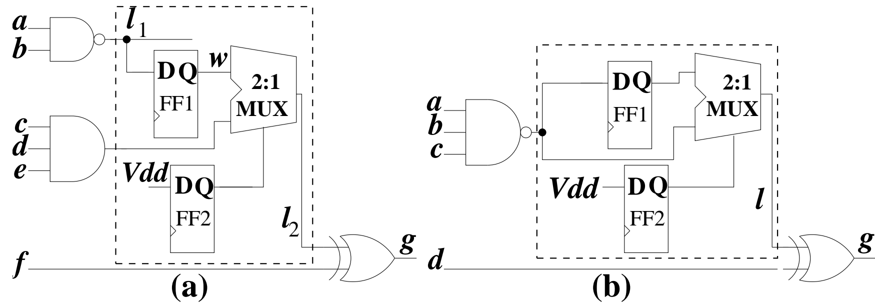


Figure 4.2: Complete Test Point

In Figure 4.2 (a), the fault effect at signal line l_1 is observed using scan flop $FF1$. The same flop simultaneously controls another signal line l_2 using an additional MUX. To control l_2, scan flop $FF2$ should be initialized to Boolean 0 and Boolean 0 (1) is loaded in $FF1$ to drive l_2 to Boolean 0 (1). A disadvantage of this implementation is that there is significant extra routing cost if signals l_1 and l_2 are located far apart in the physical design. To overcome this, we use another implementation of the CTP shown in Figure 4.2 (b) to observe and control the same signal line l. The gain in test volume reduction due to inserting a CTP at l is expressed as a theorem below:

Theorem 4.2.2 *The lower bound of the total number of X's additionally generated in the resulting test set by inserting a CTP at signal line l is:*

$$G_{CTP}\ (l) = G^1_{CP}\ (l) + G^0_{CP}\ (l) + G_{OP}\ (l) \tag{4.3}$$

Proof. When the CTP at l is used as a 1 (0)-CP, the additional number of don't cares (X's) that result in the test set is $G^1_{CP}\ (l)$ ($G^0_{CP}\ (l)$). Since scan flop $FF1$ can observe all fault effects flowing through l, at least $G_{OP}\ (l)$ additional X's are generated (from Eqn. 4.3). Also, by definition, there are no overlapping faults among $G^1_{CP}\ (l)$, $G^0_{CP}\ (l)$ and $G_{OP}\ (l)$. Hence, the sum $G^1_{CP}\ (l)+G^0_{CP}\ (l)+G_{OP}\ (l)$ is a lower bound on the additional X's that are generated in the resulting test set. □

Another disadvantage of the CTP is that it adds a MUX delay into the functional path and, hence, cannot be inserted into timing critical paths. Next, we present a *pseudo-control point* that can improve the signal line controllability without adding any extra delay.

4.2.3 Pseudo Control Point (PCP)

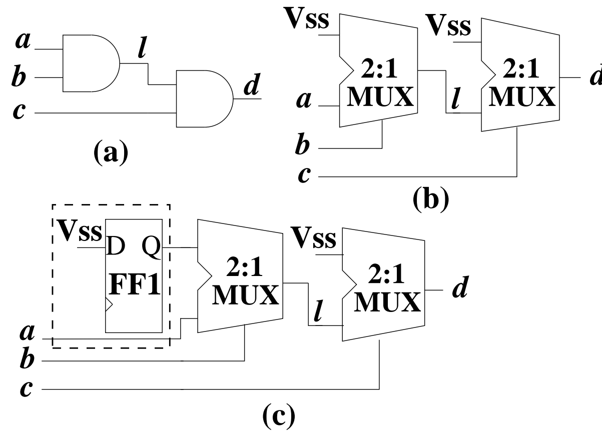


Figure 4.3: Pseudo Control Point

A *pseudo-control point* (PCP) is a control structure that does not add any functional path delay. Hence, it can improve controllability of the design without

degrading the circuit timing. It exploits the fact that in an ISSP design, several signal lines are tied to V_{dd}/V_{ss} to implement the user logic. For example, Figure 4.3 (b) shows the physical realization of the netlist of Figure 4.3 (a). The dotted box in Figure 4.3 (c) shows a PCP to improve the controllability of signal line l. In the original netlist (Figure 4.3 (a)), to set l=1 both a and b should be set to 1. This may require a large number of inputs to be specified if both a and b are driven by large fanin cones. We improve controllability at l by inserting a PCP (see Figure 4.3 (c)). The scan flop $FF1$ should be initialized to 0 in functional mode. In test mode, l can be set to 1 by setting b=0 and enabling the PCP, i.e., FF=1. This is much easier than setting both a and b to 1. The PCP is turned off by setting $FF1 = 0$. Since we add no extra logic into the functional path, inserting a PCP will not degrade the circuit timing and, hence, a PCP can even be used to improve the controllability of signal lines on timing critical paths. The gain in test volume reduction due to inserting a PCP at l is expressed as a theorem below:

Theorem 4.2.3 *The lower bound of the total number of X's additionally generated in the resulting test set by inserting a PCP at signal line l is:*

$$G^{v}_{PCP}\ (l) = (F^{v}_{C}\ (l) + F^{v}_{O}\ (l)) \times \Delta N^{v}_{C}\ (l) \tag{4.4}$$

where $v(= 0\ or\ 1)$ *represents whether we are improving the 0- or 1-controllability of signal* l *and* $\Delta N^{v}_{C}\ (l)$ *is defined as:*

$$\Delta N^{v}_{C}\ (l) = N^{v}_{C}\ (l) - N^{\overline{v}}_{C}\ (l) \tag{4.5}$$

where $N^{v}_{C}\ (l)$ *is the number of PIs/PPIs that must be specified to set* l *to logic* v *and* $N^{\overline{v}}_{C}\ (l)$ *is the number of PIs/PPIs that must be specified to set* l *to* $\overline{v}$*.*

Proof. From the definition of a PCP, inserting a PCP at l means we can set l to logic v by specifying $N^{\overline{v}}_{C}\ (l)$ PIs/PPIs and setting the scan flop of the PCP to logic v. Thus, the total number of PIs/PPIs that need not be specified to set signal l to logic v is $N^{v}_{C}\ (l) - N^{\overline{v}}_{C}\ (l)$ $(= \Delta N^{v}_{C}\ (l))$. Also by the definitions of

$F_C^v(l)$ and $F_O^v(l)$, the total number of independent faults that benefit due to the PCP at l is $F_C^v(l) + F_O^v(l)$. Hence, at least a total of $(F_C^v(l) + F_O^v(l)) \times \Delta N_C^v(l)$ additional X's will be generated in the resulting test set. □

A PCP can also be implemented for other medium and coarse-grained SA architectures [74]. Consider a 3-input *lookup table* (LUT) based SA [63] implementing the Boolean logic $F{=}A\overline{B}C + AB\overline{C}$ shown in Figure 4.4 (a). The symbol

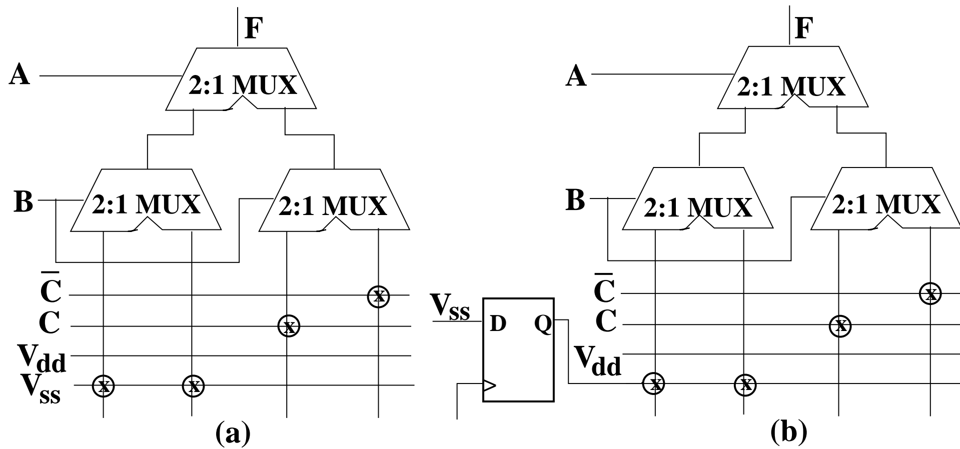


Figure 4.4: A 3-input LUT Implementing $F{=}A\overline{B}C + AB\overline{C}$

$\otimes$ means the horizontal and the vertical wires are shorted. Now, setting $F{=}1$ is very difficult because ATPG will have to specify all three inputs A, B, and C. Figure 4.4 (b) shows a PCP in which a scan flop is used instead of V_{ss}. During functional mode, the scan flop is initialized to logic 0. During test mode, the scan flop can be initialized to logic 1 and by setting $A = 0$, we get $F = 1$. Since ATPG now has to specify only one input, ATPG CPU time is drastically reduced. Test volume is reduced because the other two inputs B and C can now be specified to detect several other faults.

4.2.4 Inversion Test Point (ITP)

ITPs [13] can be implemented in regular cell-based ASICs using an XOR gate. Since SA designs only have unused MUXes and flops, we show how an ITP is implemented with these elements. The dotted box in Figure 4.5 shows the extra

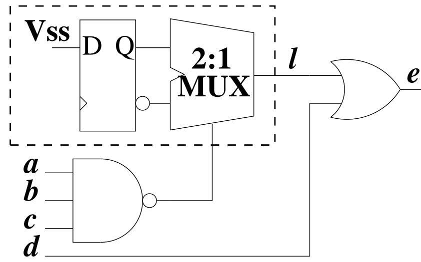


Figure 4.5: Inversion Test Point

DFT hardware added to implement an ITP. In functional mode, the scan flop is initialized to logic 0 and hence the design functionality remains the same. However, in test mode the scan flop is initialized to logic 1/0 through the scan path. When initialized to 1, l always takes the Boolean complement (inversion) of the NAND gate output. If the scan flop is set to 0, the TP is switched off. A disadvantage of the ITP is that a MUX delay is added into the functional path of the design. Hence, timing information should be used when an ITP is inserted into the design.

To understand how an ITP can reduce test volume and ATPG CPU time, consider a line l for which 0-controllability is very high whereas 1-controllability is very low. Using an ITP at l, the 0-controllability can be significantly improved, because now to generate a logic 0 we can generate a logic 1 and invert it using the ITP. Thus, the total effort required to generate logic v at any signal line l is reduced by $N_C^v(l) - N_C^{\overline{v}}(l)$ ($= \Delta N_C^v(l)$ in Eqn. 4.5) assuming $N_C^v(l) > N_C^{\overline{v}}(l)$. Since by definition, $F_C^v(l) + F_O^v(l)$ faults benefit due to a TP at l, the total gain from inserting an ITP at l is:

$$G_{ITP}^v(l) = (F_C^v(l) + F_O^v(l)) \times (N_C^v(l) - N_C^{\overline{v}}(l)) \tag{4.6}$$

which is similar to the PCP gain function. However, there are several differences between a PCP and an ITP. For example, a PCP can be inserted on critical paths whereas an ITP cannot. Also, a PCP cannot be inserted on signal lines whose fanins are not tied to V_{dd}/V_{ss} whereas an ITP can be.

We described various TP types that can be implemented using the unused

hardware that are already available in a SA design. We also described mathematically (Eqns. 4.2-4.6) the reduction in test volume and ATPG CPU time achievable by inserting a particular type of TP at line l. Next, we describe efficient algorithms to pick signal lines for TPI that maximize the reduction in test volume and ATPG CPU time.

4.3 Algorithm

The gain functions described in Equations 4.2-4.6 are used as heuristics to select signal lines for inserting a particular TP. To compute gain functions for all types of TPs, we use a two-pass algorithm. In the first pass, we compute $N_f\ (l)$, $N_i\ (l)$, $N_C^v\ (l)$, $F_O^v\ (l)$, and $F_C^v\ (l)$ for all circuit signal lines. In the next pass, we calculate the gain functions for all signal lines using Eqns. 4.2-4.6. Since this is compute intensive, we approximate wherever necessary. We also present several optimizations to significantly reduce run time and memory usage of our TPI tool.

4.3.1 Controllability and Observability Cost

The controllability cost $N_C^v\ (l)$ $(v = 0/1)$ for a signal line l is the total number of PIs/PPIs that need to be specified to set l to logic v. The observability cost $N_i\ (l)$ is the total number of PIs/PPIs that need to be specified to propagate the fault effect at l to a PO/PPO. If the design has reconvergent fanouts, then calculating accurate controllability and observability for signal l is difficult. In Chapter 3, we showed a SCOAP-like [21] measure that will give us inaccurate controllability and observability measures (see Section 3.2.2.1). This can be overcome using bit maps. Three bit arrays, denoted $CC_1\ (l)$, $CC_0\ (l)$, and $O\ (l)$, are used for each signal line l to store the 1-, the 0-controllabilities, and the observabilities, respectively. The jth entry of the controllability arrays of line l tells whether the jth PI/PPI in that circuit cone controls l or not. Similarly, the jth entry of the observability array, $O\ (l)$, tells whether the jth PI/PPI needs to be specified or not to observe l at the PO/PPO of the circuit cone.

A disadvantage of using a bit map is that it requires a large memory space to store intermediate bit operation results. To reduce memory, we partition the design into a large number of circuit cones C_i, $i = 1, 2, \ldots$. A particular cone C_i comprises one PO/PPO, denoted p, and all signal lines located in the *transitive fanin* of p. Memory allocated for computing the testability measures ($N_C^v\ (l)$ and $N_i\ (l)$) of all signal lines in C_i is freed before considering the next circuit cone. This approach drastically reduces the peak run time memory consumed by our TPI tool.

A two-pass algorithm computes testability values of all lines l in the cone C_i. All of the bit arrays (CC_0, CC_1, and O) of all signal lines l in C_i are initialized to 0's. In the first pass, we traverse from PIs/PPIs in level order toward the PO/PPO of cone C_i, computing both $CC_1\ (l)$ and $CC_0\ (l)$ for each signal line l. Since a 1 in the bit array $CC_1\ (l)$ ($CC_0\ (l)$) means that particular PI/PPI must be specified to set l to 1 (0), counting the number of 1's in $CC_1\ (l)$ ($CC_0\ (l)$) gives the value of $N_C^1\ (l)$ ($N_C^0\ (l)$). In the second pass, we traverse from the PO/PPO of C_i toward PIs/PPIs. In this pass, we compute the bit array $O\ (l)$ for observing the fault effect at each signal line l at the PO/PPO of this cone. $N_i\ (l)$ for signal line l is the number of 1's in the bit array for $O\ (l)$.

Next, we describe procedures to compute N_f, F_C^v, and F_O^v for all circuit signal lines. Since this computation is not memory intensive, the procedure does not use partitioning.

4.3.2 Computing $N_f\ (l)$, $F_C^v\ (l)$, and $F_O^v\ (l)$

Testability measure $F_C^v\ (l)$ is the total number of independent faults that are excited when a v-CP, where $v = 0/1$, at signal line l is switched on. Testability measure $F_O^v\ (l)$ is the total number of independent faults that pass through all off-path signals of l by setting l to logic value v. Computing $N_f\ (l)$, $F_C^v\ (l)$, and $F_O^v\ (l)$ involves counting the number of independent faults but identifying independent faults in a circuit is very computation intensive. So, we approximate

the total number of independent faults by the total number of collapsed faults and this is described next.

4.3.2.1 Computing $N_f\ (l)$

After obtaining the collapsed fault list, we then compute $N_f\ (l)$, $F_C^v\ (l)$, and $F_O\ (l)$ for all signal lines l. All $N_f\ (l)$ values are initially zeroed. The algorithm propagates each fault f guided by the observability cost, N_i, toward a PO/PPO. When a fault effect propagates through line l, $N_f\ (l)$ is incremented. When all faults are propagated, we obtain N_f values for all circuit signal lines.

4.3.2.2 Computing $F_C^v\ (l)$ and $F_O^v\ (l)$

We compute $F_C^v\ (l)$ and $F_O^v\ (l)$ from the $N_f\ (l)$ values. We first initialize all $F_C^v\ (l)$ values in the circuit to 0. Then, we assign a logic value v to l and imply the Boolean value to as many signal lines as possible that are located in the transitive fanout of l. When a Boolean value v_f propagates to line l_f, such that the fault stuck-at $\overline{v_f}$ at l_f belongs to the final collapsed fault list FS_m, then $F_C^v\ (l)$ is incremented. When this operation is performed for all circuit signal lines, we obtain $F_C^v\ (l)$ values for the entire circuit.

To compute $F_O^v\ (l)$, the algorithm identifies the list of all gates G_f such that: a) l is a fanin of G_f and b) v is a non-controlling value of G_f. Then, it obtains all fanin signals, F_{in}, of all gates G_f excluding l, so $\sum_{f_{in} \in F_{in}} N_f\ (f_{in})$ represents $F_O^v\ (l)$.

So far we presented procedures to compute the gain functions for all types of TPs described in Eqns. 4.2-4.6. After inserting a particular type of TP in the circuit, we have to update the testability values of all nearby signal lines.

4.3.3 Updating Testability Measures

Once a TP is inserted into line l, the gain values for all lines in the neighborhood of l are updated to reflect the extra TP that has been added. When an OP

is inserted at l, we first identify all lines L_{in} in the fanin cone of l such that a fault effect from $l_{in} \in L_{in}$ passes through l before reaching a PO/PPO before inserting an OP. We then subtract the value of $N_i\ (l)$ from $N_i\ (l_{in})$, because the fault effect at l_{in} will now be observed at l itself. We then obtain all signal lines L_{out} in the fanout cone of l such that fault effects from l pass through $l_{out} \in L_{out}$ before reaching a PO/PPO before inserting an OP. Since $N_f\ (l)$ faults will now be observed at l, we subtract the value of $N_f\ (l)$ from $N_f\ (l_{out})$ to update the N_f values.

When a CP, PCP, or ITP is inserted at l, the controllabilty of l and signals in the transitive fanout of l will also decrease. Let ΔCC^v represent the reduction in v-controllability of line l due to inserting a TP (CP, PCP, or ITP) where $v = 0/1$. To update gain values of other lines, we obtain the set of all lines L_{DI} that are directly implied by setting l to logic v. We then obtain the list of all output signals $l_{DI}^{out} \in L_{DI}^{out}$ such that: a) $l_{DI}^{out} \notin L_{DI}$ and b) l_{DI}^{out} is a fanout signal of some $l_{DI} \in L_{DI}$. Thus, the entire set of signals $l_{all} \in L_{DI} \cup L_{DI}^{out}$ represent signal lines that directly benefit from inserting any CP that improves the controllability of l. Then, we decrement the value $\Delta CC^v\ (l)$ from the controllabilty values of all signal lines $l_{all} \in L_{DI} \cup L_{DI}^{out}$.

Since inserting a CTP is equivalent to inserting a 0-CP, 1-CP, and OP, we call the update routines of 0-CPs, 1-CPs, and OPs, as described above, when a CTP is inserted. This completes the procedures for updating testability measures after inserting a particular TP.

4.3.4 Overall Algorithm

We describe the overall algorithm to insert a requested number of TPs into the design. Figure 4.6 shows the top-level algorithm for inserting N TPs. The variable $TPType$ represents the user requested TP type and can be OP, CP, CTP, PCP, or ITP. To insert a combination of different types of TPs, we can invoke the routine $insertNTestPoint$ () multiple times with different $TPType$ values.

```
insertNTestPoint( N, TPType, ckt )
1. computeGainFunction (ckt, TPType)
2. createCandidateTestPoints ( C_SIZE, TPType, &candidateList)
3. for i <- 1 to N
4.     obtainNextTP ( candidateList, &tp)
5.     storeThisTP ( &TPList, tp)
6.     updateTestability( tp, ckt, TPType)
7. insertTPIntoDesign( TPList, ckt)
```

Figure 4.6: Pseudo-code for the Overall Algorithm

We now describe the algorithm in detail. Procedure *computeGainFunction* () computes the gain function for all signal lines in the circuit as described in Section 4.3. Procedure *createCandidateTestPoints* () creates a list comprising C_SIZE signals with highest gain values. This list represents the search space of our TPI tool. Since searching only C_SIZE signals for TPI purposes is less expensive than evaluating all circuit signals, this step greatly reduces the total run time of our TPI tool. In our experiments, we used $C_SIZE = 10 \times N$. Procedure *obtainNextTP* () gets the signal line with the highest gain function from the candidate list. Procedure *storeThisTP* () stores the chosen signal line in a list and is later used by *insertTPIntoDesign* (), which writes the design along with all inserted TPs back onto the disk. Procedure *updateTestability* () updates the value of testability measures of all signal lines in the candidate list due to inserting the chosen TP (see Section 4.3.3).

4.4 Complexity Analysis

As described earlier in Section 3.4, the cost for computing the controllability and observability measures along with fault collapsing is $O(n)$ where n is the number of signal lines in the circuit. The cost for computing the F_C and F_O values for all of the signal lines in the circuit is also $O(n)$. This is because during the entire computation we propagate a Boolean value (0/1) from each signal line and visit each node in the circuit only once. So the overall cost for the algorithm presented in Section 4.3.4 is $O(n)$.

4.5 Summary

In this chapter, we presented four different test points: a) conventional control points, b) complete test points, c) pseudo-control points, and d) inversion test points. We also described how these test points can be implemented in an SA design using the unused *multiplexors* (MUXes) and the SFFs, which exist in any SA design in large quantities. This is described in detail in Section 4.2. In this section, we also presented gain functions that mathematically quantify the benefits obtained by inserting a particular type of test point into the design. We, then, presented efficient algorithms in Section 4.3 to accurately compute the gain values. Inserting a particular test point can affect the gain values of other signal lines in the circuit. Hence, in Section 3.2.4, we described an updation algorithm to correct the gain values. Finally, Section 4.3.4 presented the overall algorithm for inserting the requested number of observation points into the SA design.

Chapter 5
Test Points for a Hardware Compressor

In this chapter, we present a new TPI technique for full scan designs employing a *broadcast scan*-based compressor for test data compression. In the last two chapters, we presented TPI techniques for structured ASICs that do not use any hardware test data compression scheme. However, in this chapter, we present a TPI technique along with a scan chain re-ordering technique for regular cell-based ASIC designs that already have a broadcast scan compressor. We present the algorithm for TPI in Section 5.3. In this section, we also present gain functions to identify the best signal lines for inserting test points. Next, in Section 5.4, we present a layout-aware scan chain re-ordering algorithm that further reduces the test volume.

5.1 Introduction

As mentioned earlier, shrinking feature size has aggravated the testing problem because new types of defects such as crosstalk noise, small delay defects, etc., are manifested in the device. Now test engineers require even more test data to test for these defects, further exacerbating the problem of increasing scan test data volume and test application time. This has led the researchers to present several compression techniques to address the growing test volume problem (see Chapter 2). The two popular input test pattern compression techniques widely used in the industry are: a) the *linear feedback shift register* (LFSR) reseeding-based technique [37] and b) the *broadcast scan-based compression* technique [39].

In the LFSR reseeding scheme, test data are encoded into *seeds* and are stored

in the tester memory. During the test application, the seeds are loaded into an on-chip LFSR that expands each seed into a test vector, which is then applied to the DUT. However, the major disadvantage of this scheme is that additional procedures are required to characterize and solve a system of linear equations to obtain the seeds from the test set. The total number of stages of the LFSR should also be carefully chosen as linear dependencies in the LFSR sequence may sometimes lead to an unsolvable system of linear equations. Additional hardware is also required to implement the on-chip LFSR, which could become expensive if the number of specified bits in test cubes is large.

In the broadcast scan scheme [39], a single scan chain is divided into multiple small scan chains such that each of these small scan chains is driven by a single scan input. Additional constraints are added to the ATPG tool to generate the broadcast scan test data. A disadvantage of this scheme is that since the same test data is "broadcasted" to multiple scan chains, several useful test vectors cannot be applied to the DUT. Due to this artificial constraint, the compression ratio achieved by the broadcast scan scheme is generally inferior compared to the LFSR reseeding technique. However, the major advantage of the broadcast scan scheme is its simplicity and the ease of implementation.

In this chapter, we will present a novel two-step *design-for-testability* (DFT) technique that can improve the performance of the broadcast scan scheme. In the first step, we insert test points into the design that break correlations among different signal lines in the design that are introduced due to the artificial broadcast scan constraints. The second step of our scheme is the layout aware scan chain re-ordering process that further breaks these correlations and drastically improves the performance of the broadcast scan-based test compressor.

5.2 The Proposed Design-for-Test Scheme

As mentioned earlier, the artificial broadcast scan constraints introduce several correlations among different signal lines in the circuit. Due to this, several faults

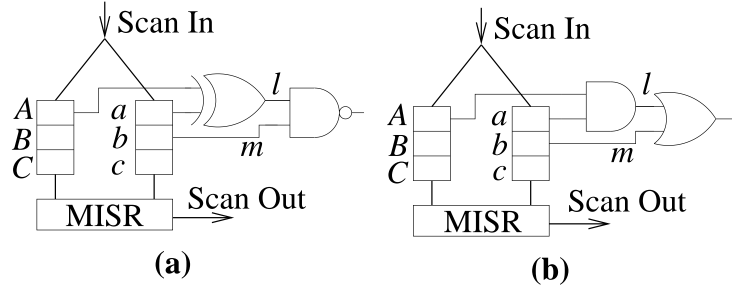


Figure 5.1: Artificially Untestable Faults

remain untestable in the parallel mode broadcast scan architecture. We call them *artificially untestable* faults and classify them into three groups:

1. ***Artificially Uncontrollable*** faults are those for which the excitation condition is not met due to the broadcast scan constraints. Consider the circuit shown in Figure 5.1 where the block labeled $CMPT$ is a compactor. The fault l sa 0 in Figure 5.1(a) is an artificially uncontrollable fault because it is not possible to set l=1 because this requires setting A and a to opposite values, which is impossible.

2. ***Artificially Unobservable*** faults are untestable faults for which the observability condition is not met. The fault m sa 1 in Figure 5.1(a) is an artificially unobservable fault because line m is always unobservable in the parallel broadcast scan mode since l can never be set to 1 to observe m.

3. ***Artificially Undrivable*** faults are those for which it is possible to independently excite and/or observe the fault effect at a PO/PPO but simultaneously exciting and observing the fault results in a conflict. The fault l sa 0 in Figure 5.1(b) is an artificially undrivable fault. We can set b=0 to observe l and also a=1, B=1 to excite the fault but the assignment b=0, a=1, B=1 results in a conflict.

Example 1. The untestable faults in Figure 5.1 can be made testable by inserting test points or by re-ordering the SFFs. For example, inserting an OP at l and a CP at m in Figure 5.1(b) can detect the stuck at faults at these signal lines,

respectively. Similarly, swapping SFFs A and B in the above designs converts all of the untestable faults into testable faults due to the following reason: SFFs A and a will always take the same Boolean value due to the broadcast scan architecture. Swapping SFFs A and B breaks this relationship, making all faults testable. The above example illustrates the effects of correlations in the parallel mode of the broadcast scan scheme and how test point insertion and scan re-ordering can independently fix the correlations.

For a given placement and routing, modifying the physical design to insert test points is very difficult because finding enough room in the design layout to insert this test point hardware is difficult. Note that the structural information of the netlist is required to identify the best places to insert test points and is available only after the logic synthesis step. This explains why we insert test points after the logic synthesis step and not after the physical synthesis step. Also, we use the timing information of the design to make sure that we insert test points only on those signal lines that have enough timing slack. If this condition is not met, then we will have to perform additional iterations between the timing verification and the logic synthesis step to fix all timing violations. A disadvantage of inserting the test points after the logic synthesis step is that one cannot use the scan chain ordering information for selecting the signal lines for inserting test points because scan chain ordering information is typically generated after the placement and routing step. In Section 5.3, we describe a heuristic approach that inserts test points just after the logic synthesis step such that the resulting design is more amenable for the parallel mode of the broadcast scan test generation without using the scan ordering information.

In the second step, we re-order the scan chain to further improve the testability of the design. The re-ordering step is done directly into the physical design by modifying the scan chain routing. We modify the scan chain routing between two SFFs only if these SFFs are located within distance r in the layout. Hence, this step does not incur much routing overhead and is practically feasible. This step is described in detail in Section 5.4. An added advantage of this step is that it eases

accommodating any post-layout modifications directly into the physical design, which will be explained now. In a typical VLSI design cycle, designers incorporate several modifications (e.g., last minute bug fixes) late in the design cycle into the physical design. From the DFT viewpoint, the proposed layout-aware re-ordering scheme is very useful in this scenario because it can directly modify the physical design without incurring much timing overhead while quickly converging to the routing closure. Thus, by executing the DFT scheme into two steps by using algorithms, which utilize the information available at that particular phase of the design cycle, we can seamlessly integrate the proposed technique with any existing VLSI design flow without affecting the overall turn-around time.

5.3 Our Test Point Insertion Procedure

As mentioned before, since we insert test points after the logic synthesis step and the exact scan chain order is not available at this stage of the design, the proposed TPI algorithm does not use the scan chain ordering information. But, typically, all of the SFFs, belonging to a particular module (block) in the design, belong to one internal scan chain. In this work, we use this information and assume that after the logic synthesis step we know which SFF belongs to which scan chain. Also, we assume that the given design has several *scan groups* where a scan group means a group of scan chains that derive the same test input data from a single external scan input pin as shown in Figure 2.7(b).

To understand the TPI algorithm, we define a new term called *correlated SFFs* using a parameter θ. Let CF_1 and CF_2 be sets comprising at least θ SFFs from two different scan chains SC_1 and SC_2, respectively. All of the SFFs in the sets CF_1 and CF_2 are correlated SFFs if the following condition is satisfied: For every SFF $\psi_1 \in CF_1$ there exists a SFF $\psi_2 \in CF_2$ and vice-versa, such that the two SFFs ψ_1 and ψ_2 "converge" at an internal signal line. Here, *converge* means there exists an internal line l whose fanin cone contains SFFs ψ_1 and ψ_2. We compute these correlated SFFs because when the ATPG tool specifies a large number of correlated SFFs during the parallel mode test generation step, it can

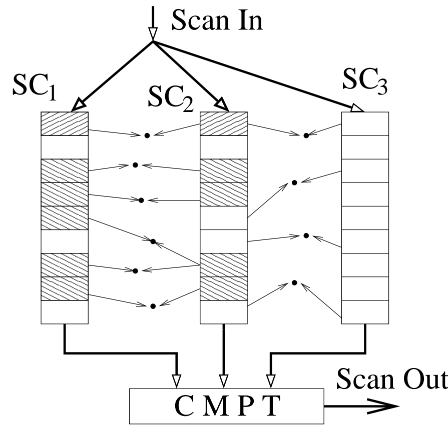


Figure 5.2: Example Illustrating Correlated SFFs

generate conflicts (because of the reconvergent paths connecting these SFFs) that, in turn, affect the performance of the ATPG tool. Correlated SFFs are illustrated in Figure 5.2 that shows three scan chains SC_1, SC_2, and SC_3 in the same scan group. We assume θ=5. The $\rightarrow$ in the figure represents a path and the $\cdot$ represents internal signal lines where the two paths converge. We see that six SFFs in SC_1 correlate with five SFFs in SC_2 (the shaded SFFs in Figure 5.2) and, hence, there are 11 correlated SFFs. Correlated SFFs can generate several conflicting assignments during the parallel mode broadcast scan test pattern generation and, hence, they increase the test volume and the ATPG run time for broadcast scan.

The key idea of the proposed TPI algorithm is to insert the TPs into the design such that the parallel mode broadcast scan TPG will now specify fewer of these correlated SFFs. Hence, our TPI scheme minimizes the number of times a correlated SFF is specified during the parallel mode broadcast scan ATPG. This will reduce the number of conflicts occurring during the parallel mode of the broadcast scan TPG. This is because all of the reconvergences among different correlated SFFs are "broken" by the inserted test points. Hence, our TPI algorithm is significantly different from all other prior known TPI schemes [60, 61, 72], as it specifically improves the testability of designs employing the broadcast scan architecture.

5.3.1 The Test Point Hardware

We will now describe how to implement a test point and the overhead associated with it. In Chapter 4 [61], we presented the *complete test point* (CTP) (see Figure 4.2(a)) for structured ASICs that can observe a signal line l_1 while also controlling another line l_2 to Boolean 0/1. Figure 4.2(b) shows a CTP that can observe and control the same signal line l. Note that two scan flops and a MUX are required to implement a CTP. In this chapter, we present an optimized implementation of a CTP that controls and observes a signal line l. It is shown in Figure 5.3 and is implemented by modifying the SFF design and adding an OR-AND gate at the output. The input labeled TM represents the *test mode* signal. During the test mode (TM=1), we can load the desired Boolean value 0/1 into the CTP through the scan chain. In this mode, the CTP observes the input D while also controlling the output Q to Boolean 0/1. The highlighted lines in Figure 5.3(a) represent the data path for the test mode of operation. During the functional mode (TM=0), the test point behaves as a regular buffer. The highlighted lines in Figure 5.3(b) represent the data path for the functional mode of operation. The delay added by this test point hardware during the functional mode is the delay of an OR-AND gate. Thus, this test point can be used only when the available slack in that path is greater than the OR-AND gate delay. When the available slack is small, then one can insert an OP [60] to improve the testability. Since SFFs implementing the test points are also inserted into the scan chains, inserting them increases the pattern length by one.

5.3.2 Identifying Correlated Flip-flops

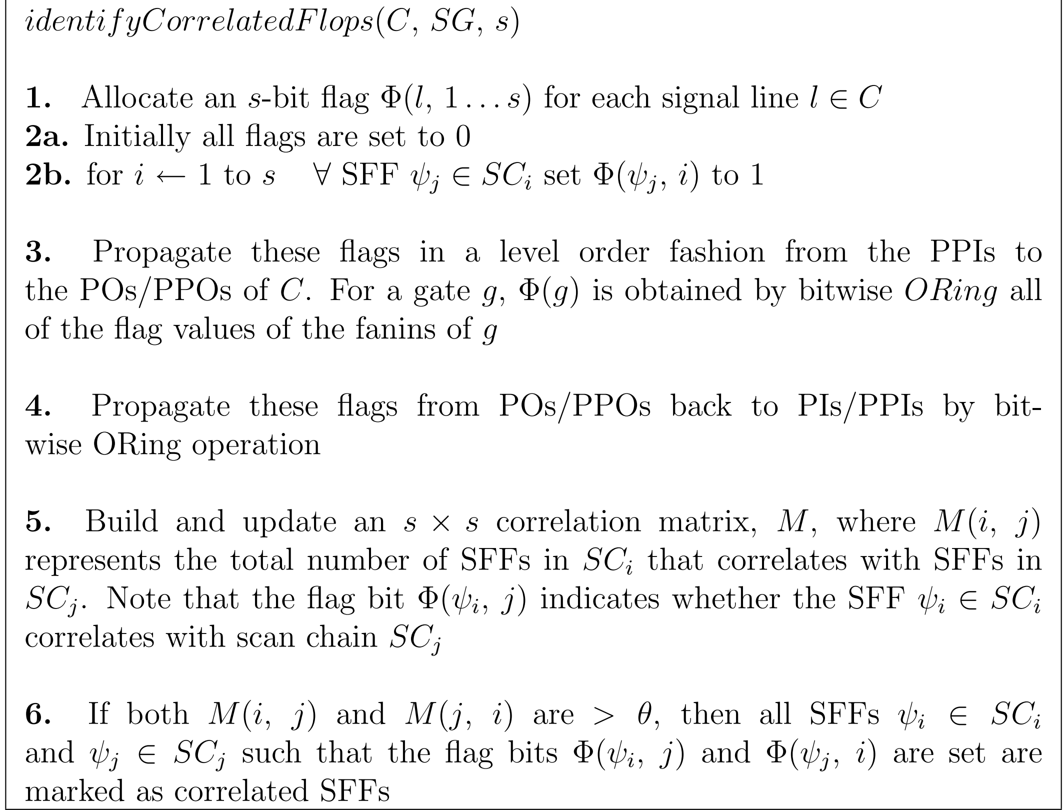
The first step of our TPI algorithm is to identify all of the correlated SFFs in the design. The module $identifyCorrelatedFlops()$ shown in Figure 5.4 identifies all of the correlated SFFs in the given scan group SG. Let C be the circuit, s be the number of internal scan chains in SG, and SC_1, SC_2, …, SC_s denote the s internal scan chains. We assign an s-bit flag denoted by $\Phi(l)$ for each signal

Figure 5.3: An Optimized Complete Test Point Highlighting the Operation for: a) Test Mode and b) Functional Mode

line l where the ith bit of $\Phi(l)$ tells whether the fanin cone of l contains a SFF in scan chain SC_i. In Steps 1 and 2, we initialize $\Phi(l)$ for all the signal lines $l \in C$. In Steps 3 and 4, we propagate the flag values from the PIs/PPIs to the POs/PPOs. By propagating the flag bits back to the SFFs, we identify SFF correlation information. In Steps 5 and 6, we identify all of the correlated SFFs in SG.

After identifying all of the correlated SFFs in the design, we compute the gain functions for all of the signal lines to choose the best places to insert the TPs. To describe the gain function, let $N_{OF}(l)$ be the number of correlated SFFs that must be specified to observe a fault effect at line l, $N^1_{CF}(l)$ ($N^0_{CF}(l)$) be the number of correlated SFFs that must be specified to control l to 1 (0), $N_f(l)$ be the total number of independent faults that need to pass through l to be detected, $F^1_C(l)$ ($F^0_C(l)$) represent the number of independent faults that are excited due to inserting a CP at l, and $F^1_O(l)$ ($F^0_O(l)$) represent the number of independent faults that pass through the AND/NAND/XOR/XNOR (OR/NOR/XOR/XNOR) gate g where l is an input of g. The gain function G_{CTP} for the complete test point is expressed as this theorem:

Theorem 5.3.1 *The lower bound on the total number of times a correlated SFF*



$identifyCorrelatedFlops(C,\ SG,\ s)$

1. Allocate an s-bit flag $\Phi(l,\ 1 \ldots s)$ for each signal line $l \in C$
2a. Initially all flags are set to 0
2b. for $i \leftarrow 1$ to s $\quad \forall$ SFF $\psi_j \in SC_i$ set $\Phi(\psi_j,\ i)$ to 1

3. Propagate these flags in a level order fashion from the PPIs to the POs/PPOs of C. For a gate g, $\Phi(g)$ is obtained by bitwise *ORing* all of the flag values of the fanins of g

4. Propagate these flags from POs/PPOs back to PIs/PPIs by bitwise ORing operation

5. Build and update an $s \times s$ correlation matrix, M, where $M(i,\ j)$ represents the total number of SFFs in SC_i that correlates with SFFs in SC_j. Note that the flag bit $\Phi(\psi_i,\ j)$ indicates whether the SFF $\psi_i \in SC_i$ correlates with scan chain SC_j

6. If both $M(i,\ j)$ and $M(j,\ i)$ are $> \theta$, then all SFFs $\psi_i \in SC_i$ and $\psi_j \in SC_j$ such that the flag bits $\Phi(\psi_i,\ j)$ and $\Phi(\psi_j,\ i)$ are set are marked as correlated SFFs

Figure 5.4: Algorithm to Identify Correlated Flip-flops

need not be specified due to inserting a CTP at signal line l is:

$$G_{CTP}(l) = G^1_{CP}(l) + G^0_{CP}(l) + G_{OP}(l) \tag{5.1}$$

where

$$\begin{aligned} G^v_{CP}(l) &= (F^v_C(l) + F^v_O(l)) \times N^v_{CF}(l), v = 0/1 \\ G_{OP}(l) &= N_f(l) \times N_{OF}(l) \end{aligned}$$

Proof. The ATPG tool has to generate at least $F^1_C\ (l)$ vectors to excite the faults in the fanout of l and at least $F^1_O\ (l)$ vectors to observe the faults in the fanin cone of the off-path signals of l (because all of the $F^1_C\ (l) + F^1_O\ (l)$ faults are independent). When the CTP at l is used as a 1 (0)-CP, the correlated SFF bits in these vectors need not be specified by the ATPG. So, the gain due to this is $G^0_{CP}(l)\ +\ G^1_{CP}(l)$. Also, at least $N_{OF}(l)$ vectors are needed to detect the faults in the fanin cone of l because $N_{OF}(l)$ fault effects pass through l in the absence

of this test point. When the CTP observes l, the correlated SFF bits in these vectors will not be specified by the ATPG. So, the gain due to this is $G_{OP}(l)$. Also, by definition, there are no overlapping faults among $G^1_{CP}(l)$, $G^0_{CP}(l)$, and $G_{OP}(l)$. Hence, the sum $G^1_{CP}(l) + G^0_{CP}(l) + G_{OP}(l)$ is a lower bound on the total number of times correlated SFFs need not be specified due to inserting a CTP at signal line l. □

We use a two pass algorithm to compute the gain function represented by Eqn. 5.1. In the first pass, we compute $N_f(l)$, $N_{OF}(l)$, $N^v_{CF}(l)$, $F^v_O(l)$, and $F^v_C(l)$ for all circuit signal lines. In the next pass, we calculate the gain functions for all signal lines using Eqn. 5.1. Since this is compute intensive, we approximate wherever necessary. We also present several optimizations to significantly reduce run time and memory usage of our TPI tool.

5.3.3 Controllability and Observability Cost

The controllability cost $N^v_{CF}(l)$ $(v = 0/1)$ for a signal line l is the total number of the correlated SFF bits that need to be specified to set l to logic v. The observability cost $N_{OF}(l)$ is the total number of the correlated SFF bits that need to be specified to propagate the fault effect at l to a PO/PPO. If the design has reconvergent fanouts, then calculating accurate controllability and observability for signal l is difficult. As a simple estimation, we can use a SCOAP-like measure [21] where the observability cost of the line l_i is obtained by simple additions of controllability costs of all its other inputs and the observability cost of l_O where signal line l_i is the ith input of gate g and l_O is g's output. However, in Chapter 3 [60], we showed that the SCOAP-like measure is fairly inaccurate due to the presence of reconvergent signals. Hence, to improve the accuracy of the controllability/observability calculations, we use bit arrays to represent controllability and observability of signal lines.

We use three bit arrays, denoted $CC_0(l)$, $CC_1(l)$, and $O(l)$, for each signal line l to store the 1-, the 0-controllabilities, and the observabilities, respectively.

The jth entry of the controllability arrays of line l tells whether the jth PI/PPI in that circuit cone controls l or not. Similarly, the jth entry of the observability array, $O(l)$, tells whether the jth PI/PPI needs to be specified or not to observe l at the PO/PPO of the circuit cone.

A disadvantage of using a bit map is that it requires a large memory space to store intermediate bit operation results. To reduce memory, we partition the design into a large number of circuit cones C_i, $i = 1, 2, \ldots$. This is similar to the algorithm described in Section 4.3.1. A particular cone C_i comprises one PO/PPO, denoted p_i, and all signal lines located in the *transitive fanin* of p_i. Memory allocated for computing the testability measures ($N^v_{CF}(l)$ and $N_{OF}(l)$) of all signal lines in C_i is freed before considering the next circuit cone. This approach drastically reduces the peak run time memory consumed by our TPI tool.

A two-pass algorithm computes testability values of all lines l in the cone C_i. All of the bit arrays (CC_0, CC_1, and O) of all signal lines l in C_i are initialized to 0's. In the first pass, we traverse from PIs/PPIs in level order toward the PO/PPO of cone C_i, computing both $CC_1(l)$ and $CC_0(l)$ for each signal line l. Since a 1 in the bit array $CC_1(l)$ ($CC_0(l)$) means that a particular PI/PPI must be specified to set l to 1 (0), counting the number of 1's in $CC_1(l)$ ($CC_0(l)$) that correspond to the correlated SFF gives the value of $N^1_{CF}(l)$ ($N^0_{CF}(l)$). In the second pass, we traverse from the PO/PPO of C_i toward PIs/PPIs. In this pass, we compute the bit array $O(l)$ for observing the fault effect at each signal line l at the PO/PPO of this cone. $N_{OF}(l)$ for signal line l is the number of 1's in the bit array that corresponds to the correlated SFF.

Next, we describe procedures to compute N_f, F^v_C and F^v_O for all circuit signal lines. Since this computation is not memory intensive, the procedure does not use partitioning.

5.3.4 Computing $N_f\ (l)$, $F_C^v\ (l)$, and $F_O^v\ (l)$

As mentioned earlier, $N_f(l)$ is defined as the total number of independent faults that need to pass through l to be detected, and $F_C^v(l)$ is the total number of independent faults that are excited when a v-CP, where $v = 0/1$ at signal line l, is switched on. Testability measure $F_O^v(l)$ is the total number of independent faults that pass through all off-path signals of l by setting l to logic value v. Computing $N_f(l)$, $F_C^v(l)$, and $F_O^v(l)$ involves counting the number of independent faults but identifying independent faults in a circuit is very computation intensive. So, we approximate the total number of independent faults by the total number of collapsed faults and this is described next.

5.3.4.1 Computing $N_f(l)$

After obtaining the collapsed fault list, we then compute $N_f(l)$, $F_C^v(l)$, and $F_O(l)$ for all signal lines l. All $N_f(l)$ values are initially zeroed. The algorithm propagates each fault f guided by the observability cost, N_i, toward a PO/PPO. When a fault effect propagates through line l, $N_f(l)$ is incremented. When all faults are propagated, we obtain N_f values for all circuit signal lines.

5.3.4.2 Computing $F_C^v\ (l)$ and $F_O^v\ (l)$

We compute $F_C^v(l)$ and $F_O^v(l)$ from the $N_f(l)$ values. We first initialize all $F_C^v(l)$ values in the circuit to 0. Then, we assign a logic value v to l and imply the Boolean value to as many signal lines as possible that are located in the transitive fanout of l. When a Boolean value v_f propagates to line l_f, such that the fault stuck-at $\overline{v_f}$ at l_f belongs to the final collapsed fault list FS_m, then $F_C^v(l)$ is incremented. When this operation is performed for all circuit signal lines, we obtain $F_C^v(l)$ values for the entire circuit.

To compute $F_O^v(l)$, the algorithm identifies the list of all gates G_f such that: (a) l is a fanin of G_f and (b) v is a non-controlling value of G_f. Then, it obtains all fanin signals F_{in}, of all gates G_f excluding l, so $\sum_{f_{in} \in F_{in}} N_f\ (f_{in})$ represents $F_O^v(l)$.

5.3.5 Updating Testability Measures

Once a CTP is inserted into line l, the gain values for all lines in the neighborhood of l are updated to reflect the extra CTP that has been added. We first identify all lines L_{in} in the fanin cone of l such that a fault effect from $l_{in} \in L_{in}$ passes through l before reaching a PO/PPO before inserting a CTP. We then subtract the value of $N_f(l)$ from $N_f(l_{in})$, because the fault effect at l_{in} will now be observed at l itself. We then obtain all signal lines L_{out} in the fanout cone of l such that fault effects from l pass through $l_{out} \in L_{out}$ before reaching a PO/PPO before inserting a CTP. Since $N_f(l)$ faults will now be observed at l, we subtract the value of $N_f(l)$ from $N_f(l_{out})$ to update the N_f values.

Also, inserting a CTP at l decreases the controllability of l and all signals in the transitive fanout of l. To update gain values of other lines, we obtain the set of all lines L_{DI} that are directly implied by setting l to logic v. We then obtain the list of all output signals $l_{DI}^{out} \in L_{DI}^{out}$ such that: (a) $l_{DI}^{out} \notin L_{DI}$ and (b) l_{DI}^{out} is a fanout signal of some $l_{DI} \in L_{DI}$. Thus, the entire set of signals $l_{all} \in L_{DI} \cup L_{DI}^{out}$ represents signal lines that directly benefit from inserting a CTP at l. Then, we decrement the value $N_{CF}^{v}(l)$ from the controllability values of all signal lines l_{all} $\in L_{DI} \cup L_{DI}^{out}$. This completes the procedures for updating testability measures after inserting a particular TP.

5.3.6 The Overall TPI Algorithm

Figure 5.5 shows the overall algorithm to insert N TPs into the design C. The procedure $identifyAllCorrelatedFlops()$ uses the algorithm described in Section 5.3.2 to identify all correlated SFFs in C. The procedure *computeGainFunction* () computes the gain function for all signal lines in the circuit. The procedure *createCandidateTestPoints* () creates a list comprising L signals with highest gain values. This list represents the search space of our TPI tool. Since searching only L signals for TPI purposes is less expensive than evaluating all circuit signals, this step greatly reduces the total run time of our TPI tool. Procedure *obtainNextTP*

```
insertNTestPoints(N, C)
1. identifyAllCorrelatedFlops(C)
2. computeGainFunction(C)
3. createCandidateList(L, &candidateList)
4. for i ← 1 to N
5.     obtainNextTP(candidateList, &tp)
6.     storeThisTP (&TPList, tp)
7.     updateTestability(tp, C)
8. insertTPIntoDesign(TPList, C)
```

Figure 5.5: The Overall TPI Algorithm

() gets a signal line with the highest gain function from the candidate list. Procedure *storeThisTP* () stores the chosen signal line in a list and is later used by *insertTPIntoDesign* (), which writes the design along with all inserted TPs back onto the disk. Procedure *updateTestability* () updates the values of testability measures of all signal lines in the candidate list due to inserting the chosen TP (see Section 5.3.5).

5.4 Scan Chain Re-ordering Technique

Earlier, we showed that a particular scan chain order can prevent us from applying certain input vectors in the parallel mode leaving several faults untestable. By re-ordering the scan chain (see Example 1) these faults can be detected in the parallel mode. In the scan chain re-ordering step, we obtain the existing scan chain order generated by the physical synthesis tool and then re-order the chain sequence by swapping SFF pairs that are located close to each other. This swap operation is showed in Figure 5.6 where the original scan chain order $a{-}b{-}c{-}d{-}e$ is changed to $a-d-c-b-e$. We assume that the SFF pair (b, d) is located close to each other in the layout. The objective of the re-ordering step is that a large number of artificially untestable faults becomes testable during the parallel mode of the broadcast scan TPG. In this section, we will describe a heuristic approach to achieve this objective while using minimal swap operations.

Hamzaoglu and Patel's work [25] uses a compatibility analysis on the entire set

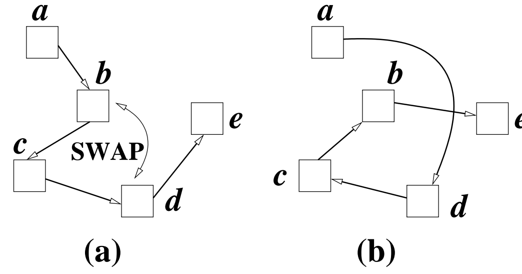


Figure 5.6: Swap Operation: a) Old and b) New Order

of test cubes for ordering the scan chain. For large designs, generating the entire test cube can be time consuming. Hence, we use a very inexpensive algorithm, similar to the correlation analysis described in Section 5.3, to re-order the scan chain. Our algorithm uses the existing scan ordering information to quantify the gain $G_{RO}(\psi_1, \psi_2)$ (described in Section 5.4.3) achieved by swapping the SFF pair (ψ_1, ψ_2). But first, we will define a few terms that will be used in the rest of this section.

1. ***Scan Slice***, denoted by σ, is a set of SFFs in the same scan group that has to take the same bits from the scan input (see Figure 2.7(b)).

2. ***Conflicting SFF Pair.*** The SFF pair (ψ_1, ψ_2) is said to be conflicting if they have to take different (conflicting) Boolean values to detect one or more faults in the design. The variable $\kappa(\psi_1, \psi_2)$ quantifies the number of times the SFF pair (ψ_1, ψ_2) conflicts to detect all faults in the circuit.

3. ***Slice Correlation***, $S(\psi, \sigma)$, of a SFF ψ with respect to the scan slice σ represents the total number of times ψ conflicts with each of the SFF $\psi_\sigma \in \sigma$. In Figure 5.1(b), $S(b, \sigma_1)$ is 1.

4. ***Slice Conflict*** represents the total number of times one or more pairs of SFFs in σ conflict to detect all of the faults in the design.

   The key idea of our scan re-ordering algorithm is that a SFF pair ($\psi_1 \in \sigma_1$, $\psi_2 \in \sigma_2$) will be swapped only if: a) ψ_1 lies in the neighborhood of ψ_2, and b) swapping reduces the slice conflicts of σ_1 or σ_2 or both. Hence, the

scan re-ordering algorithm uses the existing scan ordering to convert many artificially untestable faults into testable faults.

5.4.1 Build Neighborhood Information

A SFF ψ_1 is a neighbor of ψ_2 only if the distance between (ψ_1, ψ_2) is $< r$ (r is a user-defined parameter). Since we swap only the neighboring SFF pairs, the first step of the scan re-ordering algorithm involves building a neighborhood list, denoted by $\psi.NList$, for each SFF ψ in the design. For the purpose of this research, we use the SFF ordering in the netlist to decide whether SFFs ψ_1 and ψ_2 are separated by a distance $< r$. Building the neighborhood list for all SFFs in the design will require us to compute the distance between every SFF pair using the layout file, which will cost $O(N_{SFF}^2)$ where N_{SFF} is the total number of SFFs in the design. This is prohibitively expensive especially when N_{SFF} is very large. So, we propose an approximate and inexpensive algorithm for this.

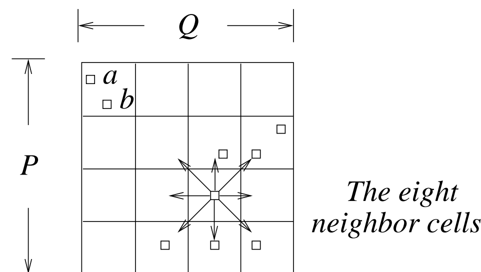


Figure 5.7: Building Neighborhood Information

First, map the layout onto an imaginary two dimensional grid of size $P \times Q$ as shown in Figure 5.7 where the choice of P, Q determines the accuracy of the algorithm. Only the SFF placement is shown in the figure and P=4, Q=4 for the purpose of illustration. Build a 2D matrix data structure G of size $P \times Q$ such that $G(i,\ j)$ stores the list of SFFs in the $(i,\ j)$ cell. For the layout given in Figure 5.7, $G(1,\ 1)$ will store SFFs a and b whereas $G(1,\ 2)$ will be $NULL$. Now, to obtain the neighbor of any SFF located inside any grid $G(x,\ y)$, we can include all SFFs located at the eight neighboring cells of $G(x,\ y)$.

5.4.2 Slice Correlation

As mentioned before, we use the slice correlation information to compute the gain values (see Section 5.4.3). We will now describe how the algorithm described in Section 5.3.3 can be reused for the slice correlation computation. Recall that $CC0(l)$ ($CC1(l)$) represents the PIs/PPIs that need to be specified to obtain Boolean 0 (1) at line l and $CO(l)$ represents the bit map to observe l. Then, the bit array $S0(l)$ = OR($CC1(l)$, $CO(l)$) represents the set of PIs/PPIs that need to be specified to detect the l sa0 fault because we need to control signal line l to 1 and also observe l at some PO/PPO. Similarly, $S1(l)$ = OR($CC0(l)$, $CO(l)$) represents the set of PIs/PPIs that need to be specified to detect l sa1. Here, OR() represents the bitwise logical OR operation. We compute $S0(l)$ and $S1(l)$ for all of the signal lines in the design. Using these bitmaps, we directly compute both SFF conflict and the slice correlation for the entire design.

5.4.3 Computing the Gain Value

We compute gain values for SFF pairs (ψ_1, ψ_2) such that ψ_1 is located in the neighborhood of ψ_2 (distance $< r$). The gain value reflects the overall reduction in the slice conflicts in the entire design due to the swapping of that SFF pair and is given using the theorem below:

Theorem 5.4.1 *Let $\psi_1 \in \sigma_1$ be a SFF located in the neighborhood of $\psi_2 \in \sigma_2$. The gain due to swapping the SFF pair (ψ_1, ψ_2), denoted by $G_{RO}(\psi_1, \psi_2)$, is:*

$$\begin{aligned} G_{RO}(\psi_1, \psi_2) = & \ S(\psi_1, \sigma_1) + S(\psi_2, \sigma_2) \\ & - S(\psi_1, \sigma_2) - S(\psi_2, \sigma_1) \end{aligned}$$

Proof. The total number of slice conflicts that are reduced due to removing ψ_1 from σ_1 and ψ_2 from σ_2 is $S(\psi_1, \sigma_1) + S(\psi_2, \sigma_2)$. However, moving the SFF ψ_1 into σ_2 and ψ_2 into σ_1 introduces slice conflicts, which are $S(\psi_1, \sigma_2) + S(\psi_2, \sigma_1)$. Hence, the total gain is $G_{RO}(\psi_1, \psi_2) = S(\psi_1, \sigma_1) + S(\psi_2, \sigma_2) - S(\psi_1, \sigma_2) - S(\psi_2, \sigma_1)$. This completes the proof. □

5.4.4 Updating Gain Values

Each time when a SFF pair (ψ_1, ψ_2) is swapped, the slice correlation values corresponding to the SFFs in the neighborhood of ψ_1 and ψ_2 change. Since re-computing the gain values for all of the candidate pairs after each swap operation can be very expensive, we only update the gain values of the neighbors of ψ_1 and ψ_2 using the theorem given below.

Theorem 5.4.2 *Let $\psi_1 \in \sigma_1$ be a SFF located in the neighborhood of $\psi_2 \in \sigma_2$. The change in the slice correlation value, $\Delta S(\psi, \sigma_1)$, of the SFF ψ located in the neighborhood of ψ_1 due to the swapping of the SFF pair (ψ_1, ψ_2) is:*

$$\Delta S(\psi, \sigma_1) = \kappa(\psi, \psi_1) - \kappa(\psi, \psi_2) \tag{5.2}$$

Proof. The total number of SFF conflicts that are reduced due to removing ψ_1 from σ_1 is $\kappa(\psi, \psi_1)$. However, moving the SFF ψ_1 into σ_2 introduces SFF conflicts, which are $\kappa(\psi, \psi_2)$. Hence, the total reduction in slice correlation is $\Delta S(\psi, \sigma_1) = \kappa(\psi, \psi_1) - \kappa(\psi, \psi_2)$. Similarly, we can say that $\Delta S(\psi, \sigma_2) = \kappa(\psi, \psi_2) - \kappa(\psi, \psi_1)$. This completes the proof. □

The algorithm for updating the gain values for the entire design due to the swapping of SFF pair (ψ_1, ψ_2) is given in Figure 5.8. The first step prevents the algorithm from moving a particular SFF multiple times. In the second step, we collect all of the neighbors of ψ_1 and ψ_2 and use Theorem 5.4.2 in Steps 3a-c for updating their slice correlation values.

***updateGainValues* (ψ_1, ψ_2)**
1. Remove SFF pairs (ψ_i, ψ) from the candidate list L where ψ_i=ψ_1 or ψ_2.
2. Let $\psi.NList = \psi_1.NList \cup \psi_2.NList - (\psi_1, \psi_2)$
3a. for each $\psi \in \psi.NList$
3b. $S(\psi, \sigma_1) = S(\psi, \sigma_1) - \kappa(\psi, \psi_1) + \kappa(\psi, \psi_2)$
3c. $S(\psi, \sigma_2) = S(\psi, \sigma_2) - \kappa(\psi, \psi_2) + \kappa(\psi, \psi_1)$

Figure 5.8: The Algorithm for Updating the Gain Values

5.4.5 The Overall Algorithm

The overall algorithm for layout-aware scan chain re-ordering is given in Figure 5.9. The first step builds the neighborhood information and also constructs the candidate SFF pair list. The second step computes both the SFF conflict values and the slice correlation values. The SFF correlation values are later used in Step 7 for updating. In Step 3, gain values are computed using the slice correlation values. The scan re-ordering is an iterative process and is implemented by Steps 4-8. The method $obtainNextBestPair()$ will obtain the SFF pair in the candidate list that has the highest gain value. This pair is then swapped in Step 6 and in Step 7, we update the gain values as described in Section 5.4.4. This process stops when the maximum gain values of the candidate SFF pairs fall below a certain threshold value.

reorderScanChain ()
1. Build neighborhood information (see Section 5.4.1).
2. Compute slice correlations (see Section 5.4.2).
3. Compute the gain values (see Section 5.4.3).
4. *do* {
5. $obtainNextBestPair$ (ψ_1, ψ_2);
6. $SWAP$ (ψ_1, ψ_2);
7. $updateGainValues$ (ψ_1, ψ_2);
8. } *while* ($G_{RO}(\psi_1, \psi_2) >$ THRESHOLD);

Figure 5.9: The Overall Re-ordering Algorithm

5.5 Complexity Analysis

The TPI scheme identifies correlated SFFs in the design using the algorithm shown in Figure 5.4. The time and memory complexity of this algorithm is $O(n)$ where n is the total number of signals reachable from the SFFs in a scan group SG. This is because the cost of Steps 1 and 2 is $O(N_{SG})$ where N_{SG} represents the total number of SFFs in SG, which is $< n$. Steps 3 and 4 visit each reachable signal line only once and hence, the cost is $O(n)$. The cost of Steps 5 and 6 is

$O(s^2)$, but note that $s << n$. So the total complexity is $O(n + s^2 + N_{SG}) \approx O(n)$. Then, gain functions for all of the signal lines in the circuit are computed using the Equation 5.1. The algorithm for computing these values is similar to the algorithms presented in Chapter 4. Hence, the complexity of this algorithm is also $O(n)$ (see Section 4.4). Hence, the cost of the overall TPI algorithm presented in Section 5.5 is $O(n)$.

The scan chain re-ordering involves building the neighborhood information for all of the SFFs in the circuit (see Section 5.4.1). The cost for initializing the grid is $P \times Q$ (see Figure 5.7) and the cost to visit all of the eight neighboring SFF cells for all of the SFFs in the design is $8 \times N_{SFF}$. Hence, the total cost of this algorithm is $8 \times N_{SFF} + P \times Q \approx O(N_{SFF})$ if we choose P, Q such that the product $P \times Q$ is linearly proportional to N_{SFF}. The cost for computing the gain function G_{RO} (see Equation 5.2) for all of the signal lines in the circuit is also $O(n)$. This is because it utilizes the algorithms (see Section 5.4.3) given in Section 4.3 that are already shown to be linear to the number of signal lines in the circuit. Hence, the overall cost for the entire DFT scheme proposed in this chapter is $O(n)$.

5.6 Summary

In this chapter, we presented a new two step DFT to improve the performance of an existing broadcast scan-based compressor. We showed that broadcast scan compressors introduce several undesired correlations in the design. The proposed DFT scheme can break these correlations and help reduce the test volume. The first step of the proposed DFT scheme is a TPI process. This is described in detail in Section 5.3. We proposed novel gain functions and efficient algorithms to compute the gain functions. The overall algorithm for TPI is given in Section 5.3.6.

The second step of the proposed DFT scheme is the layout-aware scan chain re-ordering process. This is described in detail in Section 5.4. During the scan

chain re-ordering step, we restrict the maximum distance by which any SFF is moved in the physical design. Hence, our technique incurs very little routing overhead. The overall algorithm for scan chain re-ordering is given in Section 5.4.5. In Section 5.5, we also showed that the time complexity of the proposed DFT algorithms is $O(n)$ where n is the number of signal lines in the design, making them scalable for large industrial designs.

Chapter 6
Results

We conducted several experiments to validate the efficacy and scalability of the proposed schemes. This chapter presents the results of all of our experiments. The entire TPI tool described in Chapters 3, 4, and 5 was implemented in the C programming language. In all of our experiments, we used NEC's in-house ATPG [15] tool called $TRAN$ for test pattern generation and test compaction. $TRAN$ uses an *implication graph* (IG) to generate the test data for the given design. It employs both static and dynamic compaction for test data compaction. Section 6.1.1 presents the results for the algorithms presented in Chapter 3, Section 6.1.2 describes the results for the algorithms in Chapter 4, and the results for enhancing the broadcast scan compressor (see Chapter 5) are presented in Section 6.2.

6.1 Structured ASICs

In Chapters 3 and 4, we introduced a zero-cost TPI for structured ASICs. We will now describe the results of our experiments with very large structured ASIC designs.

6.1.1 Inserting Observation Points

In Chapter 3, we described a TPI scheme to insert observation points into a structured ASIC design. Four ISSP SA designs were used to conduct our experiments. Details of these designs are given in Table 6.1. Here, the column labeled *Target Fault Efficiency* represents the fault efficiency that can be achieved in reasonable

Table 6.1: Design Characteristics

Ckt.	*# Signals*	*# Flip-flops*	*Target Fault Efficiency*
*ckt*1	403.6 $\times 10^3$	96.70 $\times 10^3$	98.5%
*ckt*2	741.4 $\times 10^3$	144.90 $\times 10^3$	99.0%
*ckt*3	476.4 $\times 10^3$	145.60 $\times 10^3$	100.0%
*ckt*4	5.0 $\times 10^6$	1.25 $\times 10^6$	98.7%

CPU time for a particular design and was set as the desired fault efficiency in all of our experiments. Fault efficiency is defined as the percentage of the total faults that are detected over the number of testable faults proven by the ATPG tool. Several experiments were conducted to validate the feasibility of the proposed

Table 6.2: Experimental Results

Ckt.	*TPI*		*Test Generation Time*			*Test Volume*		
	Time (*min*)	*Mem.* (*MB*)	*Before* (*min*)	*After* (*min*)	*Reduc.* (%)	*Before* #	*After* #	*Reduc.* (%)
*ckt*1	11	243	1843.2	1051.6	42.9	1717	1318	23.2
*ckt*2	8	106	263.0	186.0	29.3	562	416	25.9
*ckt*3	1	26	87.8	63.9	27.2	361	349	3.3
*ckt*4	10	143	2295.6	1516.1	34.0	10255	9683	5.6

TPI technique. In the first experiment, we inserted a fixed number of test points to measure the test volume and test generation time reduction. We inserted 1000 TPs into *ckt*1, *ckt*2, and *ckt*4 and only 500 TPs into *ckt*3 because *ckt*3 is an easy-to-test design. Results are shown in Table 6.2. The two columns under the heading TPI show the total run time (in minutes) and memory usage (in MB) of the TPI tool. This includes the total run time and memory required to compute the gain function and for inserting test points into the design. The columns under the heading $Test\ Time$ give the total test generation time before and after the test point insertion and the time reduction achieved (in %). The columns under the heading $Test\ Volume$ give the total number of test patterns generated before and after the test point insertion and the pattern reduction achieved (in %). We reduced test generation time by up to 42.9% and test volume by up to 25.9%. For the design *ckt*3, we did not get a high test volume reduction. According to our extensive experiments, inserting test points for easy-to-test circuits does not

reduce test data volume significantly.

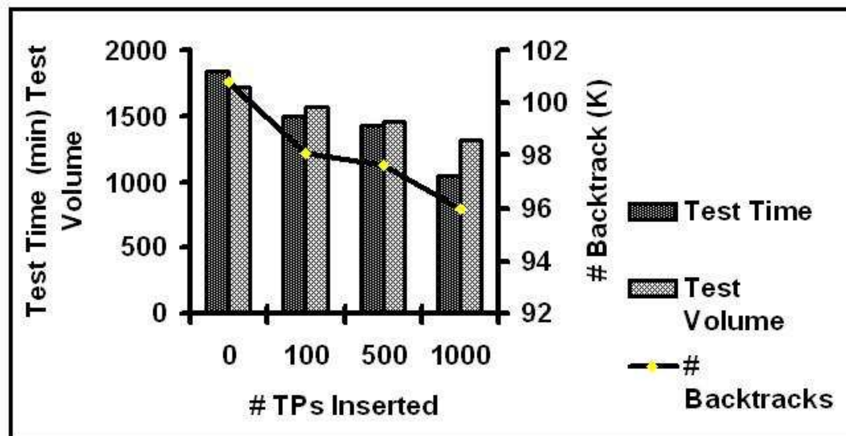


Figure 6.1: Results for Inserting 100, 500, and 1000 TPs

In order to understand the relationship between test generation and the number of TPs inserted, we made three different versions of *ckt*1 that have 100, 500, and 1000 test points, respectively. Figure 6.1 shows how test volume, test generation time, and backtrack count vary by increasing the number of inserted TPs. Note that even though the test generation time was reduced by 42.9%, the backtrack count did not change proportionately. This is because, unlike in prior approaches, our test points do not target hard-to-detect faults. Instead, test points computed by our technique facilitate generating a large number of don't cares in the resulting test set, due to which both the test volume and test generation time decrease drastically, which has little impact on the backtrack count.

In another experiment (see Figure 6.2), we studied the relationship between compaction time and test generation time (time for detecting primary and secondary faults as described in Section 3.2). Note that the ATPG run time for circuits with 100 and 500 TPs reduced drastically, while the compaction time reduced little. Inserting 1000 TPs, however, reduced the compaction time significantly because a large number of don't cares were generated, so ATPG generated significantly fewer vectors and the compactor was able to compact these vectors in a very short time. To compare our approach with the SCOAP-like measures (see

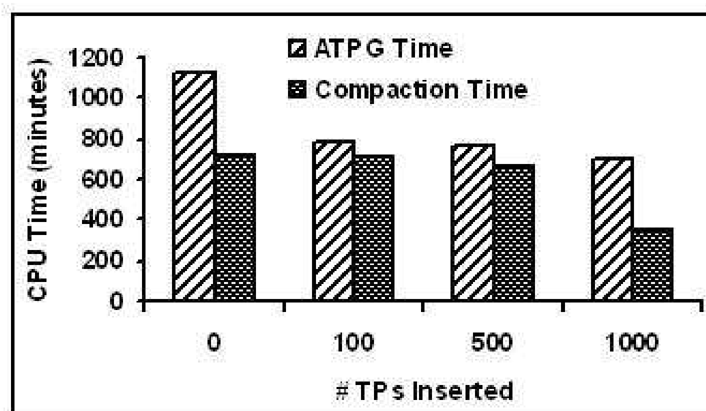


Figure 6.2: ATPG vs. Compaction Time

Section 3.2.2), we made two versions of $ckt1$ each with 1000 TPs. TP's in the first design were chosen using the proposed gain function and in the other design, the SCOAP-like measures (see Equation 3.2) were used to choose the TPs. Results in Figure 6.3 indicate that the proposed gain function outperforms SCOAP-like measures that are very inaccurate.

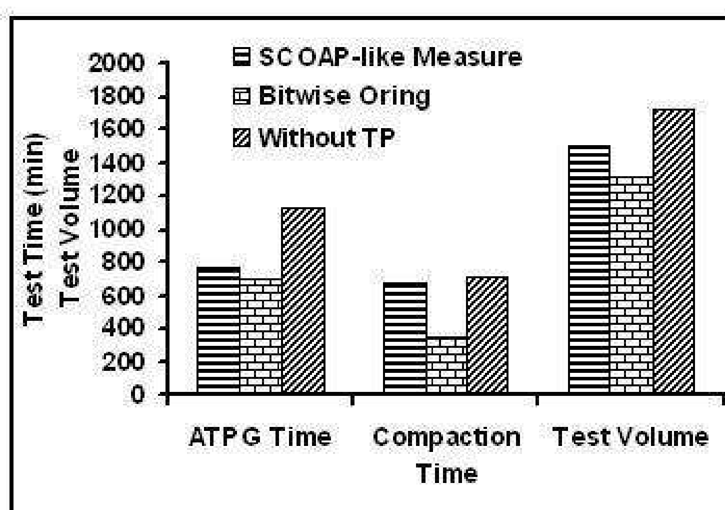


Figure 6.3: Proposed Technique vs. SCOAP-like Measure

6.1.2 Inserting Control Points

In Chapter 4, we presented different types of CPs for improving the testability of SA designs. Extensive experiments were conducted with ISCAS '89, ITC '99, and

industrial benchmark circuits (*ckt*1, *ckt*2) to validate the efficacy of the proposed algorithms for inserting CPs. Table 6.3 shows the results that were obtained by inserting a fixed number of complete test points into the design.

Table 6.3: Test Vector Length Reduction and CPU Time with TPI with Near 100% FE

Circuit	# *Lines*	#	*Vector Length*		*CPU Time (seconds)*	
	($\times 10^3$)	*TPs*	*Original*	*Reduced*	*Original*	*Reduced*
*s*9234	20.9	100	177	86	8.7	5.6
*b*20*s*	67.5	100	786	343	81.2	48.1
*b*21*s*	40.9	100	704	386	75.7	58.8
*b*22*s*	78.3	120	589	308	122.2	101.0
*ckt*1	274.9	150	566	318	1129.2	1254.0
*ckt*2	352.9	1000	836	297	2080.5	767.0
$\sum_{total}$		1570	3658	1738	3497.5	2234.5

We see that our TPI technique gives very good results, especially for the largest industrial design, *ckt*2, in which ATPG time was reduced by 63.1% and test data volume by 64.5%. For *ckt*1, the ATPG CPU time increased because compaction of test vectors took significantly longer after the TPI step.

It is important to understand the strengths and weaknesses of each type of TP presented in Chapter 4. So, we inserted 20, 40, 60, 80, and 100 TPs of each type in the designs *s*9234 and *b*22*s*. We averaged the ATPG time and test volume values. The reductions in ATPG time and test volume in all of these runs are shown in Table 6.4. From the results we see that, inserting 20, 40, and 60 OPs produced good results. But after that, inserting more OPs produced far less extra reduction. However, in the case of CTPs, inserting even more than 60 CTPs significantly reduced the test volume. This is because inserting more CTPs improves both controllability and observability. Since typically a large number of control points are necessary to improve the controllability of a circuit, more CTPs produced better results.

We see that ITPs reduced test volume only by up to 18.8%. To understand the usefulness of ITPs, we conducted another experiment in which we inserted 30 OPs and 30 ITPs into *s*9234 and *b*22*s*. The average reduction in test volume

Table 6.4: Test Vector Length Reduction during ATPG as a Function of # and Type of Test Points Inserted

Test Point Type	*# Test Points Inserted*									
	20		40		60		80		100	
	% *Leng. Red.*	% *Time Red.*	% *Leng. Red.*	% *Time Red.*	% *Leng. Red.*	% *Time Red.*	% *Leng. Red.*	% *Time Red.*	% *Leng. Red.*	% *Time Red.*
OP	27.4	22.8	28.0	27.9	29.1	30.3	29.7	32.3	29.5	32.5
CTP	10.7	18.4	14.0	20.6	37.2	31.6	39.5	35.5	44.6	38.1
ITP	16.3	20.7	12.2	18.3	16.4	17.9	14.2	18.0	12.1	18.8

was 31.7%, which is much larger than inserting 60 OPs (in which the reduction was only 29.7%). So the ITP is useful on occasions where there is not enough unused hardware (two flops and a MUX) to construct a CTP, but we have enough hardware (one flop and MUX) to build an ITP. We also conducted experiments with PCPs. Experiments indicated that test volume reduced by 1-2% by inserting very few (10-20) PCPs. The strength of PCPs is that PCPs can be inserted even on timing critical paths without degrading the timing and they do not need any MUX. Next, we will compare our work with prior work on TPI.

6.1.2.1 Comparison with Prior Work

Several TPI methods have been proposed in the past that differ in the way they choose locations to insert test points. Table 6.5 shows the comparison. Our results in Table 6.5 are the average values of our method on the ISCAS '89 ($s9234$), ITC '99 ($b20s$, $b21s$ and $b22s$) and industrial benchmarks ($ckt1$, $ckt2$). We reduced test volume by 56.1% for the industrial benchmark circuits and by 52.5% for the ISCAS/ITC benchmarks. Even though Guezebroek *et al.* [23, 24] achieve a good reduction in test volume, they do not get a good reduction in ATPG CPU time. This is because their TPI program consumes enormous CPU time. We use several optimization techniques, so our tool consumes very little run time ($<$ 1 *min* for all benchmarks put together) to insert test points. In contrast, the tool of Yoshimura *et al.* [72] consumes close to one hour of CPU time for inserting test points. Guezebroek *et al.* [24] and Yoshimura *et al.* [72] do not report the ATPG

CPU time reduction. These techniques all were done on disjoint sets of circuits.

Table 6.5: Comparison with Previous Work (All Achieve Near 100% FE)

Method	*% Average Vector Length Reduction*	*% Average CPU (s) Time Reduction*
Inserting CPs	**52.5**	36
Inserting OPs	8.8	**37**
Guezebroek et al. [23]	40.7	-3.76
Guezebroek et al. [24]	51.8	Not Reported
Yoshimura et al. [72]	44.4	Not Reported

6.2 Cell-based ASICs

In Chapter 5, we described a novel DFT technique to enhance the performance of a broadcast scan-based compressor for regular cell-based ASICs. We implemented the proposed TPI scheme and the layout-aware scan chain re-ordering scheme in C. Experiments were conducted with ISCAS '89, ITC '99, and industrial benchmark (see Table 6.9) circuits to validate the efficacy of the proposed DFT technique.

6.2.1 Test Point Insertion

First, we evaluated the performance of the TPI step by inserting a fixed number of test points (35 for *b*20*s* and 60 each for *b*21*s* and *b*22*s*) into different scan configurations of the benchmarks. Even though we inserted 60 test points into *b*20*s*, our tool was able to find only 35 signal lines whose gain value was higher than the threshold value as there were a very small number of correlated SFFs. The results are shown in Table 6.6. The column labeled *# SCs* represents the total number of internal scan chains in that design and *Additional Faults* represents the additional number of faults detected in the parallel mode due to the TPI step. The column labeled *Test Volume* represents the total test volume (in bits) generated for the circuits without TPs (*Before*) and the circuit with TPs (*After*). The column *CPU Time* represents the total ATPG run time in *seconds*. We see that

Table 6.6: Results for Inserting Complete Test Points in Broadcast Scan

Ckt.	# *SCs*	*Additional Faults*	*Test Volume*		*CPU Time*	
			Before	*After*	*Before*	*After*
	8	858	151.3	85.5	98.6	54.4
$b20s$	16	606	146.6	118.8	114.5	60.6
	20	480	144.5	121.1	123.4	62.4
	8	1055	143.2	98.6	110.4	69.5
$b21s$	16	779	141.6	128.6	124.0	77.9
	20	400	126.0	140.0	13.0	56.1
	8	803	153.8	123.4	143.4	98.5
$b22s$	16	1186	132.9	126.1	177.5	111.3
	20	1165	162.4	141.1	181.8	118.5
$s9234$	8	571	25.8	23.3	11.0	6.8
	16	854	38.7	24.7	13.9	8.5
$s13207$	8	576	43.5	36.7	11.6	11.3
	16	654	39.6	36.3	12.2	11.8
$s38417$	16	589	130.1	70.2	46.0	37.6
	24	493	121.9	73.0	48.7	39.1
$s38584$	20	552	85.9	64.7	32.3	29.4
	24	319	74.7	74.7	32.5	30.0
$ckt1$	32	1051	122.6	99.1	1505.3	1399.6

the TPI step alone helps reduce the compressed test data for the broadcast scan architecture by up to 43.5%. Also, the ATPG CPU time is reduced by 44.8%, which is very useful in reducing the overall design turn-around time. An added advantage of the TPI step is that it helps the broadcast scan test generator tool to detect a few extra *hard-to-detect* faults that were originally undetected. Hence, we obtained a marginal improvement in fault coverage while also reducing the test volume. To confirm this, we inserted 200 CTPs into $b17s$ with one scan group comprising 32 internal scan chains. For this design, the test volume before inserting the test points was $453KB$ and the fault coverage was only 92.45%. After inserting the test points, the fault coverage was 98.5% (fault efficiency was 99.9%) and the test volume was $496KB$. Hence, the proposed TPI step can significantly improve the test quality using very little extra test input.

In another experiment with $b20s$ with eight internal scan chains, we set the target fault efficiency to 99% and generated two broadcast scan test vectors, one

for the original design and the other for the design with the test points. *The inserted test points completely eliminated the need for the serial scan mode test generation step as the parallel mode achieved the target 99% fault efficiency and reduced the test volume by 70.9%.*

6.2.2 Scan Chain Re-ordering

Next, we performed scan chain re-ordering without inserting any test points. Results (see Table 6.7) indicate that the scan re-ordering step can reduce the overall test volume only if there is a large number of artificially untestable faults. This was confirmed when we configured *b21s* to contain 24 internal scan chains that had 400 artificially untestable faults. Upon re-ordering the scan chain, 300 additional faults were tested in the parallel mode that resulted in a 49% reduction in test volume.

Table 6.7: Results for the Broadcast Scan Chain Re-ordering Operation

Ckt.	#	*Test Volume*		*CPU Time (s)*	
	SCs	*Before*	*After*	*Before*	*After*
b20s	8	151.3	139.1	98.6	93.3
	16	146.6	141.2	114.5	113.6
	20	144.5	138.4	123.4	119.2
b21s	8	143.2	133.8	110.4	104.9
	16	141.6	142.9	124.0	129.2
	20	126.0	128.8	133.0	131.6
b22s	8	167.5	165.0	143.4	146.3
	16	163.7	177.0	177.5	111.3
	20	213.9	185.9	181.8	118.5

6.2.3 The Overall DFT Results

Table 6.8 shows the results obtained by running both the TPI and the scan chain re-ordering step. We see that the proposed technique reduces the compressed test volume by up to 46.6%. We also see that the proposed DFT technique drastically improves the performance of the ATPG tool and the test compaction tool. Figure 6.4 shows the impact on the backtrack count of the ATPG tool due

Table 6.8: The Overall DFT Results

Ckt.	# *SCs*	*Test Volume*			*CPU Time (s)*	
		Before	*After*	*Reduction %*	*Before*	*After*
*b*20*s*	8	151.3	80.8	46.6	98.6	60.0
*b*21*s*	8	143.2	106.1	25.9	110.4	72.1
*b*22*s*	20	213.9	137.4	35.8	181.8	116.6
*s*9234	8	25.8	14.2	45.1	11.0	6.8
*s*13207	20	46.3	40.7	12.1	12.4	11.8
*s*38417	16	130.1	71.0	45.4	48.7	40.0
*s*38584	20	85.9	49.2	42.7	32.3	29.2
*ckt*1	32	122.6	99.7	18.7	1505.3	1344.7

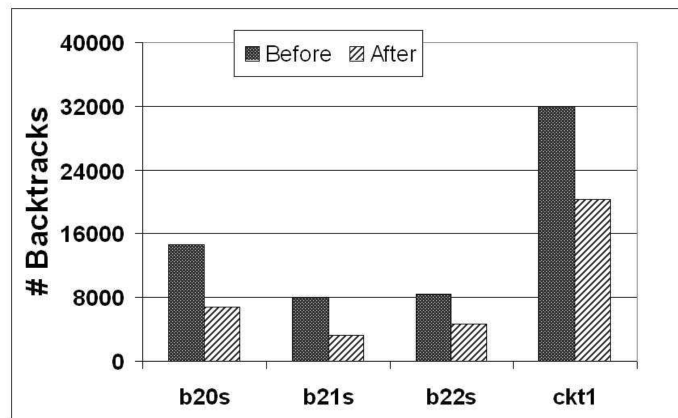


Figure 6.4: The ATPG Backtrack Count

to the proposed scheme and Figure 6.5 shows the impact on the reduction in test volume achieved by the compactor. Here *Before* means the circuit without the proposed DFT scheme and *After* represents the circuit incorporating the proposed DFT scheme. We see that our technique reduces the backtrack count by up to 59.4% and helps improve the compaction tool's performance by up to 9% (i.e., the compactor is able to compact another 9% additional test vectors to achieve a greater reduction in test volume). *These two figures confirm that the proposed DFT scheme makes the design more amenable for the test generation and the test compaction tools.* Table 6.9 shows the CPU time (in s) and peak memory (in MB) consumed by our DFT procedure. The column labeled *# Lines* represents the number of signal lines in the circuit and *# SFFs* represents the number of SFFs in the circuit.

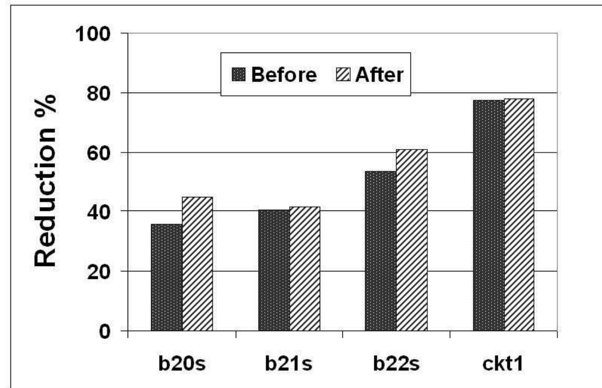


Figure 6.5: The Compactor Performance

Table 6.9: CPU Time and Peak Memory of our Tool

Ckt.	# *Lines* ($\times 10^3$)	# *FFs*	*TPI* *Time* (*s*)	*TPI* *Memory* (*MB*)	*Scan RO* *Time* (*s*)	*Scan RO* *Memory* (*MB*)
b17s	127.7	1415	38.8	120	728.5	123
b20s	48.8	490	4.3	49	37.7	50
b21s	50.7	490	4.8	91	34.0	92
b22s	78.0	735	7.1	63	61.1	95
s9234	26.2	228	< 1.0	11	< 1.0	12
s13207	39.4	669	< 1.0	13	< 1.0	14
s38417	108.7	1636	2.0	11	2.4	13
s38584	100.2	1452	1.3	11	2.4	52
ckt1	274.9	4699	13.3	122	35.5	130

6.3 Summary

In this chapter, we presented the results obtained by the proposed technique. We showed that our technique can reduce the test volume, test application time, and the test generation time for structured ASICs. We also showed that the proposed TPI scheme with a layout-aware scan chain re-ordering algorithm can drastically enhance the performance of an existing broadcast scan compressor.

Chapter 7
Conclusions and Future Work

7.1 Conclusion

In this dissertation, we presented a new TPI scheme for SAs and a new DFT scheme for regular cell-based ASICs to reduce the test data volume. Unlike traditional algorithms for TPI that insert test points to improve fault coverage, the proposed TPI scheme inserts TPs to facilitate the ATPG and the compaction algorithms to drastically reduce the test data volume and the test generation time. The ideas and concepts presented in this dissertation have made the following original contributions:

1. A zero-cost TPI for SAs. In Chapters 3 and 4, we presented a new test point insertion technique for SAs. The proposed technique utilizes unused cells, which exist in any SA in a large quantity, and re-configures them into different types of test points. Hence the proposed technique incurs no hardware overhead to reduce test generation time and test data volume for SAs. By inserting very limited numbers of OPs on large ISSP designs, we showed that the proposed TPI technique can reduce test generation time by up to 42.9% and test data volume by up to 25.9% while also achieving a near 100% fault efficiency. By inserting different types of TPs (see Chapter 4), we showed that the proposed TPI scheme can reduce test generation time by up to 63.1% and test data volume by up to 64.5% for very large industrial designs.

2. A TPI scheme to enhance the performance of a broadcast scan-based compressor for regular cell-based ASICs. In Chapter 5, we presented the first

DFT scheme for regular cell-based ASICs comprising test point insertion and layout aware scan chain re-ordering techniques to enhance the performance of a broadcast scan-based compressor. Experiments indicate that, by using our technique, the compressed and the compacted test vectors generated for a "broadcast scan" design can be further reduced by up to 46.6%. The proposed technique can also reduce the overall ATPG CPU time by up to 49.3%.

3. Linear-time algorithms to compute all of the proposed testability measures. We also presented linear-time algorithms to compute the testability measures proposed in this dissertation (see Sections 3.4, 4.4, and 5.5). The computation of these testability measures is shown to be accurate for designs comprising reconverging signals (see Table 3.1). Our testability measures are more accurate than prior testability measures because we account for the signal correlations. Hence, the testability measures presented in our work are scalable and it can be used to analyze the testability of very large industrial designs.

7.2 Future Work

This work can be extended in the following directions:

1. Test Point Insertion for Advanced Fault Models. In this work, we only considered the single stuck-at fault model. However, as mentioned earlier in Chapter 1, other advanced fault models like the bridging fault, the delay fault, the crosstalk fault, and the N-detection fault model are now necessary to achieve the desired test quality. One can propose new test point insertion heuristics for detecting these advanced faults as well.

2. Test Point Insertion for BIST. As mentioned earlier in Section 2.5.1, several authors have proposed novel ideas to improve the fault coverage for BIST.

However, not much work is available in the literature that focuses on improving the fault coverage for BIST to support advanced fault models. This work can be easily extended for the BIST based systems as well.

3. Test Point Insertion for Other Test Compressors. In Chapter 2, we presented several test compression that are available in the literature. One can extend the proposed test point insertion scheme to enhance the performance of other compressors as well.

4. Novel Test Data Compressor. New test compressors can be implemented and the proposed test point insertion can be used to boost the fault coverage for the new test compressors.

5. Impact of Test Points on Response Compactors. Mitra and Kim [41] and Rajski *et al.* [53] have presented new types of response compactors that can compact test responses with several don't cares in them. One can propose new test point insertion algorithms that can help block the don't cares from being propagated to the primary/pseudo-primary outputs. Hence, test points can be used to enhance test compactors as well.

6. Novel Scan Chain Routing Algorithm for Test Data Compressors. In this work, we presented a novel layout-aware scan chain re-ordering algorithm to enhance the performance of a broadcast scan-based test data compressor. Not much work has been reported in the literature that routes scan chains while also improving the testability of the circuit. The scan chain re-ordering heuristic presented in this dissertation can be extended to implement a scan chain routing algorithm that minimizes the scan chain routing overhead while also improving the testability of the designs for test data compressors.

Appendix A
User's Guide

A.1 Directory

All of the executables and scripts required to insert test points are available at

$$/caip/u8/rajamani/research/bin \tag{A.1}$$

A.2 Command Synopsis

1.	genObsPoints	-run *input.v* $tName$ -n # *TPs* -o $TPList$
2.	modifyDesign	[-issp|-noissp] $TPList$ *input.v* *output.v*
3.	genTestPoints	-run *input.v* $tName$ -tpConfig $TPConfig$ -o $TPList$

The above commands are used to insert a specified number of test points into a given structured ASIC design described in verilog format. The definition for each of the options in the command is given below. The first command dumps

Command Line Argument	Their Meaning
input.v	The input verilog file name
output.v	The output verilog file name
$tName$	The top level module name
# *TPs*	The # TPs to be inserted
$TPList$	The output file containing the list of signals for inserting observe points
$TPConfig$	Test point configuration file

the following three types of information into the $TPList$ file: a) the list of signals

into which test points have to be inserted, b) the list of unused scan flip-flops that should be used to build each test point, and c) the test point type (in this case, an observation point). The second command is a PERL script used to modify the netlist to insert the test points at each of the signals in $TPList$. The -issp option uses the unused scan flip-flops to implement the test points. However, when the -noissp option is used, the program adds extra scan flip-flops to implement these test points. The file $sgFile$ contains the scan group information. An example of an $sgFile$ is given in Figure A.1. The *numScanChannel* represents the number of scan groups or the number of scan pins in the design. The *numScanChain* represents the number of scan chains driven by a particular scan pin. Here, the number of scan chains driven by $SCPINA$ is four and by $SCPINB$ is three. The tokens $SFF1$, $SFF2$, ..., $SFF24$ in Figure A.1 represent the names of the scan flip-flops. The third command is used to insert different types of test points into the design using a configuration file $TPList$. Each line in the $TPList$ file specifies the number of test points and its type (OP, CP, PCP, CTP, and ITP) will have to be inserted. Here, OP means observation points, CP means control points, PCP means pseudo-control points, CTP means complete test points, and ITP means inversion test points.

```
numScanChannel 2

numScanChain 4 SCPINA
SFF1 SFF2 SFF3 SFF4
SFF5 SFF6 SFF7 SFF8
SFF21 SFF22 SFF23 SFF24

numScanChain 3 SCPINB
SFF9 SFF10 SFF11
SFF12 SFF13 SFF14
SFF15 SFF16 SFF17
SFF18 SFF19 SFF20
```

Figure A.1: Scan Group and Scan Chain Information File

The above two commands should be used to perform the proposed design-for-test scheme for regular cell-based ASICs. The first command uses the -ls option

1.	*bScan*	-run *input.v* *tName* -ls *sgFile* -o *TPList* -tpConfig *TPConfig*
2.	*bScan*	-run *input.v* *tName* -ps *current.SC* -o *new.SC*

to insert the requested number of test points into the file called *TPList*. The -ps option is used to perform the scan chain re-ordering. The *current.SC* is a file containing the current scan chain ordering information. The format of this file is same as that of the *sgFile* and it is shown in Figure A.1. The program generates the new scan ordering and dumps it into the file called *new.SC*.

Command Line Argument	Their Meaning
input.v	The input verilog file name
tName	The top level module name
TPList	The output file containing the list of signals for inserting test points
TPConfig	Test point configuration file
current.SC	The current scan chain order file
new.SC	The new scan chain order file

References

[1] S. B. Akers and W. Jansz. Test Set Embedding in Built-in Self-test Environment. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 257–263, 1989.

[2] D. H. Baik, K. K. Saluja, and S. Kajihara. Random Access Scan: A Solution to Test Power, Test Data Volume and Test Time. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 883–888, Jan. 2004.

[3] K. Balakrishnan. A Dynamic Programming Approach to the Test Point Insertion Problem. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 195–705, June 1987.

[4] K. J. Balakrishnan and N. A. Touba. Relating Entropy Theory to Test Data Compression. In *Proc. of the European Test Symp.*, pages 94–99, 2004.

[5] P. Bardell, W. H. McAnney, and J. Savir. *Built-In Test for VLSI.* New York: Wiley-Interscience, 1987.

[6] I. Bayraktaroglu and A. Orailoglu. Test volume and Application Time Reduction Through Scan Chain Concealment. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 151–155, 2001.

[7] I. Berger and Z. Kohavi. Fault Detection in Fanout-Free Combinational Networks. *IEEE Trans. on Computer Aided Design*, C-22(1):908–914, 1973.

[8] S. Bhatia and P. Varma. Test Compaction in a Parallel Access Scan Environment. In *Proc. of the Asian Test Symp.*, pages 300–305, Nov. 1997.

[9] E. Brglez. On Testability Analysis of Combinational Networks. In *Proc. of the Int'l. Symp. on Circuits and Systems*, pages 221–225, 1984.

[10] F. Brglez, C. Gloster, and G. Kedem. Hardware-based Weighted Random Pattern Generation for Boundary Scan. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 264–274, 1989.

[11] F. Brglez, P. Pownall, and P. Hum. Applications of Testability Analysis: From ATPG to Critical Path Tracing. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 705–712, 1984.

[12] A. J Briers and K. A. E Totton. Random Pattern Testability by Fast Fault Simulation. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 274–281, 1986.

[13] M. L. Bushnell and V. D Agrawal. *Essentials of Electronic Testing.* Boston: Springer, 2000.

[14] A. Carbine and D. Feltham. Pentium Pro Processor Design for Test and Debug. *IEEE Design and Test of Computers*, 15(3):77–82, July 1998.

[15] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler. A Transitive Closure Algorithm for Test Generation. *IEEE Trans. on Computer Aided Design*, 12(7):1015–1028, July 1993.

[16] A. Chandra and K. Chakrabarty. Frequency-directed Run-length (FDR) Codes with Application to System-on-a-chip Test Data Compression. In *Proc. of the VLSI Test Symp.*, pages 42–47, April 2001.

[17] A. Chandra and K. Chakrabarty. System-on-a-chip Test-data Compression and Decompression Architectures Based on Golomb Codes. *IEEE Trans. on Computer Aided Design*, 20(3):355–368, Mar. 2001.

[18] J. S. Chang and C. S. Lin. Test Set Compaction for Combinational Circuits. *IEEE Trans. on Computer Aided Design*, 14(11):1370–1378, Nov. 1995.

[19] J. Cong, Y. Fan, X. Yang, and Z. Zhang. Architecture and Synthesis for Multi-cycle Communication. In *Proc. of the Int'l. Symp. on Physical Design*, pages 190–196, 2003.

[20] W. Daehn and J. Mucha. Hardware Test Pattern Generators for Built-in Test. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 110–113, 1981.

[21] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia Controllability / Observability Analysis Program. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 190–196, June 1980.

[22] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici. Variable-length Input Huffman coding for System-on-a-chip Test. *IEEE Trans. on Computer Aided Design*, 22(6):783–796, June 2002.

[23] M. J. Guezebroek, J. T. van der Linden, and A. J. van de Goor. Test Point Insertion for Compact Test Sets. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 292–301, 2000.

[24] M. J. Guezebroek, J. T. van der Linden, and A. J. van de Goor. Test Point Insertion that Facilitates ATPG in Reducing Test Time and Data Volume. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 138–147, 2002.

[25] I. Hamzaoglu and J. H. Patel. Reducing Test Application Time for Full Scan Embedded Cores. In *Proc. of the Int'l. Fault-Tolerant Computing Symp.*, pages 260–267, 1999.

[26] I. Hamzaoglu and J. H. Patel. Test Set Compaction Algorithms for Combinational Circuits. *IEEE Trans. on Computer Aided Design*, 19(8):957–963, Aug. 2000.

[27] J. P. Hayes and A. D. Friedman. Test Point Placement to Simplify Fault Detection. In *Proc. of the Int'l. Fault-Tolerant Computing Symp.*, pages 73–78, 1973.

[28] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois. Generation of Vector Patterns through Reseeding of Multiple-polynomial Linear Feedback Shift Registers. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 120–129, 1992.

[29] F. F. Hsu, K. M. Butler, and J. H. Patel. A Case Study on the Implementation of the Illinois Scan Architecture. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 538–547, 2001.

[30] B. Hu, H. Jiang, Q. Liu, and M. M. Sadowska. Synthesis and Placement Flow for Gain-Based Programmable Regular Fabrics. In *Proc. of the Int'l. Symp. on Physical Design*, pages 197–203, 2003.

[31] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proc. of the Inst. of Electrical and Radio Engineers*, 40(9):1098–1101, Sept. 1952.

[32] V. S. Iyengar and D. Brand. Synthesis of Pseudo-Random Pattern Testable Designs. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 501–508, 1989.

[33] A. Jas, J. G. Dastidar, and N. A. Touba. Scan Vector Compression/Decompression Using Statistical Coding. In *Proc. of the VLSI Test Symp.*, pages 114–120, 1999.

[34] A. Jas, J. Ghosh-Dastidar, Mom-Eng Ng, and N.A. Touba. An Efficient Test Vector Compression Scheme Using Selective Huffman Coding. *IEEE Trans. on Computer Aided Design*, 22(6):797–806, June 2003.

[35] A. Jas and N. A. Touba. Test Vector Decompression via Cyclical Scan Chains and its Application to Testing Core-based Designs. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 458–464, Oct. 1998.

[36] S. Kajihara, K. Taniguchi, K. Miyase, I. Pomeranz, and S. M. Reddy. Test Data Compression Using Don't-care Identification and Statistical Encoding. In *Proc. of the Asian Test Symp.*, pages 67–72, Nov. 2002.

[37] B. Koenemann. LFSR-coded Test Patterns for Scan Designs. In *Proc. of the European Test Conf. (ETC)*, pages 237–242, 1991.

[38] C. V. Krishna and N. A. Touba. Reducing Test Data Volume Using LFSR Reseeding with Seed Compression. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 321–330, 2002.

[39] K. J. Lee, J. J. Chen, and C. H. Huang. Using a Single Input to Support Multiple Scan Chains. In *Proc. of the Int'l. Conf. on Computer-Aided Design*, pages 74–78, 1998.

[40] L. Lei and K. Chakrabarty. Test Data Compression Using Dictionaries with Fixed-length Indices. In *Proc. of the VLSI Test Symp.*, pages 219–224, April 2003.

[41] S. Mitra and K. S. Kim. X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 311–320, 2002.

[42] F. Mo and R. K. Brayton. Fishbone: A Block-Level Placement and Routing Scheme. In *Proc. of the Int'l. Symp. on Physical Design*, pages 204–209, 2003.

[43] M. Nakao, S. Kobayashi, K. Hatayama, K. Iilima, and S. Terada. Low Overhead Test Point Insertion For Scan-based BIST. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 348–357, 1999.

[44] T. Okamoto, T. Kimoto, and N. Maeda. Design Methodology and Tools for NEC Electronics' Structured ASIC ISSP. In *Proc. of the Int'l. Symp. on Physical Design*, pages 90–96, April 2004.

[45] A. Pandey and J. H. Patel. Reconfiguration Technique for Reducing Test Time and Test Volume in Illinois Scan Architecture Based Designs. In *Proc. of the VLSI Test Symp.*, pages 9–15, 2002.

[46] C. Patel, A. Cozzie, H. Schmitt, and L. Pillegi. An Architectural Exploration of Via Patterned Gate Arrays. In *Proc. of the Int'l. Symp. on Physical Design*, pages 184–189, 2003.

[47] I. Pomeranz, L. N. Reddy, and S. M. Reddy. A Method to Generate Compact Test Sets for Combinational Circuits. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 291–299, 1991.

[48] I. Pomeranz and S. M. Reddy. COMPACTEST: A Method to Compact Test Sets for Combinational Circuits. *IEEE Trans. on Computer Aided Design*, 12(7):1040–1049, July 1993.

[49] I. Pomeranz and S. M. Reddy. Static Test Compaction for Scan-based Designs to Reduce Test Application Time. In *Proc. of the VLSI Test Symp.*, pages 198–203, Nov. 1998.

[50] J. Rajski and J. Tyszer. *Arithmetic Built-In Self-Test for Embedded Systems.* Prentice Hall, Upper Saddle River, New Jersey, 1998.

[51] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee. Embedded Deterministic Test. *IEEE Trans. on Computer Aided Design*, 23(5):776–792, May 2004.

[52] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R.Thompson, K-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide, and J. Qian. Embedded Deterministic Test for Low Cost Manufacturing Test. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 301–310, 2002.

[53] J. Rajski, J. Tyszer, C. Wang, and S. M. Reddy. Convolutional Compaction of Test Responses. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 745–754, 2003.

[54] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz. On Test Data Volume Reduction for Multiple Scan Chain Designs. *ACM Trans. on Design Automation of Electronic Systems*, 8(4):460–469, 2003.

[55] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. W. Williams. A Reconfigurable Shared Scan-In Architecture. In *Proc. of the VLSI Test Symp.*, pages 9–14, 2003.

[56] R. Sankaralingam, R. Oruganti, and N. A. Touba. Static Compaction Techniques to Control Scan Vector Power Dissipation. In *Proc. of the VLSI Test Symp.*, pages 35–30, Nov. 2000.

[57] Y. Savaria, M. Youssef, B. Kaminska, and M. Koudil. Automatic Test Point Insertion for Pseudo-random Testing. In *Proc. of the Int'l. Symp. on Circuits and Systems*, pages 1960–1963, 1991.

[58] M. H. Schulz, E. Trischler, and T.M. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. *IEEE Trans. on Computer Aided Design*, 7(1):126–137, June 1988.

[59] B. Seiss, P. Trouborst, and M. Schulz. Test Point Insertion for Scan-based BIST. In *Proc. of the European Design Automation Conf.*, pages 253–262, 1991.

[60] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell. Zero Cost Test Point Insertion Technique to Reduce Test Volume and Test Generation Time for Structured ASICs. In *Proc. of the Asian Test Symp.*, pages 339–348, Nov. 2006.

[61] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell. Zero Cost Test Point Insertion Technique for Structured ASICs. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 357–363, Jan. 2007.

[62] N. Tamarapalli and J. Rajski. Constructive Multi Phase Test Point Insertion for Scan-based BIST. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 649–658, 1996.

[63] K.Y. Tong, V. Kheterpal, V. Rovner, L. Pileggi, and H. Schmit. Regular Logic Fabrics for a Via Patterned Gate Array (VPGA). In *Proc. of the Custom Integrated Circuits Conf.*, pages 53–56, Sept. 2003.

[64] N. A. Touba. *Synthesis Techniques for Pseudo-Random Built-In Self-Test.* PhD thesis, Stanford University, August 1996. Departments of Electrical Engineering and Computer Science.

[65] N. A. Touba and E. J. McClusky. Test Point Insertion Based on Path Tracing. In *Proc. of the VLSI Test Symp.*, pages 2–8, June 1996.

[66] H.-C. Tsai, K.-T. Cheng, C.-J. Lin, and S. Bhawmik. A Hybrid Algorithm for Test Point Selection for Scan-based BIST. In *Proc. of the IEEE/ACM Design Automation Conf.*, pages 478–483, 1997.

[67] L. T. Wang, X. Wen, H. Furukawa, F. S. Hsu, S. H Lin, S. W Tsai, K. S. A. Hafez, and S. Wu. VirtualScan: A New Compressed Scan Technology for Test Cost Reduction. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 916–925, 2004.

[68] F. G. Wolff and C. Papachristou. Multiscan-based Test Compression and Hardware Decompression Using LZ77. In *Proc. of the Int'l. Test Conf. (ITC)*, pages 331–339, 2002.

[69] K. C Wu and Y. W Tsai. Structured ASIC – Evolution or Revolution? In *Proc. of the Int'l. Symp. on Physical Design*, pages 103–106, 'April 2004.

[70] H.-J. Wunderlich. Self Test Using Unequiprobable Random Patterns. In *Proc. of the Int'l. Fault-Tolerant Computing Symp.*, pages 258–263, 1987.

[71] H.-J. Wunderlich and S. Hellebrand. The Pseudo-exhaustive Test of Sequential Circuits. *IEEE Trans. on Computer Aided Design*, 11(1):26–33, Jan. 1992.

[72] M. Yoshimura, T. Hosokawa, and M. Ohta. A Test Point Insertion Method to Reduce the Number of Test Patterns. In *Proc. of the Asian Test Symp.*, pages 298–304, Nov. 2002.

[73] M. Youssef, Y. Savaria, B. Kaminska, and M. Koudil. Methodology for Efficiently Inserting and Condensing Test Points. *IEE Proceedings-E*, 140(2):154–160, May 1993.

[74] Behrooz Zahiri. Structured ASICs: Oppurtunity and Challenges. In *Proc. of the Int'l. Conf. on Computer Design*, pages 404–409, 2003.

Vita

Rajamani Sethuram
Cell: 908-812-5942
E-mail: thisisraja@yahoo.com
92C, Cedar Lane,
Highland Park, NJ -08904

Research Interests

Computer Aided Design (*CAD*) of *Very Large Scale Integrated* (VLSI) Circuits. Computer algorithms for functional and structural *automatic test-pattern generation* (ATPG). Algorithms for fault collapsing. *Design for testability.* Test data compaction and compression. Architectural level low-power design techniques.

Academic Degrees

Ph.D in Computer Engineering, Rutgers university
Thesis Title: Reducing Digital Test Volume Using Test Point Insertion
Oct. 2005 - May 2007
Advisor: Prof. M. L. Bushnell
GPA: 4.00 / 4.00

M.S. in Computer Engineering, Rutgers University
Oct. 2002 - Oct. 2005
Thesis Title: Sequential ATPG Using Implications over Time
Advisor: Prof. M. L. Bushnell
GPA: 3.83/4.00

B. Tech in Computer Science, Indian Institute of Technology, Roorkee
Aug. 1997 - May 2001
First Division with Honors

Conference Reviewing

1. Reviewer, *IEEE Int'l. Conf. on VLSI Design* - 2005
2. Reviewer, *IEEE Int'l. Conf. on VLSI Design* - 2006
3. Reviewer, *IEEE Int'l. Conf. on VLSI Design* - 2007

Professional Experience

1996	Graduated from Kendriya Vidyalaya, Madurai, India
1996-97	Undergraduate work in Physics, The American College, Madurai, India
1997-01	Bachelors in Computer Science, Indian Institute of Technology, Roorkee, India
2001-02	Research and Development Engineer Synopsys India, Bangalore, India
2002-07	Graduate work in Electrical Engineering, Rutgers University
2002-05	Teaching Assistant, ECE Dept., Rutgers Univeristy, New Brunswick, New Jersey
2005	M.S., Rutgers University, New Brunswick, New Jersey Majored in Computer Engineering
2005-07	Research Assistant, NEC Laboratories America, Princeton, New Jersey
2007	Ph.D., Rutgers University, New Brunswick, New Jersey Majored in Computer Engineering

Publications

1. R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell, "Zero Cost Test Point Insertion Technique to Reduce Test Data Volume and ATPG CPU Time for Structured ASICs," *Proc. of the IEEE Asian Test Symposium*, pp. 339-348, 2006

2. R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell,"Zero Cost Test Point Insertion Technique for Structured ASICs," *Proc. of the Int'l. Conf. on VLSI Design*, pp. 357-363, 2007

3. R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell, "Enhancing the Performance of Broadcast Scan-based Test Compression Using Test Point Insertion and Scan Chain Re-ordering," Submitted to the *IEEE Int'l. Test Conf.*, 2007

4. R. Sethuram and M. L. Bushnell, "A Graph Condensation-based Transitive Closure Algorithm for Implication Graphs to Identify False Paths," Accepted at the *IEEE North Atlantic Test Workshop*, May, 2007

5. G. D. Sandha, M. L. Bushnell, and R. Sethuram, “Combinational Hardware False Path Identification without Search,” Accepted at the *IEEE North Atlantic Test Workshop*, 2007

6. R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell, “A New Design for Test Technique for Broadcast Scan-based Test Compression,” Accepted at the *IEEE North Atlantic Test Workshop*, May, 2007

7. R. Sethuram and M. Parashar, “Ant Colony Optimization and its Application to Boolean Satisfiability of Digital VLSI Circuits,” Accepted for the *Int'l. Journal of Information Processing*, 2007

8. R. Sethuram and M. Parashar, “Ant Colony Optimization and its Application to Boolean Satisfiability of Digital VLSI Circuits,” *Proc. of the IEEE Int'l. Conf. on Advanced Computing and Communication*, pp. 507-512, 2006

9. R. Sethuram, O. Khan, H. V. Venkatanarayanan, and M. L. Bushnell, “Power Reduction Using a Neural Net Branch Predictor,” *Proc. of the IEEE Int'l. Conf. on VLSI Design*, pp. 161-168, 2007