SPARTAN: A SPECTRAL AND ENTROPY-BASED PARTIAL-SCAN AND TEST POINT INSERTION ALGORITHM

BY OMAR I. KHAN

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

October, 2007

ABSTRACT OF THE DISSERTATION

SPARTAN: A Spectral and Entropy-Based Partial-Scan and Test Point Insertion Algorithm

by Omar I. Khan

Dissertation Director: Prof. Michael L. Bushnell

We propose a new partial-scan algorithm and two new *test point insertion* (TPI) algorithms, TPI1 and TPI2, to improve the testability, reduce the test volume, and reduce the test application time of the *circuit-under-test* (CUT). The partial-scan algorithm and the test point insertion algorithms use toggling rates of the flip-flops and the candidate test point lines, analyzed using *digital signal processing* (DSP) methods, and Shannon entropy measures of flip-flops and candidate test point lines to select scan flip-flops and test points. We also discovered a new and the most potent combination of *design-for-testability* (DFT) and sequential *automatic test-pattern generation* (ATPG) algorithms.

Testing of complex circuits is time consuming and extremely difficult. Most circuit designs employ full-scan to alleviate the testability problems. Full-scan has two advantages. It uses a combinational ATPG for test generation and full-scan can be used to scan out the states of the CUT for post-silicon debug. But, full-scan requires long scan-in and scan-out sequences, thus resulting in longer testing times, leading to increased test power and test cost. Another disadvantage of using full-scan is increased delay on the critical paths of the CUT usually by 5 to 10%. The flip-flops on the critical path will have extra hardware.

Partial-scan with test points can help reduce test volume and test application time and achieve very high *fault coverages* (FCs). This reduces the test time and lowers the test power because the partial-scan chain requires shorter scan-in and scan-out sequences. Also, by avoiding scanning of flip-flops and addition of test points on critical paths, the CUT can operate

without any extra delay.

The average test volume of SPARTAN partial-scan with TPI1 (excluding s38417) is 17.23% shorter than full-scan with TRAN and the average test volume of SPARTAN partial-scan with TPI2 (excluding s38417) is 7.55% shorter than full-scan with TRAN. SPARTAN's average test application time (excluding s38417) with partial-scan only is 44.30% longer than full-scan with TRAN. The average test application time of SPARTAN partial-scan with TPI1 (excluding s38417) is 18.05% shorter than full-scan with TRAN and the average test application time of SPARTAN partial-scan with TPI2 (excluding s38417) is 8.71% shorter than full-scan with TRAN.

Acknowledgements

I would like to express my gratitude to everyone who has helped me complete this degree. I am much obliged to Prof. Michael Bushnell and Prof. Vishwani Agrawal whose cooperation, encouragement, and support helped me achieve my goals. It is Prof. Bushnell's sincere and relentless efforts that helped transform this dissertation into first-rate work. This degree is an important milestone in my life and the entire experience has been daunting and intimidating, but inevitably gratifying. I would also like to thank Prof. Tapan Chakraborty for his encouragement and help with this project.

I would also like to acknowledge my parents for their steadfastness and support throughout my graduate school years. They stood by me through thick and thin and I would have never achieved anything in my life without their persisting motivation and hard work.

There is a very special group of people in my life that helped me stay focused and sane, my friends. I want to extend my thanks to my best friends James Chun, Sri Kakrala, Dhruv Choksey, Sumit Ahluwalia, Rajamani Sethuram, Rami Elghandour, Mohammad Soliman, and Aditya Jagirdar. I want to thank James for making my stay at CAIP so comfortable and interesting. I would also like to thank all my friends Jeannie, Prasann, Ken, Hari, Roy, Varadan, Raghu, Gaurav, and Baozhen.

I am also very thankful to CAIP for providing a very constructive environment. I would specially like to thank the technical staff Bill Kish and Skip Carter for their support. I am obliged to Prof. Janak Patel from University of Illinois at Urbana-Champaign for the sequential test generator GATEST and I thank Suresh Devanathan for his compactor program. I am thankful to NSF for providing the financial support for this project.

Dedication

To my parents Rehana Ilyas and Ilyas A. Khan, my grandparents Sharif A. Khan, Nafisa Sharif, Syed Manzoor Hussain and Qamar Jahan Begum, my brother Faraz I. Khan, my sister Sana I. Khan, and all my teachers

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Rapid advancements in chip fabrication technology allow hundreds of millions of transistors on a single VLSI chip, at the cost of increased testing difficulty. An AMD K8™ microprocessor has 233 million transistors, the Intel Smithfield™ has 230 million transistors, and the IBM CELL™ processor has 234 million transistors [60]. The testing cost is alleviated by improving the circuit testability, using *design-for-testability* (DFT) techniques. Logic DFT consists of *ad hoc* methods, such as *test point insertion* (TPI), and structured techniques. Structured DFT can be further split into full-scan, partial-scan and *random-access scan* (RAS) methods.

Most complex circuits today, except very high-end Intel™ and IBM™ microprocessors, employ full-scan to improve testability because it reduces the ATPG problem to a combinational one and full-scan designs are very useful for silicon debug. Also, scan chain insertion and test generation can be fully automated, and almost always require little manual DFT. Disadvantages of using full-scan are added delay to the data path (5-10% of clock period), long scan shifting sequences, over testing, and higher power consumption. Each flip-flop in the circuit has to be converted into a scan flip-flop, which requires the addition of a multiplexer to the input of each flip-flop. The multiplexer selects the data input during functional mode and the scan-in sequence during the test mode. The multiplexer adds delay, thus requiring the circuit to operate at a lower clock frequency. Long scan chains need long shift sequences, which consume much time because scan-in and scan-out of shift sequences is done at a lower clock rate to avoid circuit burnout. Using full-scan also causes the detection of redundant faults, because in a fully scanned circuit, *any* arbitrary state can be achieved, but in normal operation, only a tiny fraction of these states are legal functional states [59]. This can cause unnecessary rejection of otherwise fault-free circuits, resulting in yield loss. Since we are shifting sequences through all flip-flops, the circuit in test mode can consume up to 200% of the normal circuit power [27]. Partial-scan can help alleviate the problems with full-scan. Partial-scan has shorter scan chains and will require shorter shift sequences. Shorter shift sequences will reduce the time required to

scan the sequences in and out of the scan chain, reduce the toggling activity during shifting of the sequences in and out of the scan chain, and thus reduce the test power consumption. Partial-scan also eliminates the delay, incurred by full-scan, to the data path by avoiding scanning flip-flops on critical paths. Also, partial-scan has a lower hardware overhead because it scans less flip-flops than full-scan. A disadvantage of full-scan is a higher test generation time becuase the partial-scan circuit requires the use of a sequential test generator.

Partial-scan can be categorized into three categories: structure based [15, 17, 33, 56, 67], testability-measure based [78], and test-generation based [65] methods. Of the three categories, structure based methods have been most successful. Some partial-scan algorithms use a combination of testability measures, structural information, and test-generation information [47]. However, none of these methods have gained widespread use, because of inadequate fault coverage, severe problems and long computation times for sequential ATPG, and the need, therefore, to repeat partial-scan insertion many times. This leads to a number of cycles of partial-scan, sequential ATPG, and *ad hoc* testability insertion, which takes too long compared with full-scan and combinational ATPG. Also, most of the prior partial-scan methods have not emphasized high fault coverage, but rather selected the minimal number of flip-flops to scan. The industry prefers full-scan and combinational ATPG, because high fault coverage and rapid ATPG are more important to them than scanning fewer flip-flops. For the present, the comparison metric is full-scan and combinational ATPG. Therefore, we abandon the focus of the prior work on selecting the minimal number of flip-flops, and instead focus on maximizing the fault coverage, and minimizing *test volume* (TV) and *test application time* (TAT). This means that we will compare our results in this work to *mpscan* [78], the most successful prior partial-scan method.

In this dissertation, we present SPARTAN, a partial-scan algorithm and a compendium of partial-scan and test point insertion algorithms. The partial-scan algorithm is referred to as PS and the two test point insertion algorithms are called TPI1 and TPI2. The algorithm dissects the sequential circuit states in the spectral information theoretic domains. SPARTAN uses spectral analysis and entropy analysis for scan flip-flop and test point selection by logic simulation with an analysis of the circuit structure. We will show that these two new analysis techniques are superior testability measures to SCOAP [29, 30], COP [11], and probabilities.

Spectral Analysis

For the spectral analysis, we use the *Rademacher-Walsh transform* (RWT) matrix to extract *spectral coefficients* (SCs) for each candidate scan flip-flop and candidate test point in the circuit. The SCs represent the time domain data (flip-flop states and logic values of the test points) in the spectral domain. The spectral transformation of the time domain data is akin to using the Fourier transform for transforming the time domain data into the frequency domain. The spectral analysis is performed because certain properties (frequency content) of the time domain data are better represented in the spectral domain. The coefficients of the candidate scan flip-flop states and candidate test points in the spectral domain are used as a controllability measure. A low toggling candidate flip-flop or candidate test point line will have small SC values and thus will be difficult to control. If the average value of the SCs of a candidate flip-flop or candidate test point line is low (low toggling), the candidate flip-flop or candidate test point has low controllability and will be scanned or a test point will be inserted. To compute the observability, we first find, for each candidate flip-flop or candidate test point, all of the *primary outputs* (POs) in its fanout cone. We then compute the SCs of all the POs. Then we perform a SC comparison of each candidate flip-flop or candidate test point with each PO in its fanout cone. If the SCs of the candidate flip-flop or the candidate test point under consideration are identical to SCs of any one of the POs in its fanout cone, we consider that candidate flip-flop or candidate test point observable.

Entropy Analysis

SPARTAN uses entropy as a testability measure to select scan flip-flops and test points such that the entropy of the circuit is enhanced. The entropy of a line in the circuit is the amount of information (in bits) flowing through the line and the entropy of the circuit is the amount of information (in bits) flowing through the circuit. Increasing the entropy (information) of the circuit is akin to increasing the testability of the circuit [2]. The entropy analysis uses random-pattern testability analysis since $p(1)$ and $p(0)$ are required to compute entropy [57]. Using entropy analysis is analogous to using Fourier analysis. Given the time domain data, Fourier analysis gives the spectral components of the data. Similarly, $p(0)$ and $p(1)$ of each line in the circuit are like time domain data and the entropy analysis gives us the amount of information on each line.

The Need for Spectral and Entropy Analysis Combination

The spectral analysis for partial-scan is localized to each flip-flop in the circuit. A flip-flop that displays low toggling is selected for scanning without taking into account the effect that scanning this low toggling flip-flop may have on other flip-flops. This can cause the spectral analysis to scan more flip-flops than what may actually be required. Scanning all of the low toggling flip-flops may not be necessary because it is possible to increase the testability of the unscanned flip-flops by scanning a subset of the low toggling flip-flops. In contrast, when a flip-flop is tried for scanning, the entropy analysis only considers the changes in information flow through unscanned flip-flops in the *strongly connected component* (SCC) of the condensed circuit *structure graph* (s-graph) [75] to which the trial flip-flop belongs to. To keep the computation time tractable, the entropy analysis does not check for changes in information flow in the other SCCs (see Chapter 3). Moreover, a flip-flop can display a high toggling activity, but the toggling activity could be because the flip-flop is part of a cycle with combinational logic, which has odd inversion parity, between each flip-flop in the cycle (e.g., a ring oscillator). In this case, the spectral analysis will not select this flip-flop, but the entropy analysis may reveal that very little information is actually reaching the flip-flop from the PIs in its fanin cone and the high toggling flip-flop is a good candidate for scanning.

On the contrary, entropy analysis only considers SCCs that contain more than one flip-flop. This is because the entropy analysis grades the quality of a candidate flip-flop in a SCC by scanning it and computing the change in the entropy values of the unscanned flip-flops in the SCC. Since, a SCC with one flip-flop does not have other flip-flops for comparison, the spectral analysis is used for analyzing SCCs that contain only one flip-flop. Similarly, the spectral and entropy analysis combination is required for test point insertion.

In this dissertation we will show that SPARTAN attains a very high fault coverage, low test lengths, low test volume, and low test application times for the *ISCAS*-89 benchmarks. The results were compared to the full-scan results obtained using TRAN [14]. SPARTAN got a lower test volume and test application time than TRAN for six of the seven largest benchmarks. Lower test volume and test application time will reduce the test time and the power consumed during test. Since test cost is proportional to test time and test power, shorter test time and lower power consumption will reduce test cost. Zorian reported that that total test cost for a system is typically 40 to 50% of the total product cost [82]. Moreover, the test length, test volume and

test application time for SPARTAN can be further reduced by applying compression techniques (code-based, linear-decompression-based or broadcast-scan-based schemes) that are usually incorporated with full-scan [73].

The main goal of the previous DFT techniques, especially partial-scan algorithms, was attaining a high fault coverage while keeping the hardware overhead to a minimum. Instead, the main goals of the DFT techniques in this dissertation were high fault coverage and low test application time. High fault coverage is important because a low fault coverage will force the designer to manually add more DFT hardware to the circuit to boost the fault coverage. The reiteration of DFT insertion in the testing phase to boost the fault coverage will delay the completion of the project and may increase production time and time to market for the product. Adding the DFT hardware in the low testability parts of a circuit can help detect hard-to-detect faults easily and thus reduce the the test length and test application time. Although the added DFT hardware raises the silicon cost, this can be more than outweighed by a larger reduction in the test cost due to lower test volume and test application time. We will show that even though the SPARTAN algorithm illustrates the potential to replace full-scan completely, the ability of full-scan to shift out the complete state of the circuit for post-silicon debug can be very instrumental for DFT engineers. Since full-scan incurs increased delays on the clock path, the tradeoff can be using partial-scan for timing critical parts of the circuit and full-scan in the remaining circuit. Partial-scan can be the DFT technique of choice for very high-end microprocessors used in speed-critical servers.

All of the SPARTAN algorithm average fault coverages (97.44%, 97.60%, and 97.72%) were higher than *mpscan*'s average fault coverage (96.75%). TRAN had the highest average fault coverge (98.16%) and SPARTAN partial-scan with TPI2 has a slightly lower average fault coverage (97.72%). SPARTAN's average test volume (excluding s38417) with only partial-scan is 52.91% longer than full-scan with TRAN. The average test volume of SPARTAN partial-scan with TPI1 (excluding s38417) is 17.23% shorter than full-scan with TRAN and the average test volume of SPARTAN partial-scan with TPI2 (excluding s38417) is 7.55% shorter than full-scan with TRAN. SPARTAN's average test application time (excluding s38417) with partial-scan only is 44.30% longer than full-scan with TRAN. The average test application time of SPARTAN partial-scan with TPI1 (excluding s38417) is 18.05% shorter than full-scan with TRAN and the average test application time of SPARTAN partial-scan with TPI2 (excluding s38417) is 8.71% shorter than full-scan with TRAN. SPARTAN algorithms do not perform very

well on benchmark s38417 and the high test volume is skewing the average, therefore we did not include the test volume results of s38417 in the average calculations here.

1.1 Outline of the Thesis

This dissertation is organized as follows. In Chapter 2 we present a detailed survey of the structured DFT techniques, full-scan and partial-scan, in Sections 2.2 and 2.3. Section 2.4 summarizes some of the most popular test point insertion algorithms. Then we summarize the testability measures in Section 2.5. The information theory concepts and their applications in testing are described in Section 2.6. In Section 2.7, we discuss only GATEST because all the results in this dissertation were generated with GATEST. Chapter 3 describes the SPARTAN partial-scan algorithm. Chapter 4 describes the SPARTAN test point insertion algorithms. Chapter 5 presents all of the results. Chapter 6 presents the conclusions and Chapter 7 proposes future research directions.

Chapter 2

Prior Work

2.1 Introduction

In this chapter we discuss *design-for-testability* (DFT) techniques for testing digital circuits. The DFT techniques are required to improve the testability and reduce the test cost of the circuit. DFT techniques include *ad hoc* and structured techniques. *Test point insertion* (TPI) is a common *ad hoc* DFT technique to improve the circuit testability and scan design is the most widely used structured DFT methodology. Full-scan, partial-scan and *random-access scan* (RAS) design are the three most common scan architectures.

2.2 Full-Scan Design

2.2.1 Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic

Williams and Angell [74] introduced the concept of incorporating full-scan into a circuit for testing. They studied methods of using test points in conjunction with additional logic gates to provide an easy means to set or check the state of a sequential circuit. The logic added can switch from *normal* to *test* mode. In the *test* mode the flip-flops are configured to form a shift register, so that the state can be easily set or checked. After an initial state has been set, the circuit can be switched to *normal* mode for a test, then return to *test* mode so that the final state can be shifted out and checked. Williams and Angell provided formal cost analysis of modified LSI circuits indicating the range of circumstances under which adding test points and logic may be economically feasible. Simulation results reporting fault coverages, test set lengths or fault efficiencies, and hardware overhead results were not provided.

2.2.2 Random-Access Scan (RAS) Design

In RAS, the scan function is implemented like a *random-access memory* (RAM). A general architecture is given in Figure 2.2(a). All flip-flops form a RAM in scan mode. In general, a subset of flip-flops can be included in the RAM if partial-scan is desired. In the normal mode ($TC = 1$), all flip-flops receive data from the combinational logic under the control of the clock *CK*. Flip-flop outputs directly feed into combinational logic (see Figure 2.2(b)). The *scan flip-flop* (SFF) in Figure 2.2(b) is shown in Figure 2.1. In scan mode, this scheme allows reading or writing of any selected flip-flop. The flip-flop address, which may contain $\log_2 n_{ff}$ bits when there are n_{ff} flip-flops in the RAM, is serially loaded into an *address shift register* (ASR) using an address clock *ACK*. The address decoder now produces the select signal $SEL = 1$ for the addressed flip-flop. The *SEL* signals to all other flip-flops remain 0. The *SCANOUT* signals of all flip-flops are tied together to the *SCANOUT* pin. Thus, the content of the addressed flip-flop appears at this output. An advantage of this method is that any flop-flop can be observed even when the circuit is in the normal mode ($TC = 1$).

For scanning data into the flip-flop, the scan mode ($TC = 0$) is used. Assuming that the select signal ($SEL = 1$) has been generated for the addressed flip-flop, the state of *SD* is latched in that flip-flop by the clock *CK*. The *SD* inputs of all flip-flops in RAM are tied together to the *SCANIN* pin. However, the *SCANIN* signal is not latched in the unaddressed flip-flops because $SEL = 0$ inhibits the clock *CK*. The ability to control and observe individual flip-flops has the advantage of reducing the test sequence length, but has not gained popularity perhaps due to a large hardware overhead.

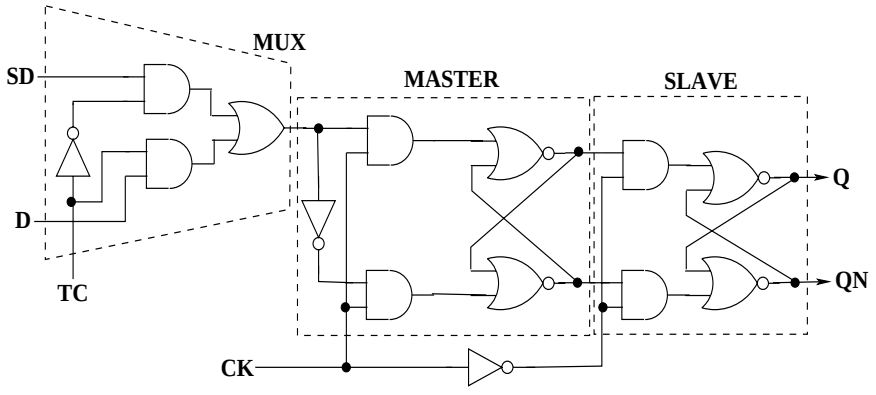

Figure 2.1: A single-clock *scan flip-flop* (SFF).

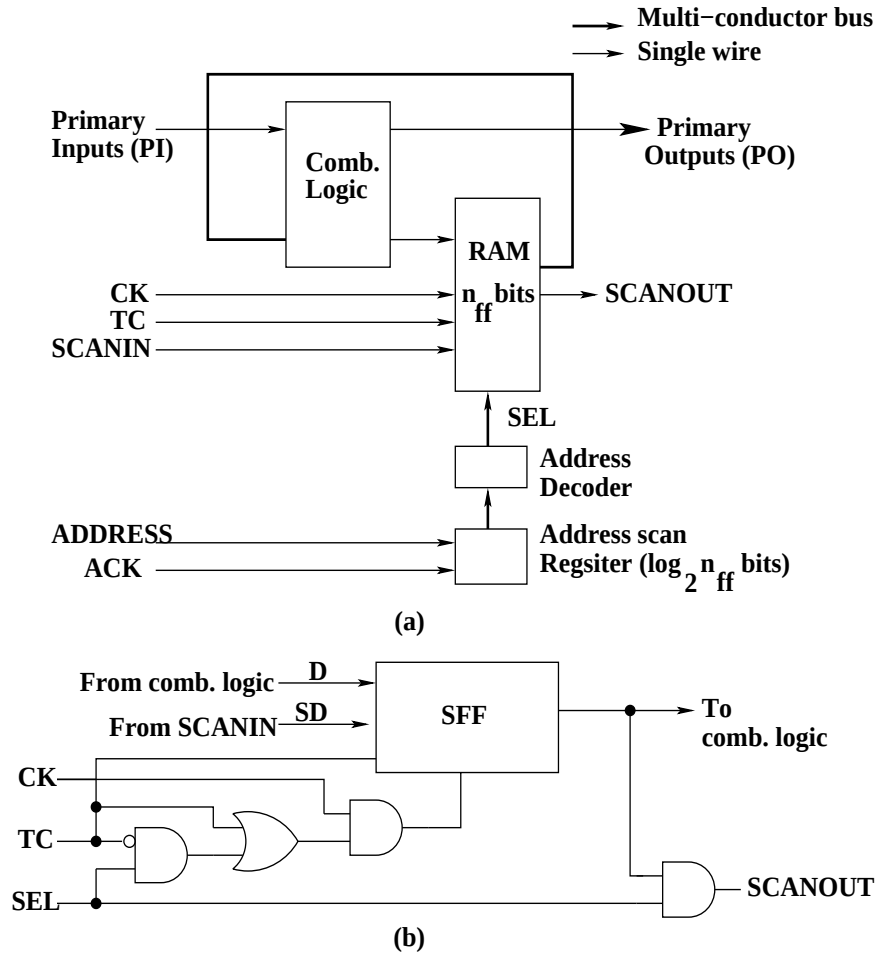

Figure 2.2: (a) Architecture of random-access scan. (b) *Random-access scan* (RAS).

2.3 Partial-Scan Design

2.3.1 Designing Circuits with Partial-Scan

Agrawal *et al.* presented methods to select a near optimal set of scan flip-flops [3]. The method tried to achieve a required test coverage with minimum overhead through the use of an automatic combinational test generator. The different techniques introduced for flip-flop selection are presented below.

2.3.1.1 Frequency Approach

This approach uses a modified *path-oriented decision making* (PODEM) test generation program [28]. Instead of generating just one test per fault, the modified version generates all possible tests for each single target fault in the combinational part of the circuit. In these tests, only inputs that are essential to detect the fault are set to 0 or 1, leaving all other inputs as "don't care." From these, tests are selected such that target faults are covered by a minimal set of flip-flops. Refer to Agrawal *et al.* [3] for the procedure.

Option Table:

Increasing the fault coverage and reducing the number of scan flip-flops are conflicting requirements, so an option table (Tables 2.1 and 2.2) is generated so the designer can make trade-offs. During generation of test vectors, the number of flip-flops used by each vector is recorded into a table. The objective is to select, for a given number of flip-flops, a set of tests (one per fault) that give the highest fault coverage (cover the largest number of faults).

Estimating Fault Coverage:

The estimated fault coverage is given by:

$$Estimated\ Coverage\ =\ \frac{F_{func}\ +\ f_{comb}\ +\ \alpha F_{ff}\ +\ \alpha N_{ff} f_{mux}}{F_{total}\ +\ \alpha N_{ff} f_{mux}} \tag{2.1}$$

where

F_{func} = the number of faults covered by functional vectors

F_{comb} = the number of combinational faults not detected by functional vectors

f_{comb} = the scan coverage of combinational faults $f_{comb} \subseteq F_{comb}$

F_{ff} = the number of flip-flop faults that functional vectors did not detect

α = the fraction of flip-flops in the scan register

N_{ff} = the total number of flip-flops in the circuit

f_{mux} = number of faults in the multiplexer of the scan flip-flop

F_{total} = total faults in the unscanned circuit, or $F_{func} + F_{comb} + F_{ff}$

They enhanced the PODEM test generator to generate all tests for a given fault. They also incorporated flip-slop selection and coverage estimation into the program to generate the option table. For test generation, the combinational part of the circuit is isolated by removing the flip-flops and setting the flip-flop I/O as the primary outputs and primary inputs of the combinational circuit. An option table for a four-bit multiplier is shown in Table 2.1. For the full table refer to Agrawal *et al.* [3].

Table 2.1: Option table for a four-bit multiplier – frequency approach.

Option	Target Fault Coverage (%)	Estimated Coverage (%)	Flip-Flops in Scan Chain	
			No.	%
1	97.06	99.84	15	100.00
2	79.41	98.75	14	93.33
3	67.65	97.95	13	86.67
4	61.76	97.46	12	80.00
⋮	⋮	⋮	⋮	⋮
8	38.24	95.18	7	46.67
⋮	⋮	⋮	⋮	⋮
10	23.53	93.64	4	26.67
11	20.59	93.23	3	20.00
12	17.65	92.80	2	13.33
13	5.88	91.80	1	6.67

2.3.1.2 Distance Approach

Unlike the frequency approach, the distance approach requires that only one test be generated per fault. The distance method uses an enhancement of the distance heuristic of PODEM. To generate tests in PODEM, a path is sensitized to propagate the state of the faulty line to a primary output. Path sensitizing is done by establishing a series of objectives. An objective is setting D or $\overline{D}$ at the fault site, propagating the D or $\overline{D}$ to a primary output by setting signal lines along the way and justifying the signal values by setting primary inputs to appropriate values.

PODEM sets the primary inputs that are nearest to the site of the objective. The distance is the number of gates on the path. In case of more than one *D*, the *D* closest to a primary output is propagated first. In the test generator, all normal primary inputs and outputs are assigned a distance 0. Inputs and outputs of combinational logic, derived from flip-flops, are assigned a large distance. Once a flip-flop signal is used in a test, its distance becomes 0. Table 2.2 shows results of applying this procedure to the four-bit multiplier circuit.

Table 2.2: Option table for a four-bit multiplier – distance approach.

Option	Target Fault Coverage (%)	Estimated Coverage (%)	Flip-Flops in Scan Chain	
			No.	%
1	100.00	99.69	15	100.00
2	50.00	97.01	15	100.00
3	41.18	96.36	14	93.33
4	35.29	95.86	13	86.67
5	17.65	94.86	12	80.00
6	11.76	93.46	7	46.67

Table 2.3: Partial-scan with a design goal of 95% fault coverage.

Circuit	Type of Scan	Flip-Flops in Scan Chain		Total Vectors	Fault Coverage (%)	Total Faults
		No.	%			
A	Full	42	100.00	3477	99.48	1336
	Partial (Frequency Option 3)	27	64.29	1665	97.35	1246
	Partial (Distance Option 6)	31	73.81	2285	98.89	1266
B	Full	172	100.00	17,568	98.59	5905
	Partial (Frequency Option 2)	110	63.95	9992	96.29	5531
	Partial (Distance Option 4)	111	64.53	10,308	96.22	5533
C	Full	351	100.00	92,889	98.75	12,222
	Partial (Frequency Option 8)	176	50.14	25,347	96.37	11,150
	Partial (Distance Option 12)	209	59.54	40,457	98.14	11,374

Overall experimental results showed that scanning 65% of the flip-flops in a four-bit multiplier helped attain a fault coverage of over 95%. Partial-scan takes advantage of better functional testability and produces low scan overhead, but full-scan does not take advantage of this testability. Partial-scan also helps enhance test generation.

Table 2.3 shows the final results for the partial-scan algorithm. Circuit A is a random logic block of a large chip. It is mildly sequential and contains deep combinational logic. Circuit B is an entire chip and is highly sequential (172 flip-flops). Circuit C is also an entire chip, but is much larger (351 flip-flops) than circuits A and B. Option tables were generated for each circuit and for each partial-scan method-frequency and distance. Using a design goal of 95% fault coverage, options were selected from these tables. On average, with partial-scan, about

40% of the flip-flops are excluded from the scan register. In general, the distance method tends to use more flip-flops. However, in circuits A and C, where more flip-flops are used, the final fault coverage is also higher. According to the experimental results on actual VLSl circuits, less than 65% of the flip-flops were included in the scan chain and yet they obtained a fault coverage higher than 95%. The advantage of partial-scan is that better functional testability produces low scan overhead. Full-scan does not take advantage of this testability.

2.3.2 BALLAST: A Methodology for Partial-Scan

Gupta *et al.* formulated a partial-scan methodology, which selects flip-flops such that the rest of the circuit belongs to a class of circuits called *balanced sequential structures* (B-structures) [32, 33].

B-Structures: Let S be an arbitrary synchronous sequential circuit with topology graph $G = (V, A, H, w)$. S is said to be a B-structure if:

1. G is acyclic;
2. $\forall v_1, v_2 \in V$, all directed paths (if any) from v_1 and v_2 are of equal length (including condition 1); and
3. $\forall h \in H$, if h is removed from G, the resulting graph is disconnected.

Circuit Model

A *kernel* is defined as a part of the circuit that is either connected to scan flip-flops or primary inputs at its inputs and is connected to either scan flip-flops or primary outputs at its outputs. A circuit can have more than one kernel. Moreover, BALLAST assumes that the set of flip-flops in the circuit can be partitioned into two subsets based on the presence or absence of explicit *load enable* controls. Load set L consists of flip-flops that have no explicit *load enable* control; these flip-flops always operate in the LOAD mode (in which data is read from the data input during every clock cycle). The hold set H is the set of flip-flops that have an explicit *load enable* control signal. These flip-flops can operate in HOLD mode (in which they retain their values across consecutive clock cycles) as well as in LOAD mode.

BALLAST selects a minimal number of scan flip-flops such that the remaining circuit is a B-structure. The outline of the BALLAST methodology is:

1. Construct *G*, the topology graph of the circuit.
2. Select a minimal cost set of arcs, *R*, to be removed from *G* such that the remaining topology graph is balanced. Arcs in *R* are the registers that must be scanned. Let S^B be the B-structure corresponding to the resulting topology graph which represents the kernel of the circuit.
3. Determine the combinational equivalent C^B of S^B. Using combinational test pattern generation techniques, compute a complete test set for C^B. This will be a complete single-pattern test vector set for the kernel S^B (refer to Gupta *et al.* for the proof [32]).
4. Construct a scan path containing the scan flip-flops in *R* so that they are capable of (a) shifting test patterns in/out, (b) holding a test pattern constant at the kernel inputs for *d* clock cycles (where *d* is the depth of the B-structure comprising the kernel), and (c) loading in the test results from the kernel.

Given a circuit designed in the above manner, the test plan for applying a sequence of *N* single-pattern test vectors to the circuit with *l* flip-flops on the scan path is as follows:

1. Operate all scan flip-flops in SHIFT mode for *l* clock cycles.
2. Repeat *N* times:
   (a) Place all scan flip-flops in the HOLD mode and all non-scan flip-flops in the LOAD mode for *d* clock cycles. (This allows test data to propagate through the circuit.)
   (b) Operate all scan registers in the LOAD mode for 1 clock cycle. (This loads the test result into the scan flip-flops.)
   (c) Operate all scan registers in the SHIFT mode for *l* clock cycles. (Scan out the test result and scan in the next test vector.)

In general the reduced length of the serial scan path in a BALLAST circuit leads to a lower test time over the corresponding full-scan circuit. However, two factors may increase the test time. First, a higher number of test vectors may be required for full fault coverage of the kernel because of the depth of its combinational equivalent circuit and fewer controllable inputs and

observable outputs of the kernel. Second, each test pattern needs to be held for *d* clock cycles in Step 2a above. Hence it is possible for the test time to increase in some circuits.

The BALLAST technique was applied to a sequential module MCOMP, 32 copies of which were used in a larger signal processing circuit VC. Each copy of MCOMP contained 14 flip-flops; hence with full-scan design, MCOMP would contribute a total of 448 flip-flops to the scan path of VC. The MCOMP circuit was reduced to a B-structure of depth 4 by selecting 8 scan flip-flops; thus the total contribution of MCOMP to the scan path of VC was reduced to 256 flip-flops. No fault coverage results were reported.

2.3.3 A Partial-Scan Method for Sequential Circuits with Feedback

Cheng and Agrawal presented graph-theoretic algorithms to select a minimal set of flip-flops for eliminating cycles and reducing sequential depth [17]. Cheng and Agrawal's theory stated that difficulty in generating tests for sequential circuits arises due to the circuit's cyclic structure, which in turn causes poor observability and controllability of flip-flops. They also discovered that *sequential depth*, which is defined as the largest number of flip-flops on paths between inputs and outputs, does not effect the complexity of test generation very much. They observed the performance of a sequential logic test generator that uses time-frame expansion of the circuit to generate tests with the *D*-algorithm. For an acyclic circuit (there is no feedback among sequential elements), the size of a complete test set is bounded by $D \cdot 2^n$, where *n* is the number of primary inputs and *D* is the sequential depth of the circuit. However, if the circuit is cyclic, it will take at most $2^L - 1$ vectors to set a node in the desired state. Thus, for maximum cycle length *L*, an upper bound on the length of a test sequence can be $D \cdot 2^L$.

They presented two sequential circuits, the first being a *traffic light controller* (TLC) and the second being CHIP-A, with identical sequential depths. But, the test generator produced better fault coverage results for CHIP-A. The explanation for this behavior by Cheng and Agrawal was *cycles*. Since the TLC had a larger number of complex cycles than CHIP-A, the test pattern generator produced inferior results for the TLC. The results are shown in Table 2.4.

Table 2.4: Test generation for two sequential circuits.

Circuit	No. of Gates	No. of FFs	Test Gen. CPU sec.	Fault Coverage (%)
TLC	355	21	1246	89.01
CHIP-A	1112	39	269	98.80

2.3.3.1 Flip-Flop Selection Algorithm

A directed graph is first constructed from the circuit netlist. Each node on the graph represents a flip-flip in the circuit and each edge between two nodes is combinational logic between the respective flip-flops represented by the nodes. All self-loops are removed. The remaining graph either contains no cycles or cycles longer than length 1. Since the problem of finding a *minimum feedback vertex set* (MFVS) that will break all the cycles is NP-complete [23, 26], heuristics were used to bound computation time [81]. The process of locating cycles in the graph is divided into a multi-phase task. In each phase, an upper bound is set on the time taken to locate cycles. The operations for the current phase are suspended after the time limit is reached. The list of cycles found should contain *representative cycles* such that no cycle is embedded in another cycle. The algorithm, *BreakLoop*, for selecting flip-flops to break all cycles is as follows:

```
BreakLoop() {
  While (cycle list not empty) {
    For every vertex
     Count the frequency of appearance
      in the cycle list.
    Select the most frequently used vertex.
    Remove all cycles containing the selected
     vertex from the cycle list.
  }/*While (cycle list not empty)*/
}/*BreakLoop()*/
```

Results for a few *ISCAS*-89 benchmarks are given in Table 2.5. Tests for the resulting circuit are efficiently generated by a sequential test pattern generator. An independent control of the scan clock allows insertion of scan sequences between the vectors produced by the test pattern generator. Cheng and Agrawal were able to achieve 98% fault coverage for a 5000 gate circuit by scanning 5% of the flip-flops. Moreover, scan selection is testability measure

Table 2.5: Test generation for *ISCAS*-89 benchmark circuits with partial-scan.

Circuit	Total No. of FFs	Scan FFs		No. of Test Vectors	Fault Coverage (%)			ATPG + Fsim sec. (VAX 86507)
		No.	%		Tested	Redundant	Total	
s400	21	9	42.86	107	98.11	1.89	100.00	7
s713	19	7	36.84	83	90.71	9.29	100.00	18
s5378	179	32	17.89	2612	93.38	6.32	99.70	1253
s9234	228	53	23.25	3458	43.23	55.80	99.04	6208

independent and the need for functional vectors is eliminated.

2.3.4 PASCANT: A Partial-Scan and Test Generation System

Bhawmik *et al.* based the selection of flip-flops on identification and elimination of *strongly connected components* (SCCs) in a circuit graph [8]. SCCs are eliminated by recursively deleting nodes as scan flip-flops until the entire circuit graph becomes acyclic. Deleting any node from a nontrivial SCC of a directed graph cuts at least one cycle. However, heuristics were presented to select flip-flops such that more than one cycle is eliminated. The following heuristics were presented:

1. Select the node having the largest connectivity (in degree + out degree) of all nodes.
2. Select the node having the largest MAX (in degree, out degree) of all nodes.
3. Select the node having the largest (in degree$\times$out degree) of all nodes.

The algorithm is give below:

Algorithm 1: Select nodes to cut all cycles of length > 1.

Input: The graph $G\ (V,\ E)$, where V is the set of nodes (flip-flops) and E is the set of edges (set of combinational paths between pairs of flip-flops) of the graph.

Output: Set P of nodes that cut all the cycles of G with length > 1.

Algorithm 1 () {

Step 1. In G find all nontrivial SCCs

$G_i\ =\ (V_i,\ E_i)$, where $1 \leq i \leq r$ and r is

the number of SCCs.

If $r\ =\ 0$, then return.

Step 2. For each nontrivial SCC $G_i\ =\ (V_i,\ E_i)$, where $1 \leq i \leq r$

(i) Select a node v in V_i using any one of the heuristics

mentioned above

(ii) Delete v from V_i and delete all incoming and

outgoing edges of v from E_i. Add v to P.

(iii) Let the resulting graph be $G_i'\ =\ (V_i',\ E_i')$,

where $1 \leq i \leq r$.

Execute Algorithm 1 with G_i' as input graph.

}/**Algorithm 1* ()*/

Step 1 of *Algorithm* 1 can be computed in linear time [5]. A second algorithm is also presented, which uses the heuristics *reduction of consecutive self-loop path length* and *dropping of nodes with high fanouts.* In the former heuristic, a *consecutive self-loop path* of length K is defined as a directed path of length K, where each vertex on the path has a self loop. The latter heuristic, $dropping\ of\ nodes\ with\ high\ fanout$, selects nodes with high fanouts. The threshold is $f \times m$, where f is a user-provided threshold and m is the maximum fanout of a node in the graph. The main conclusion of the paper is that cycle elimination is an effective technique

Table 2.6: Test generation for *ISCAS*-89 benchmark circuits with partial-scan.

Circuit	Selected / Total FFs	Fault Coverage (%)
s1423	31/74	89.28
s35932	657/1728	90.28
s38417	614/1636	94.52
s38584	728/1452	92.86

for selecting scan flip-flops. Also, the results reported (for partial results see Table 2.6) show that **minimal flip-flop selection is not always sufficient**. The complete table in the article [8] reports results for 21 *ISCAS*-89 benchmark circuits. An average of 37.6% of flip-flops are selected for scanning and an average fault coverage of 89.6% is reported.

2.3.5 An Exact Algorithm for Selecting Partial-Scan Flip-Flops

The main idea used by Chakradhar *et al.* [15] is to select partial-scan flip-flops that break feedback cycles in the s-graph. The s-graph for a sequential circuit is a graph where each node in the graph represents a flip-flop and the edges between any two nodes are the combinational logic between the corresponding flip-flops represented by the two nodes. Chakradhar *et al.* formulated it as a MFVS problem and solved the MFVS problem by branch-and-bound partitioning and pruning techniques based on an *integer linear programming* (ILP) formulation of the MFVS problem. Such a formulation can be used directly to solve the MFVS problem or to find the lower bounds on the cardinality of the MFVS. The lower bounds are useful for pruning the branch-and-bound search tree.

2.3.5.1 Graph Transformations

For any arc $(v_i \rightarrow v_j)$, v_i is a predecessor of v_j and v_j is a successor of v_i and operation *remove* (v_i) denotes the process of removing all incoming and outgoing arcs of vertex v_i.

Operation *ignore* (v_i) denotes connecting each predecessor of v_i to all its successors, using *remove* (v_i) and collapsing multiple arcs (if any) into a single arc. If v_j is a successor and a predecessor, a self-loop is created. The transformations given below are MFVS preserving [48]:

T1: If v_i has a self-loop then *remove* (v_i) and return v_i. Add v_i to the MFVS of the modified graph.

T2: If v_i has either in degree or out degree equal to 0, *remove* (v_i). Nothing is added to the MFVS.

T3: If v_i has either in degree or out degree equal to 1 and no self-loop, *remove* (v_i). Any MFVS of the modified graph is a MFVS for the s-graph.

The final graph resulting from the transformations is unique regardless of the order in which they are applied. The process of repeatedly applying **T1**, **T2**, and **T3** is referred to as *compress_graph*. The class of graphs that reduce to an empty graph after applying *compress_graph* is called the *two – way reducible* class.

Another MFVS preserving transformation is introduced that covers a class of graphs larger than the *two – way reducible* class. This can help reduce s-graphs that do not belong to a *two – way reducible* class. After the transformation, fewer vertices have to be considered for the branch-and-bound search. For a graph that is not two-way reducible, procedure *transform_graph()* is introduced:

```
transform_graph (S)
do{
   extract_scc (S);
   mfvs_list = mfvs_list + compress_graph (S);
  }while (S is modified and S is not empty)
return (S and mfvs_list);
```

Procedure *transform_graph* reduces the s-graphs that do not belong to the *two-way reducible* class to an empty graph. This class of graphs is called *scc-compressible*. If the s-graph of a sequential circuit is not scc-compressible then *transform_graph* will reduce it to a final graph containing one more SCCs. An SCC that cannot be further reduced by *compress_graph* is called a *compressed SCC*.

2.3.5.2 Partitioned Branch-and-Bound Search

The partitioning scheme is used when the procedures in the previous section cannot reduce the graph completely. A Boolean variable x_i is assigned to each vertex v_i in the graph. For any assignment of 0-1 values to the Boolean variables, we can construct a vertex set that includes only those vertices for which the corresponding Boolean variables assume the value 1. The 0-1 assignments that correspond to feedback vertex sets are called *feasible* solutions and the rest are infeasible solutions. The branch-and-bound procedure systematically searches the space of all 0-1 vectors for an optimum solution. The idea here is for a graph, which cannot be further reduced, partition the branch-and-bound branch into vertex clusters given by the reduced graph.

2.3.5.3 Pruning Strategies

The MFVS problem is formulated into an *integer linear programming* (ILP) problem [53]. The formulation is done as follows. Each vertex v_i has weight w_i associated with it. For any arc $(v_i \rightarrow v_j)$, it is required that $w_i - w_j \geq 1$. This implies that weights should decrease along any path. Clearly, this requirement cannot be satisfied for all arcs of a cycle. This infeasibility is used to find the MFVS. The cost function is $\sum x_i$ and it suffices to minimize this function. The graph transformations combined with the above ILP formulation give the exact algorithm for finding the MFVS for a given s-graph.

2.3.5.4 Results

Table 2.7 shows the number of flip-flops that were selected by different algorithms. Under columns *LR*, *Pa*, *Op*, and *PSCAN*, reported data is obtained by Lee and Reddy [44], from PASCANT by Bhawmik *et al.* [8], from Opus by Chickermane and Patel [19], and from the PSCAN algorithm (the exact algorithm proposed by Chakradhar *et al.*). The CPU times for the *LR* algorithm were not reported.

Table 2.7: Partial-scan results.

Circuit	FF	Scan Flip-Flops				CPU Sec.		
		LR	Pa	Op	PSCAN	Pa	Op	PSCAN
s953	29	5	5	5	5	0	0.1	0.1
s1196	18	0	0	0	0	0	0.4	0
s1238	18	0	0	0	0	0	0.4	0
s1423	74	21	37	22	21	0.3	1.0	0.9
s1494	6	5	5	5	5	0	0.5	0.1

2.3.6 Partial-Scan Design Based on Circuit State Information and Functional Analysis

Xiang and Patel presented a multi-phase partial-scan algorithm [78]. Partial-scan is divided into phases called *critical cycle breaking* and *partial flip-flop selection with respect to conflict resolution*. A partial-scan design method is introduced by combining circuit state information and conflict analysis. Critical cycles are broken using a combination of valid circuit state information and conflict analysis. Circuit state information is obtained via logic simulation; therefore, circuit state information is iteratively updated after a given number of partial-scan flip-flops have been selected. The valid-state-based testability measure may become ineffective to select scan flip-flops when cycles remaining in the circuit do not affect the testability measure. The method then resorts to an intensive conflict-analysis-based testability measure (*conflict*).

Definition 1:

A state for a fault-free circuit is an assignment of Boolean values $\{0,1\}$ to the output values of the flip-flops. The reset state is a state that can be reached from any state of the circuit.

Definition 2:

A state is called a *valid* state if it is reachable from the reset state; a state is called an *invalid* state if it is not reachable from the reset state. Let $v_1, v_2, \ldots, v_n \in \{0,1\}$, $i_1, i_2, \ldots, i_k \in \{1,2,\ldots,n\}$ and $k \leq n$. Assume that a *state* is an n-tuple, where n is the number of flip-flops in the circuit, and a *partial state* $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ is called a *partial invalid state* if the corresponding state is invalid. *State mapping* maps a state to all of the cycles in the circuit, where each cycle contains a subset of circuit flip-flops.

Definition 3:

The vertices of the directed s-graph of a sequential circuit are the flip-flops of the circuit. There is an edge (v_m, v_n) in the s-graph if there is a combinational path from flip-flop v_m to flip-flop v_n.

Definition 4:

The density of encoding of a circuit is defined as $\frac{V}{2^n}$, where n is the number of flip-flops in the circuit, and V is the number of valid states in the circuit. The number of valid states V for a circuit with n flip-flops is usually much less than 2^n. When $\frac{V}{2^n}$ is much less than 1, the test generator may frequently justify invalid states. An invalid state indicates that backtracks are needed.

Definition 5:

A *conflict* is defined as follows: A line l is assigned value v. In the previous process of test generation, l needs to be assigned value v'. If the intersection of v and v' produces a value in the logic system, line l is assigned value $v \cap v'$; otherwise, a conflict occurs on l.

2.3.6.1 The *Conflict* Measure

The $i-controllability$, C (i), of node l should reflect the potential number of conflicts (or possibility of causing conflicts) and the number of clock cycles required in order to justify a signal requirement (l,i), where $i \in \{\mathrm{X},0,1\}$ (i can be a don't care, logic 0, or logic 1). The *easiest fault effect propagation* (EFEP) path of a fault is the easiest path to propagate the fault effect on the node to a primary output. The $v-Observability$, $O_A(v)$ ($v \in \{D,\overline{D}\}$) reflects the number of conflicts (or possibility of causing conflicts) or the number of clock cycles required to propagate a fault effect v along the EFEP path. The EFEP path can be partitioned into stem segments, where a stem segment is the path segment between two fanout stems.

2.3.6.2 Valid State Analysis for Testability

The valid state sets of *ISCAS*-89 benchmarks were analyzed. Logic simulation was used to reach as many valid states as possible. The number of changes for a specific flip-flop is defined as the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions of two consecutive valid states according to the obtained state set. Here are the conclusions.

- If a circuit has many flip-flops with unchanging values, the circuit will have poor testability and vice versa.

- When a cycle assumes only one valid state, it may cause numerous backtracks during test generation. If all partial states in a cycle are valid, it does not influence test generation complexity significantly.
- When a circuit has many cycles taking only a few valid states, testability of the circuit is bad.

2.3.6.3 Multiple Phase Scan Flip-Flop Selection

Scan Flip-Flop Selection for Critical Cycle Breaking:

A new testability measure is calculated based on the obtained valid state information (Equation 2.2) and is used to select scan flip-flops.

$$T(f,c) = \begin{cases} \frac{2^k}{f(c)} & \text{if } k \leq 15 \\ k \cdot \frac{2^{15}}{f(c)} & \text{if } k > 15 \end{cases} \tag{2.2}$$

where 15 is an empirical constant, $f(c)$ is the number of different partial valid states of cycle c with respect to the obtained valid state set, and k is the number of flip-flops in cycle c. $T(f,c)$ should be smaller when cycle c has more partial valid states. The testability measure reflects the possibility of entering invalid states in the process of test generation. The *testability improvement potential* (TIP) measure for each flip-flop is used to evaluate the potential for testability improvement of the flip-flop if it is chosen to be a scan flip-flop. The TIP measure for each flip-flop f is given in Equation 2.3.

$$TIP(f) = \sum_c T(f,c) \cdot k \tag{2.3}$$

where c represents a cycle that contains the flip-flop f. The TIP measure is adopted to evaluate testability improvement potential of a flip-flop when it is scanned and scanning it breaks all cycles containing it. The summation of the TIP measure, controllability and observability measures of the $conflict$ measure is adopted to determine scan flip-flop candidates [80]. The following measure is adopted to select scan flip-flops:

$$T_1(f) = TIP(f) + \triangle T(f) \tag{2.4}$$

where $\triangle T(f)$ is the testability improvement potential corresponding to the $conflict$ testability measure if flip-flop f is scanned. $\triangle T(f)$ is the difference between the old and the new

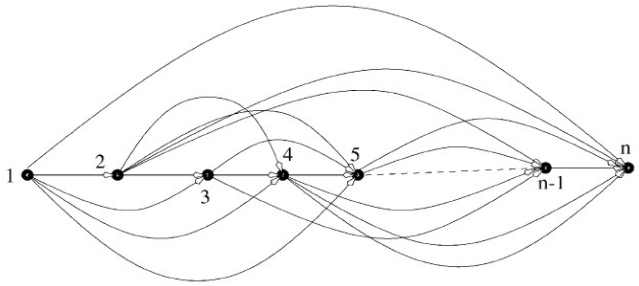

Figure 2.3: Signal interdependencies for an acyclic *n*-bit counter.

summation of all testability measures (1-controllability, 0-controllability, *D*-observability, and $\overline{D}$-observability) for the nodes with changed testability with respect to the $conflict$ measure. The following procedure is used to break critical cycles:

Critical-Cycle-Break()

1. Update valid state information of the current circuit via logic simulation.
2. Calculate the $conflict$ testability measure, TIP measure, and testability improvement potential with respect to the $conflict$ measure.
3. Select a scan flip-flop with respect to Equation 2.4; $sff \leftarrow sff + 1$. Update the TIP measure and the $conflict$ measure.
4. If $sff > limit_1$, $sff \leftarrow 0$; otherwise, go to Step 3. If the given number of scan flip-flops have been selected, end; otherwise, go to Step 1.

The valid state information is updated after a number of scan flip-flops have been selected. In the above procedure sff is the number of scan flip-flops selected in a phase and $limit_1$ is an empirical constant.

Scan Flip-Flop Selection Based on Conflict Resolution:

Procedure 'Critical-Cycle-Break()' presented in the last paragraph is not suitable for scan flip-flop selection after the critical cycles have been broken. It also will not work for acyclic circuits and may not work well for circuits with fewer cycles or circuits with cycles not affected by testability. Testability of some hard-to-test circuits is still bad even though all the cycles have been broken. In many cases, the cycle structures may not be the most important factor in test generation complexity. An *n*-bit counter is a good example. It is clear by looking at the s-graph in Figure 2.3 that there are no cycles, but this kind of structure makes test generation very difficult. In this case, summation of out-degree and in-degree can be used as a measure to select scan flip-flops. In-degree represents signal interdependencies of flip-flops during state

justification and out-degree shows the difficulty of fault propagation. Opscan [79] selected scan flip-flops based on the number of transitions after the critical cycles have been broken:

$$T(f) = \frac{\#T(0 \rightarrow 1) + \#T(1 \rightarrow 0)}{\#validstates} \tag{2.5}$$

where $\#T(0 \rightarrow 1)$ and $\#T(1 \rightarrow 0)$ represent the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions at the output of the flip-flop f based on the obtained valid state set. However, the measure in Equation 2.5 presents too much local information for testability improvement if the measure is not updated. Also, the measure does not include any observability information. In the method introduced by Xiang and Patel, the $conflict$ measure [80] is adopted to select scan flip-flops after the critical cycles have been broken. The method turns to the conflict resolution stage if one of the following situations occur:

- All cycles have been broken or the original circuit is acyclic;
- All remaining cycles in the circuit have all partial states valid;
- Only a very small ratio of cycles remain unbroken (10% for the larger circuits and 20% for the smaller circuits.)

Partial-scan flip-flops are selected based on the testability improvement potential of the $conflict$ measure. The following equation is utilized to get scan flip-flop candidates:

$$T = C_1(f) + C_0(f) + O_f(D) + O_f(\overline{D}) \tag{2.6}$$

where $C_1(f)$, $C_0(f)$, $O_f(D)$, and $O_f(\overline{D})$ are 0 and 1-controllabilities, and D and $\overline{D}$-observabilities with respect to the $conflict$ testability measure. Observability of a scanned flip-flop is reduced to 0 and its output's 1-controllability and 0-controllability measures are also reduced to 0. Observability (controllability) of lines immediately preceding (succeeding) the inserted flip-flop should be updated because an observation point is inserted into the flip-flop if it is scanned. Flip-flops with the most testability improvement potential are scanned.

Conflict-Resolution() (Scan flip-flop selection via conflict resolution):

1. Calculate the $conflict$ measure of the circuit.
2. Get the scan flip-flop candidate set with respect to Equation 2.6, the size of which is usually set at 2 or 3 times the given number of scan flip-flops.

3. Calculate the TIP of scan flip-flop candidates using the selective tracing procedure, as illustrated above. Select the flip-flop with the most testability improvement potential as a scan flip-flop. Update the *conflict* testability measures. Update the scan flip-flop candidate set.

4. If the given number of scan flip-flops has been selected, end the procedure; otherwise, go to Step 3.

Unlike the techniques in the critical cycle breaking stage, scan flip-flops during the conflict resolution stage are not selected iteratively. The *conflict* testability measures of the corresponding nodes are updated after a flip-flop is scanned. That is to say, the *conflict* measures of the circuit are calculated once and for all.

2.3.6.4 Results

A combination of the circuit-state-information-based testability measure TIP with the testability improvement potential of the *conflict* measure was utilized to select scan flip-flops. The TIP measure has been demonstrated to be a good testability measure to guide partial-scan flip-flop selection, which may become ineffective to select scan flip-flops when cycles in the circuit are not so influential to the circuit. Circuit state information is updated after a number of partial-scan flip-flops have been selected because many invalid states may become valid. The approach then turns to the *conflict* resolution stage, which selects scan flip-flops under the guidance of the conflict-analysis-based testability measure conflict. Partial experimental results are shown in Tables 2.8 and 2.9. In Tables 2.8 and 2.9 column 'sff' means number of scan flip-flops, column 'FC' is the fault coverage, column 'TE' reports the test efficiency, column 'vec' gives the vector length, and column 'cpu' is the CPU time in seconds.

Table 2.8: Comparison with previous partial-scan design methods.

Circuit	FF	mpscan				opus [18]				CoPs [54]			
		sff	FC	vec	cpu	sff	FC	vec	cpu	sff	FC	vec	cpu
s953	29	3	100	339	0.5	3	100	1831	2.0	3	99.9	263	0.3
s1423	74	32	97.1	638	5472	32	94.3	348	3441	32	97.1	423	660
s1488	6	2	100	639	4.5	3	100	2075	0.8	3	100	499	3.8
s1494	6	2	99.2	622	9.5	5	99.2	1853	0.7	3	99.2	505	4.0
s5378	179	50	97.2	1023	54.0	50	93.9	1018	1266	50	97.0	818	5645

Table 2.9: Comparison with more previous partial-scan design methods.

Circuit	FF	mpscan				zscan [62]			opscan [77]			sdscan [21]		
		sff	FC	TE	vec	sff	FC	TE	sff	FC	TE	sff	FC	TE
s953	29	3	100	100	339	3	100	100	3	100	100	3	99.8	-
s1423	74	22	96.17	98.22	567	-	-	-	-	-	-	30	87.0	-
s1488	6	2	100	100	639	2	99.9	100	2	100	100	2	99.4	-
s1494	6	2	99.1	100	622	2	99.1	100	2	99.1	100	3	99.2	-
s5378	179	50	97.2	100	1023	48	97.1	99.9	30	93.6	96.2	30	92.7	-

2.3.7 Exploiting Symbolic Techniques for Partial-Scan Flip-Flop Selection

Corno *et al.* propose a new testability measure based on the analysis of the circuit *state transition graph* (STG) through symbolic techniques [21]. They analyze how a scanned flip-flop affects the circuit behavior by working on the STG represented by a *binary decision diagram* (BDD) [12]. The testability measure is based on the notion of *state distribution* of a STG.

2.3.7.1 STG Testability Measure

State justification is a very time consuming operation: a significant amount of time is used by the ATPG algorithm while trying to identify an input sequence able to transfer the circuit to a desired state from which a propagation path exists. If the desired state is an invalid one, a significant amount of time is wasted to detect such a condition. Furthermore, in deeply sequential circuits, the amount of time required to justify a state can exceed the available time limit. In such a situation the ATPG produces a relatively high number of aborted faults. The justification process can be viewed as a search on the circuit STG to find a path from an initial state, either the unknown or the reset one, to the desired state. Through the use of scan, they control the value loaded into a flip-flop, so they are able to force a state transition by applying an input vector. Therefore, by scanning a flip-flop, they add some new transitions to the circuit STG and they obtain an *extended state transition graph* (E-STG) which, due to the increased connectivity, is easier to traverse and has a smaller sequential depth.

To select scan flip-flops, a new testability measure is presented to address both the number of reachable states and the circuit sequential depth. The *state distribution* of the STG is the average number of STG levels that must be explored to reach all reachable states. It is a real number defined as:

$$SD = \frac{\sum_l l \cdot RS_l}{\sum_l RS_l} \tag{2.7}$$

where RS_l is the number of states having minimum distance l from the initial state. Equation

2.7 is used to compute the reachable states from the reset state. The literal l measures the STG level currently explored, the distance being measured in clock cycles. The state distribution SD_g for the unmodified circuit is first computed, where no transformation is performed over its memory elements during the computation. In their testability analysis, they assume that a scanned flip-flop is fully controllable from a PI, and fully observable at a PO. They define SD_i as the state distribution of a circuit in which the flip-flop i is scanned. It is computed on the corresponding E-STG. Therefore, by comparing SD_g and SD_i they are able to evaluate the effect of a scan flip-flop insertion on the circuit behavior. The weight of the flip-flop i is thus defined as:

$$W_i = \frac{SD_g - SD_i}{SD_g} \tag{2.8}$$

This measure relates the behavior of a circuit where a single flip-flop is scanned with the behavior of the unmodified circuit. Scanning a flip-flop having a high value for W_i reduces the circuit sequential depth and increases the probability for the ATPG to reach all of the desired states.

2.3.7.2 Selection Algorithm

The testability measure defined so far assumes the availability of the circuit STG. A first feasible approach is to obtain the valid states of the machine by means of logic simulation. This approach is not exact because it depends both on the time spent for the simulation and on the input pattern distribution. As a result, a significant amount of simulation can be required to traverse all valid states of a large sequential circuit.

To apply the methodology, Equation 2.7 can be evaluated several times: once for the unmodified circuit and once for every E-STG that can be obtained from the original STG by transforming each flip-flop. This requires modification of the circuit, to rebuild the STG and to compute the required value. The operations were preformed using *binary decision diagrams* (BDDs) [12]. The circuit is modeled as a *finite state machine* (FSM), and is represented by the Boolean functions δ and *lambda*. Function δ computes the next state y from the current state s and the current input x : $y = \delta(s,x)$; function λ computes the output z starting from the same information: $z = \lambda(s,x)$. The techniques resort to the adoption of *characteristic functions* to represent sets of inputs $\chi_X(x)$, a set of states $\chi_S(s)$, the state transition relation $TR(s,y)$, and the output function $\chi_\lambda(z,x,y)$. The transition relation is defined as follows:

$$TR(s,y) = \exists x[\prod_j \delta_j(s,x)\overline{\oplus}y_j] \tag{2.9}$$

Algorithm 2.1 – Algorithm to compute the reachable states.

Input: A flip-flop.
Output: Reachable states of the circuit.

```
/* onset(c): size of the on-set of characteristic function */
compute_SD(i)
{
    TR = build_TR();              /* Equation 2.9 */
    if(i!=NULL)
       TR = ∃s_i ∃y_i(TR);        /* scan FF */
    l = 0;                        /* seq. depth */
    N_i = 0;                      /* sum in Equation 2.7 */
    end = FALSE;
    current = ResetState;
    reached = ResetState;
    do {
         new = ∃s(current · TR(s,y));
         current = new-reached;
        if(current == 0)
           end = TRUE;
        else
        {
           l++;
           N_i += l · onset(current);
           reached = reached + current;
        }
      } while(end!=TRUE);
    return(N_i/onset(reached));
}
```

It is true for every couple (s,y) for which an input x exists that satisfies $y = \delta(s,x)$, i.e., whenever y is a valid successor to s under some input value x. The algorithm has to compute the set of reachable states, thus it works on the TR function, only. We save memory space by not representing the λ function.

Algorithm 2.1 reports the algorithm used to compute the reachable states of the circuit. The circuit transformations can be easily performed by means of BDD operators. A scanned flip-flop i is functionally equivalent to a pair composed of a *primary input* (PI) and a *primary output* (PO). In terms of BDDs this can be computed by applying the existential quantification operator to the transition relation $TR(s,y)$ with respect to the variables s_i and y_i:

$$TR'(s,y) = \exists s_i \exists y_i (TR(s,y)) \tag{2.10}$$

Algorithm 2.2 – Algorithm to compute the flip-flop weights.

Input: Nothing.
Output: Flip-flop weights.

```
weight_FFs_to_scan()
{
    SDg = compute_SD(NULL);
    for (every FF i)
      {
        SDi = compute_SD(i);
        W[i] = (SDg-SDi)/SDg;
      }
    sort(W);
}
```

The paper presents heuristics to keep BDD and STG sizes manageable. The algorithm used to identify the best flip-flops to scan, described in Algorithm 2.2, performs its task in three steps: it computes the state distribution for the unmodified circuit; then it computes the state distributions of the STGs obtained by individually transforming each flip-flop into a *pseudo-primary input* (PPI); finally, it computes the flip-flop weights using Equation 2.8. The last operation performed is the sorting of the weight array in decreasing weight order. The first flip-flops in such a ranking, having the higher weights, are the ones best suited for scan.

2.3.7.3 Issues with Circuits

Circuit Partitioning:

Symbolic techniques are very efficient, but their applicability is limited to circuits with a few tens of flip-flops. In order to address larger circuits, the circuit is split into several macros and the BDD computation is applied to each macro separately. Every macro is a connected component, that is, a portion of the circuit where flip-flops are connected to each other through combinational logic, and is small enough to be handled by symbolic techniques.

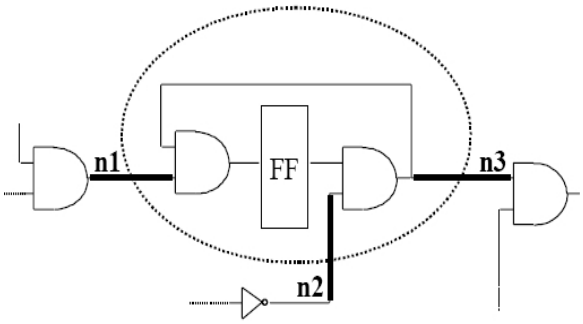

Figure 2.4: A macro example.

A macro is built in three steps: First a set of flip-flops is selected (they will become the memory elements of the macro), then the combinational logic to insert into the macro is chosen, and finally the macro inputs and outputs are identified. Figure 2.4 sketches a simple macro, which is identified by a single flip-flop. The combinational logic is computed by exploring the input and output cones of every memory element of the macro. All the combinational gates that both influence the macro flip-flops and are influenced by them are selected. In Figure 2.4 the imaginary macro boundary, depicted as a dashed line, crosses several nets. A net going from a gate outside such a boundary to a gate inside the macro is a macro input.

The memory elements are selected in such a way that all flip-flops in a macro are closely related, that is, the s-graph representing them must have at least one cycle. When all the STG partitions are not overlapping (each flip-flop appears at most in one macro), it is possible to have some invalid states included in the computations since interactions between macros are not properly taken into account. To improve the results of this approach they force the macro flip-flop selection algorithm to insert every flip-flop into at least two macros [21]. Therefore, the obtained subsets are partially overlapped. This ensures a lower probability of computing invalid states.

Flip-Flop Preselection by Structural Analysis:

At times trying to split the macros in smaller ones produces bad results, because too few flip-flops are held in the new macros. There is a high probability of obtaining a STG with few cycles, or none at all, for a macro holding four or fewer flip-flops, therefore the symbolic manipulations produce meaningless results when using such macros. The idea behind the approach described in this section is to use a structural analysis to identify which flip-flops are not well suited for scan insertion, to allow the symbolic procedure to work over a reduced set of flip-flops. The procedure is as follows:

1. Identify a set *S* of flip-flops having a known high impact on testability;
2. Apply a *depth first search* (DFS) ordering to *S* to extract the structural adjacency between flip-flops;
3. Split *S* into several macros using the approach described in the last paragraph;
4. Apply Algorithm 2.2 to the macros.

The rationale behind this procedure is to use the selection algorithm as an optimization tool over a subset of flip-flops that are known to heavily influence the circuit testability. Flip-flops that are nearly useless for scan insertion are identified by performing a testability analysis, for example by computing the SCOAP measurements [29]. The remaining flip-flops are analyzed to identify which are the ones best suited for scan insertion. By reducing the number of flip-flops to analyze, the problem becomes solvable by means of symbolic calculations.

2.3.7.4 Results

Table 2.10 shows results of some of the *ISCAS*-89 benchmarks for the prototype partial-scan algorithm SDSCAN proposed by Corno *et al*. The article [21] has a complete table with more results.

Table 2.10: OPUS *vs*. SDSCAN.

			SDSCAN			OPUS [18]	
Circuit	FFs	SFF	FC(%)	CPU(s)	M	FC(%)	CPU(s)
s953	29	3	99.8	4.40	2	99.9	3.23
s1196	18	3	100	5.38	1	99.8	6.08
s1423	74	30	87.0	429.62	18	78.6	755.88
s1488	6	2	99.4	54.57	1	100.0	38.60
s1494	6	3	99.2	25.13	1	99.2	28.55

2.3.8 More on Partial-Scan

A variety of partial-scan design methods have been developed to reduce the complexity of ATPG. Scan design methods draw much attention from industry and academia. They are usually classified into the following three categories: structure-based [6, 15–18, 33, 43, 44, 56, 67], testability-measure based [1, 9, 18, 38, 39, 41, 54, 56, 62, 71, 79], and test-generation-based methods [34, 46, 47, 55, 65].

Cheng and Agrawal [17] proposed a structure-based method in order to break cycles and reduce sequential depth for the first time. They pointed out that test generation complexity may be the exponent of the size of the cycles and linear with the sequential depth. Algorithms were presented to break cycles and reduce the sequential depth. Gupta and Breuer [33] presented a structure-based method that requires only combinational test generation and attains complete coverage of all detectable faults. Scan flip-flops are selected in such a way that the resulting kernel belongs to a so-called *balanced sequential structure* (B-structure), test generation of which can be treated as that of combinational logic. Chakradhar *et al.* [15] presented an exact algorithm to cut all cycles using the minimum number of scan flip-flops. In the method, partitioned searching and some effective pruning techniques were utilized to guide an effective search. Jiang *et al.* [37] proposed a novel method to reduce the length of the synchronizing sequences by scanning several flip-flops of sequential machines. Trischler [71] introduced a simple testability measure to select scan flip-flops, which is the first testability-analysis based partial-scan design method. Parikh and Abramovici [54] presented a partial-scan design based on a simple testability measure. The testability measure represents the number of clock cycles required to activate, propagate or detect a fault. Abramovici *et al.* [1] selected partial-scan flip-flops by untestability analysis. Many methods have been introduced to reduce test application time. Narayanan *et al.* [50] proposed an optimal *k*-scan chain configuration using dynamic programming, which can get an optimal solution in $O(k \cdot N^2)$ (N is the number of scan flip-flops).

In Section 2 we referred to a number of algorithms proposed to select scan flip-flops. After analyzing the quality of the results attained by these algorithms, the best results were reported by Xiang and Patel for their partial-scan algorithm, *mpscan* [78]. Chakradhar and Agrawal [15] solve the MFVS problem exactly. The fault coverages obtained for benchmark circuits by the algorithm were not reported.

2.4 *Test Point Insertion* (TPI)

In pseudo-random testing, the fault coverage is limited by the presence of *random-pattern resistant* (RP-resistant) faults. If the fault coverage is not sufficient, than TPI can be applied to enhance the fault coverage. TPI can also help in lowering the test generation time of hard-to-detect faults. Control points and observation points are two typical types of test points that can be used to boost the circuit's fault coverage. Note that it is not sufficient to only use observation points because some faults require control points in order to get detected. Optimal placement of test points in circuits with reconvergent fanouts has been shown to be NP-complete [42]. Several approximate techniques for test point placement have been developed. The techniques can be categorized depending on whether the technique uses fault simulation or testability measures to place the test points.

For fault simulation techniques, the set of patterns generated by the test generator is used to fault simulate the circuit to identify the undetected faults. Test points are then inserted so that the undetected faults are detected. Iyenger *et al.* use fault fault simulation to locate gates that block faults and insert test points at these gates to allow propagation [36]. Touba and McCluskey use path tracing to identify a set of test point solutions for each undetected fault and a covering algorithm that gives the minimum number of test points to detect these undetected faults [70]. The limitation of fault simulation-based techniques is that they require test patterns ahead of time. Also, a late change in the design can change the test patterns and annul the test point analysis.

Testability measure-based techniques avoid these problems because they do not require any ATPG knowledge. The testability measures approximate the detection probability of RP-resistant faults. Seiss *et al.* [63] form a cost function based on the *controllability/observability program* (COP) [11] testability measures and then compute, in linear time, the gradient of the function with respect to each possible test point. The gradients are used to approximate the global testability impact for inserting each test point. The test point with the maximum benefit is inserted and COP measures are recomputed. The process continues iteratively until the testability of the circuit is satisfactory. Tamarapalli and Rajski use probabilistic fault simulation, which provides a more accurate measure than the COP testability measures [68].

Boubezari *et al.* compute the testability measures at the *register-transfer level* (RTL) allowing RTL synthesis to take the test points into consideration when optimizing the design [10].

Xiang *et al.* insert observation points in the scan chains and multiple capture cycles are used during the shift operation [76].

Sethuram *et al.* proposed a test point insertion technique for structured *application specific integrated chips* (ASICs) to reduce test length and ATPG run times [64]. A new testability measure, called the gain function, is used to find the best candidate lines for test point insertion. The gain function is used to quantify the reduction in test volume and test application time. The gain function for a signal line computes the total number of additional don't cares (X's) generated when an test point is inserted on a line. A candidate line with a high gain value is a good candidate for test point insertion.

Timing-driven test point insertion techniques have been developed to address the problem of adding test points on a critical timing path, causing the circuit to fail the timing requirements. Tsai *et al.* compute the timing slack of each node and eliminate any node whose slack is not sufficiently long as a test point candidate [72]. As the test points are inserted, the slack information is updated. The number of test points needed to achieve sufficient fault coverage may increase because test points cannot be inserted in some locations due to the timing constraints.

2.5 Testability Measures and Analysis

An overwhelming variety of testability measures have been introduced to aid circuit design and test. Testability measures have been using structural, functional, or simulation information [7, 29, 58, 66, 69]. A combination of the information can be used as well. Information theoretic measures at the *register-transfer-level* (RTL) have also been proposed for power and testability estimation [25, 49, 69].

2.6 Information Theory and Testing

Concepts of information theory have proved their usefulness in combating noise-related errors of communication. Information theoretic measures such as entropy also apply to other scientific fields such as physics (statistical mechanics), mathematics (probability theory), electrical engineering (communication theory), and computer science (algorithmic complexity). Equation 2.11 shows the general definition of entropy for a *random variable* (RV) X with a *probability mass function* (PMF) $p(x)$. A PMF is the discrete version of a *probability density function*

(PDF). Variable x varies over all possibilities of X.

$$H(X) = -\sum_{x \in X} p(x)\ \log_2(p(x)) \tag{2.11}$$

Entropy can also be viewed as the number of bits (information), on average, required to encode a RV. Example 1 emphasizes this point.

Example 1:

Consider a RV X that has a uniform distribution and has 32 possible outcomes. The entropy of X is:

$$H(X) = -\sum_{x=1}^{32} p_x\ \log_2(p_x) = -32 \times (\frac{1}{32}\log_2(\frac{1}{32})) = 5\ bits \tag{2.12}$$

which agrees with the number of bits needed to describe X.

For logic circuits, each wire (gate inputs and outputs) is a RV with 2 possible outcomes – logic 0 and logic 1. Entropy of a wire is:

$$H(X) = -((1-p)\log_2(1-p) + p\ \log_2(p)) \tag{2.13}$$

where p is the probability of logic 1 occurring at a line. Equation 2.13 is plotted in Figure 2.5 with $H(X = p)$ on the ordinate and p on the abscissa. Maximum entropy in Figure 2.5 occurs when 0 and 1 are equally likely.

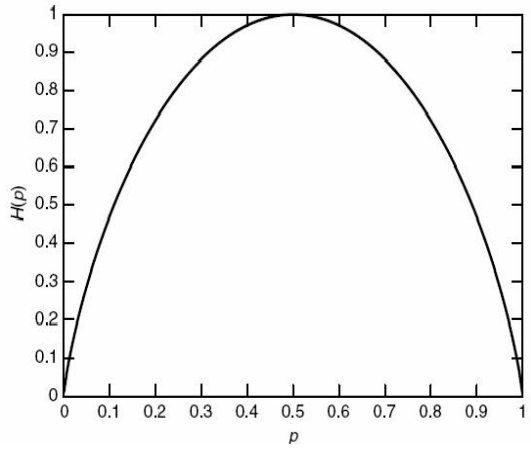

Figure 2.5: $H(p)$ vs. p.

Dussault proposed the first information theoretic testability measure [25]. *Mutual information* (MI) in Equation 2.14 is a measure of dependence between two random variables. MI gives the reduction in the uncertainty of X due to knowledge of Y and MI is 0 if X and Y are independent RVs. $H(X|Y)$ in Equation 2.15 is the information about X given that Y is known [22].

For binary logic, $X \in (0,1)$ and $Y \in (0,1)$. Equations 2.15 – 2.17 show the proposed testability measures. Inputs of the circuits are represented by the vector RV X and outputs by the vector RV Y.

$$I(X;Y) = \sum_{\substack{x \in X \\ y \in Y}} p(x,y) log_2 \frac{p(x,y)}{p(x)p(y)} = I(Y;X) \tag{2.14}$$

$$Observability = \frac{1}{H(X|Y)} \tag{2.15}$$

$$Controllability = \frac{1}{H(Y|X)} \tag{2.16}$$

$$Testability = I(X;Y) \tag{2.17}$$

$H(X|Y)$ gives the uncertainty in inputs X given that outputs Y are known. The lesser the uncertainty, the larger the amount of information about inputs reaching the outputs, implying higher observability. Equation 2.15 gives the observability since observability and uncertainty $H(X|Y)$ are inversely proportional. Equation 2.16 can be reasoned similarly. Overall testability of a circuit is given by MI of X and Y. Large MI values indicate high testability of the circuit.

Agrawal proposed an information theoretic approach to testing digital circuits [2]. Agrawal derived the probability $P(T)$ of detecting a stuck-at fault by a vector sequence T as:

$$P(T) = 1 - 2^{-\frac{H_o T}{k}} \tag{2.18}$$

where k is the number of lines through a circuit partition where the detectable fault exists and H_o is the entropy at the output of the circuit. Consider a 2-input AND gate with inputs $i1$, $i2$, and output Z. The probability of logic 0 (logic 1) at Z is 0.75 (0.25) and the entropy Z is:

$$H_Z = -0.25 \log_2(0.25) - 0.75 \log_2(0.75) = 0.811 \tag{2.19}$$

Assume that the probability of logic 0 (logic 1) occurring at the inputs is 0.5 (0.5). The entropy at the first input, $i1$, is:

$$H_{i1} = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1.0 \tag{2.20}$$

Entropy for the second input, $i2$, is calculated identically. The total information present at the inputs is $1+1=2$. There is an information loss of $2.0 - 0.811 = 1.189$. Agrawal proposed an ATPG that reduces the loss (by increasing entropy or information) of information and maximizes $P(T)$ in the circuit by adjusting the probabilities of 0 and 1 at the inputs of the circuit.

Thearling and Abraham proposed information theory based testability measures at the functional level [69]. They also proposed partitioning of the circuit, to improve testability, using entropy related measures.

2.7 *Automatic Test Pattern Generation* (ATPG)

2.7.1 Simulation-Based ATPG

In this section we will discuss we discuss a simulation-based ATPG, *genetic-algorithm-based* ATPG (GATEST), since we use it for test generation in this dissertation [61]. GATEST is a three-phased sequential test generator based on *genetic algorithms* (GAs) and uses the PROOFS sequential fault simulator [52]. In GAs a population of individuals is defined, where each individual is a candidate solution for the problem at hand. In GATEST, the populations of candidate tests are evolved by the GA starting from a random initial population, and the best test evolved is added to the test set in a given time frame. A highly accurate fitness function is used to evaluate candidate tests in order to achieve good quality test sets. Good results were obtained for combinational circuits, i.e., high fault coverages and compact test sets, although better results can be obtained with deterministic algorithms. Results for the *ISCAS*-89 sequential benchmark circuits indicate that the selection and crossover schemes used have a significant impact on fault coverage. The best results were obtained for tournament selection without replacement and uniform crossover. Variations in the mutation rate had a much smaller effect on fault coverage, and binary codings tended to give higher fault coverages when small population sizes of 16 or 32 were used. Non-overlapping populations gave the highest fault coverages, but an average speed-up of 1.3 was obtained by using overlapping populations, with only a 0.4% drop in fault coverage. More significant reductions in execution time were obtained by using small fault samples in the fitness evaluation. While GATEST often gives better results as compared to deterministic approaches, limitations do exist. In a simulation-based approach such as the one described here, untestable faults cannot be identified. Furthermore, GATEST is best suited for data-dominant circuits, and higher fault coverages are obtained for highly sequential circuits using a deterministic test generator.

High fault coverages and compact test sets have been obtained with GATEST for combinational and sequential circuits. In some cases, deterministic ATPGs achieved higher fault coverage in much less time. For sequential circuits, the number of faults detected is either

greater than or equal to that of deterministic test generators for most circuits, and the test sizes are much shorter [61]. In most cases, GATEST takes only a fraction of the execution time compared to deterministic test generators. Thus, GATEST can outperform deterministic test generators in terms of fault coverage and test length.

2.8 Summary

In this chapter, Section 2.2 presented the full-scan approach to DFT. Section 2.3 presented a variety of partial-scan algorithms. Section 2.4 presented test point insertion techniques. Section 2.5 presented a summary of different testability measures. Section 2.6 introduces the information theory concepts and its applications to testing. Section 2.7 discussed GATEST, a simulation-based ATPG.

Chapter 3

The SPARTAN Partial-Scan Algorithm

3.1 Introduction

This greedy algorithm analyzes the *circuit-under-test* (CUT) and selects *scan flip-flops* (SFFs) using three measures: spectral analysis of the flip-flop oscillations, entropy from information theory, and logic gate circuit level information to measure observability and controllability. A high quality SFF set will enhance the testability of the CUT, which allows the *automatic test-pattern generator* (ATPG) to attain high fault coverage with a shorter test set. We use the spectral analysis and entropy combination because spectral analysis incorporates only functional information and does not use any structural information of the CUT [40].

3.1.1 Spectral Analysis

3.1.1.1 Data Structure

SPARTAN uses an s-graph [75] and logic simulation for SFF selection. In a sequential circuit, an s-graph is the logic circuit graph where each node represents a flip-flop and an edge between two nodes is a path through combinational logic between the flip-flops. When the s-graph is condensed, each node of a condensed s-graph is called an *strongly connected component* (SCC).

3.1.1.2 Spectral Algorithm

The spectral algorithm dissects the function and structure of sequential circuit states in the spectral domain. The *spectral correlation coefficients* (SCs) for each flip-flop in the CUT, which are coefficients of the flip-flop states in the spectral domain, are used as a controllability measure. A low toggling rate flip-flop will have small SC values and thus is difficult to control. If

the average value of the SCs of a flip-flop is low (low toggling), the flip-flop has low controllability and will be a good SFF candidate (see Example 2). A low toggling flip-flop has bad controllability because regardless of which vectors are used, the ATPG will have to apply many vectors to set the flip-flop to a desired state in order to observe or sensitize faults. To compute observability, we first find for each flip-flop the *primary outputs* (POs) in its fanout cone. We then compute the SCs of all POs. Then we perform a SC comparison of each flip-flop with each PO in its fanout cone. If the SCs of a flip-flop are identical to SCs of any one of the POs in its fanout cone, we consider that flip-flop observable.

We employ an order 4 *Rademacher-Walsh transform* (RWT) (an order 3 RWT is shown in Equation 3.1) for analyzing states of flip-flops and POs. First, the state machine is initialized to a randomly-chosen state. Flip-flop states and PO logic values are obtained via logic simulation with 512 random vectors [35]. Table 3.1 shows the fault coverage and test length results for s9234 and s38584 with partial-scan. The results were generated for various random vector lengths and RWT orders used for spectral analysis. Varying the initial random vector length (*SPEC_NUM_RAND_VEC*) and RWT order (*MATRIXSZ*) has minimal effect on the fault converge for s9234, but the best test length is obtained when *SPEC_NUM_RAND_VEC* is set to 512. Similarly, for s38584, *SPEC_NUM_RAND_VEC* has little effect on fault coverage results. But, the test length is shorter when *SPEC_NUM_RAND_VEC* is set to 100K. We used 512 random vectors for spectral analysis because it gets a slightly higher fault coverage and the CPU time for spectral analysis is shorter compared to 100K random vectors. For s9234, the 16×16 (*MATRIXSZ*=4) RWT performed better than the 256×256 (*MATRIXSZ*=8) RWT because the frequencies represented by the basis rows of the 16×16 RWT represented the characteristic frequencies of s9234 better than the 256×256 transform. For s9234, 512 random vectors performed better than 50K and 100K random vectors possibly because the circuit is initialized to a random state before the logic simulation with the random vectors is performed and the initial state for the 512 random vector simulation projected the circuit into more productive parts of the state space than for the 50K and 100K simulations. Also, the vectors used for logic simulation are random as well and it is likely that the 512 vectors drove the circuit into parts of the state space the other vector sets were unable to explore. For s38584, the number of initial vectors did not have a profound impact on the fault coverage results but 100K random vectors had a lower test length. In contrast to s9234, s38584 gets a better result for the larger initial random vector set. The conclusion that can be drawn from this is that large random vector sets

are more beneficial for large benchmarks because they have vast state spaces and large random vector sets help explore the space. On the other hand, large random vector sets may not help the small benchmarks because the state space of a smaller circuit can be efficiently explored with a smaller test vector set as well. Finding the right size for the initial random vector set is an empirical process. A more deterministic approach would be to use ATPG test vectors for efficient state space exploration.

$$RWT(3) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad (3.1)$$

Table 3.1: Comparison of fault coverage and test length results by varying the number of random vectors for spectral analysis and size of the RWT matrix for SPARTAN partial-scan using GATEST.

Ckt.	*SPEC_NUM_RAND_VEC*	*MATRIXSZ*	GATEST Parameters			SFF	# Iterations (for SFF selection)	FC (%)	TL
			-i	-v	-g				
s9234	512	4	20	500	8	199	3	93.32	958
	50K	4	20	500	8	211	2	92.85	1855
	100K	4	20	500	8	207	2	93.01	2543
	50K	8	20	500	8	213	2	92.51	2923
	100K	8	20	500	8	211	2	92.70	2422
s38584	512	4	200	18	16	924	3	94.54	1491
	50K	4	200	12	16	1319	2	95.50	698
	100K	4	200	7	16	1029	1	94.07	1036

Spectral Correlation Coefficients

The *spectral correlation coefficients*, for each flip-flop, are obtained by dividing the flip-flop state over the last n clock periods into chunks (the state vector is divided into smaller column vectors), $\mathbf{Z}$, and pre-multiplying each chunk with the RWT of appropriate dimension (see

Equation 3.2).

$$RWT(3) \times \left[\begin{array}{c} \mathbf{Z} \end{array} \right] \quad = \quad \left[\begin{array}{c} \mathbf{S} \end{array} \right] \tag{3.2}$$

S is the SC vector. Please note that *spectral correlation coefficient* and SC will be used interchangeably. The chunk size for the flip-flop states depends on the dimension of the RWT used. We use a 16×16 matrix for our analysis. Matrix dimensions up to 512×512 were tried, but larger matrices had inconsequential effect on the quality of the SFF set. Also, it takes less time to perform spectral analysis using a 16×16 transform. Since the transform entries $\in \{+1, -1\}$, we have to translate the flip-flop states to $+1$ or -1. We translate $0 \rightarrow -1$ and $1 \rightarrow +1$ because when we calculate the spectral coefficients (a SC is the number of agreements minus the number of disagreements with the row elements of the transform matrix), logic 0 and logic 1 should be given equal and opposite magnitude. This ensures that both logic 0 and logic 1 are weighted equally and provide a contribution in the SCs. For example, if we have a vector with logic 0's and logic 1's and if we extract spectra (using any matrix transform) from this vector, the logic 0's in the vector will have no contribution in the SCs. Figure 3.1 shows the wave representation of the order 3 (8×8) RWT. When a bit stream is analyzed using an order 3 transform, each coefficient tells us how well the bit stream correlates with each wave in Figure 3.1 (or row of the matrix in Equation 3.1).

EXAMPLE 1 **–** In this example we will use the 8×8 RWT for analysis. To compute the first SC (s_0), row 1 of the 8×8 RWT is first multiplied with column vector **Z**, comprised of the flip-flop state over the last 8 clock periods, to obtain the SC with first 8 states of the flip-flop. This 8 bits of state are then multiplied by rows 2 through 8 of $RWT(3)$ to obtain the remaining 7 SCs, $s_1 - s_7$, with the RWT basis vectors. $RWT(3)$ is then multiplied with bits 2 to 9, bits 3 to 10, and so on of the flip-flop states. The partial SCs of the flip-flop obtained, after each row of the RWT is multiplied with the state over the last 8 clock periods, are aggregated and normalized by the number of times multiplication was performed by row 1 to obtain the first SC s_0. Example 2 elaborates the spectral analysis procedure. Multiplication of a row with a chunk **Z** is analogous to correlation where the data vectors being correlated slide over each other. Here the data vectors are the first row of the RWT and a state chunk **Z** (see Equation 3.2).

EXAMPLE 2 **–** Let us use the $RWT(3)$ in Equation 3.1 for spectral analysis of a

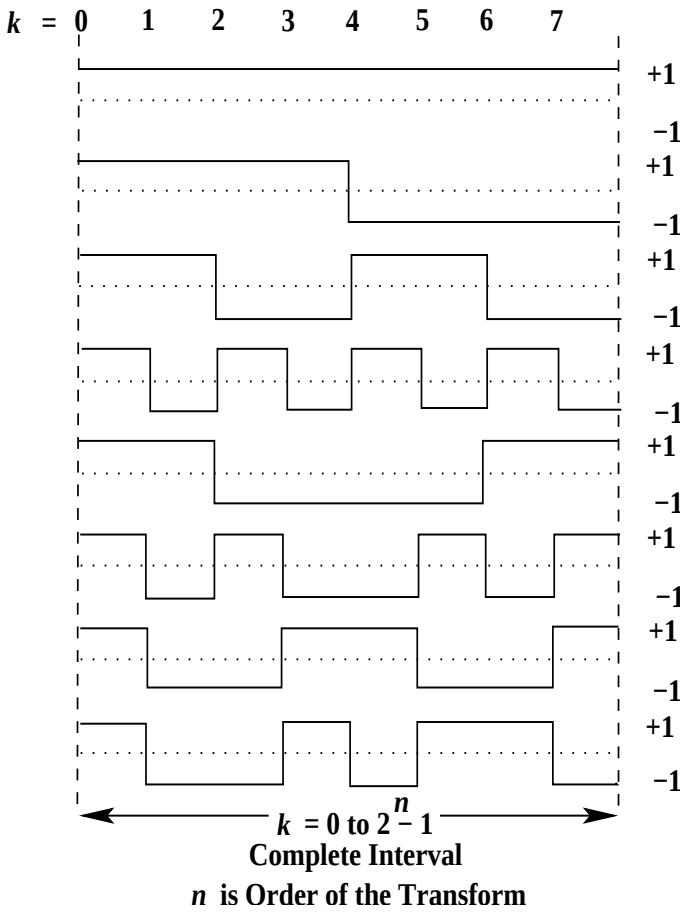

Figure 3.1: The orthogonal Rademacher-Walsh functions for $n = 3$.

flip-flop F. After ten random vectors at PIs of the CUT, F transitions through states $V = [1, 0, 1, 1, 0, 1, 1, 0, 0, 0]$. After translation of V, we get $V' = [1, -1, 1, 1, -1, 1, 1, -1, -1, -1]$. To compute s_0, we multiply row 1 of the RWT with translated bits 1-8, 2-9, and 3-10 of V'. For bits 1-8 the partial SC is $(1\times1)+(1\times-1)+(1\times1)+(1\times1)+(1\times-1)+(1\times1)+(1\times1)+(1\times-1)=2$. For bits 2-9 the partial SC is $(1\times-1)+(1\times1)+(1\times1)+(1\times-1)+(1\times1)+(1\times1)+(1\times-1)+(1\times-1)=0$ and bits 3-10 give us $(1\times1)+(1\times1)+(1\times-1)+(1\times1)+(1\times1)+(1\times-1)+(1\times-1)+(1\times-1)=0$. The aggregate of partial sums is $2+0+0=2$ and after normalization we get $s_0=\frac{2}{3}$. Normalization is done with 3 because there were 3 multiplication operations on row 1 and three 8-bit chunks (**Z**) of V'. This flip-flop will have 8 SCs. Similarly, SCs are computed for other flip-flops in the CUT. Procedure 3.1 shows the spectral analysis and Figure 3.2 presents the flow chart for the spectral analysis.

Procedure 3.1 – Spectral Analysis:

1. Logic simulate the CUT with 512 (see Table 5.2) random vectors to obtain valid FF states and PO values.
2. Compute SCs of all FFs and POs.
3. Set $SC_{avg\_MAX}=0$ and $SC_{avg\_MIN}=0$.
4. For each flip-flop:
   (a) Compute the average of SCs, SC_{i_avg}, for flip-flop i with $k=16$ using Equation 3.3. In Equation 3.3, w_j is a normalizing coefficient and it assigns lower weights to the low sequency FFs and higher weights to the high sequency FFs, which directly correlates to the controllability of the SFF. Equation 3.4 is used to compute the value for w_j where φ_j is the *sequency* of the j^{th} transform matrix row. The sequency, which is analogous to frequency, of a row is the number of $1\rightarrow-1$ and $-1\rightarrow1$ transitions in that row. For example, in Equation 3.1, the sequency of the first row of the matrix is zero, the sequency of the second row is one and so on.

$$SC_{i\_avg} = \frac{\sum_{j=1}^{k} w_j |SC_j|}{k} \tag{3.3}$$

$$w_j = \frac{(\varphi_j+1)}{\sum_{h=1}^{k}(\varphi_h+1)} \tag{3.4}$$

(b) If $SC_{i\_avg} \geq SC_{avg\_MAX} \Rightarrow SC_{avg\_MAX} = SC_{i\_avg}$.

(c) If $SC_{i\_avg} \leq SC_{avg\_MIN} \Rightarrow SC_{avg\_MIN} = SC_{i\_avg}$.

(d) Compare SCs of FF *i* with SCs of POs its fanout cone.

i. If SCs of FF $i ==$ SCs of any fanout cone PO $\Rightarrow$ $Good\_Observability(i) = TRUE$.

5. Compute the *SC Threshold* using Equation 3.5.

$$SC\ Threshold \quad = \quad \text{average}(\ SC_{avg\_MIN}, SC_{avg\_MAX}) \tag{3.5}$$

6. For each flip-flop:

(a) If $SC_{i\_avg} < SC\ Threshold$ && $Good\_Observability(i) == FALSE \Rightarrow$ $scan\_mark(FF(i)) = TRUE$ (i.e., the flip-flop will be recommended for scanning). □

After Procedure 3.1 (Figure 3.2), spectral analysis gives a list of candidate flip-flops recommended for scanning. We will scan only a subset of this list. If a flip-flop only has paths to it from other flip-flops, we automatically scan it. Next, we set the *scanning threshold* (ST) to 25% of the total FFs in the circuit. This gives us the size of the subset (of the initial candidate FF list from Procedure 3.1) that will eventaully be scanned. ST can be varied and 25% gave the best results. Once ST is set, we start scanning FFs from the list of FFs marked as candidates for scanning by Procedure 3.1, starting with FFs belonging to the largest SCC because it will help us break bigger cycles in the circuit. Once all of the marked FFs in a SCC are scanned, the SCC is marked as 'visited' and cannot be revisited. If the number of scanned FFs is less than ST, we move to subsequent SCCs of the circuit. The procedure is shown Procedure 3.1a (see Figure 3.3).

Procedure 3.1a – SFF Selection after Spectral Analysis:

1. Number of SFFs, $numScanFF = 0$.

2. Largest SCC size, $maxSCCSz = 0$.

3. Each SCC's scan flag, $SCC(i)\_scanned = FALSE$.

4. while $numScanFF < ST$

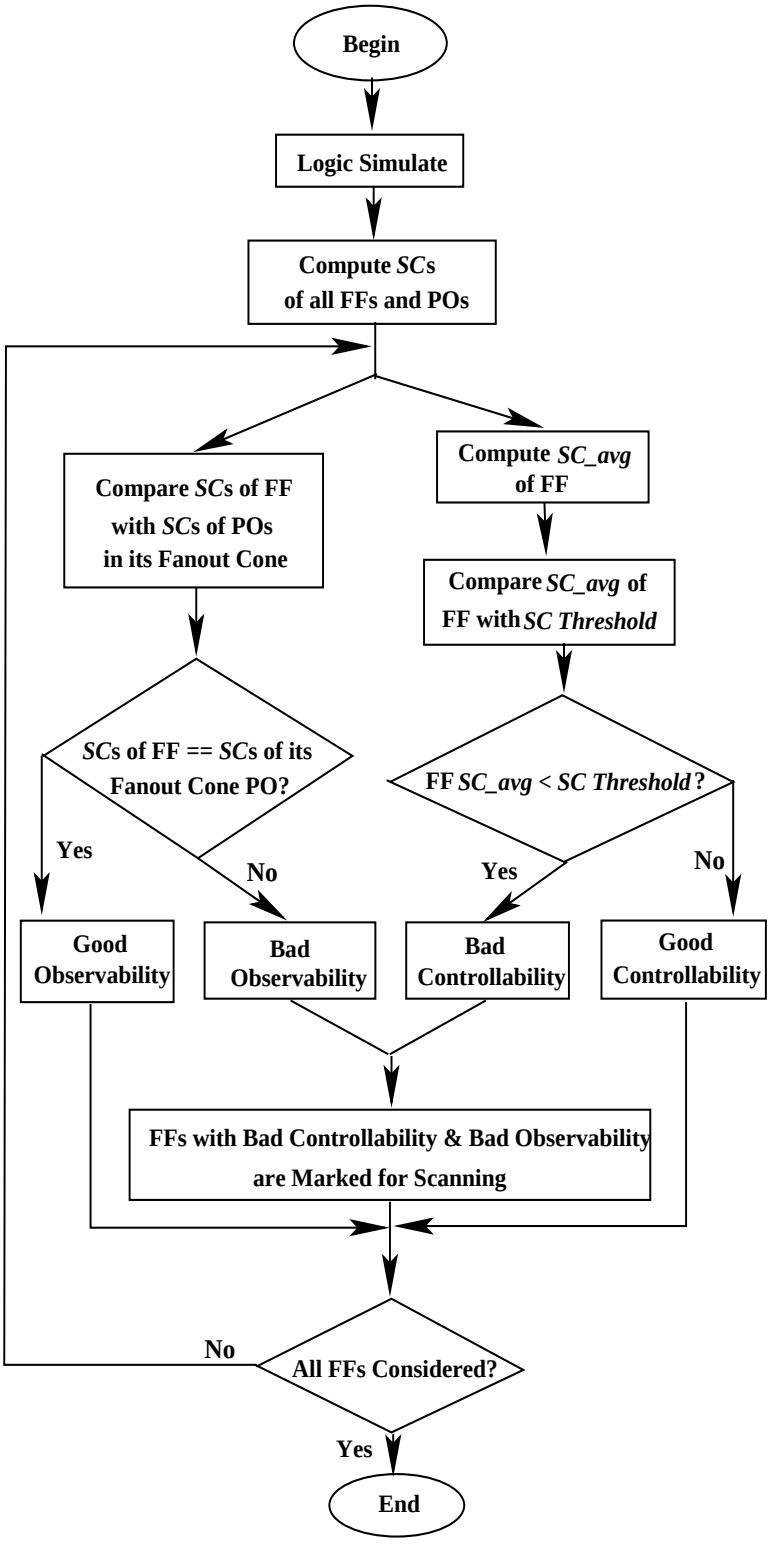

Figure 3.2: Procedure 3.1 – SPARTAN's spectral scan flip-flop selection.

(a) For each SCC i: /*Locate largest SCC not visited yet.*/

i. If $SCC(i)\_scanned == FALSE$

A. If $SCC(i)\_size > maxSCCSz \Rightarrow maxSCCSz = SCC(i)\_size$ and $SCC\_max = SCC(i)$.

(b) For each flip-flop, $FF(i)$, in SCC_max:

i. If $scan\_mark(FF(i)) == TRUE$

A. $scan(FF(i)) = TRUE$ and $numScanFF++$.

B. If $numScanFF > ST \Rightarrow$break for-loop (Step 4b).

(c) $SCC(i)\_scanned = TRUE$. □

3.1.1.3 Computational Complexity

Complexity Analysis of Procedure 3.1

Assume N logic gates, L lines, F flip-flops, and P POs in the circuit. Assume that there are P_{FO} POs in a flip-flop's fanout cone in the CUT. Also, the CUT is logic simulated using V vectors. To find SCCs, we need to perform *depth-first search* (DFS) on the s-graph. The complexity if DFS is $O(N+L)$ [20]. The complexity of Step 1 is $O(VN)$. Step 2 computes the SCs of of the flip-flops and the POs. To compute the SCs, a transform matrix is multiplied with the bit stream (states of a flip-flop or logic values of a PO) under analysis. Since we use a 16×16 RWT, the complexity of multiplying the RWT and the last 16 bits of a flip-flop state or PO response is 256. For V vectors, a total of V states for each flip-flop and V responses for each PO need to be analyzed. The 16×16 RWT analyzes 16 bits of flip-flop states or PO response in a clock cycle. The analysis starts on the 16^{th} clock cycle so that there are at least 16 bits to analyze. For each subsequent clock cycle (17 and so on), the RWT analyzes 16 bits, 15 bits from the previous 15 clock cycles and 1 bit from the current clock cycle. It is analogous to sliding a 16 bit window, over the states of a flip-flop or responses at a PO, by 1 bit in each clock cycle and analyzing the bits in the window. So, the number of times the RWT performs spectral analysis on each flip-flop and PO is $V-15$ (or $V-(16-1)$). Therefore, the cost of analyzing F flip-flops and P POs is $(F+P) \times (V-(16-1)) \times 256$. If a $R \times R$ RWT is used, the total cost of Step 2 will then be $(F+P) \times (V-(R-1)) \times R^2$. The loop in Step 3 compares each flip-flop's SCs with the SCs of all POs in its fanout cone. Since there are 16 coefficients for each PO and P_{FO} POs in

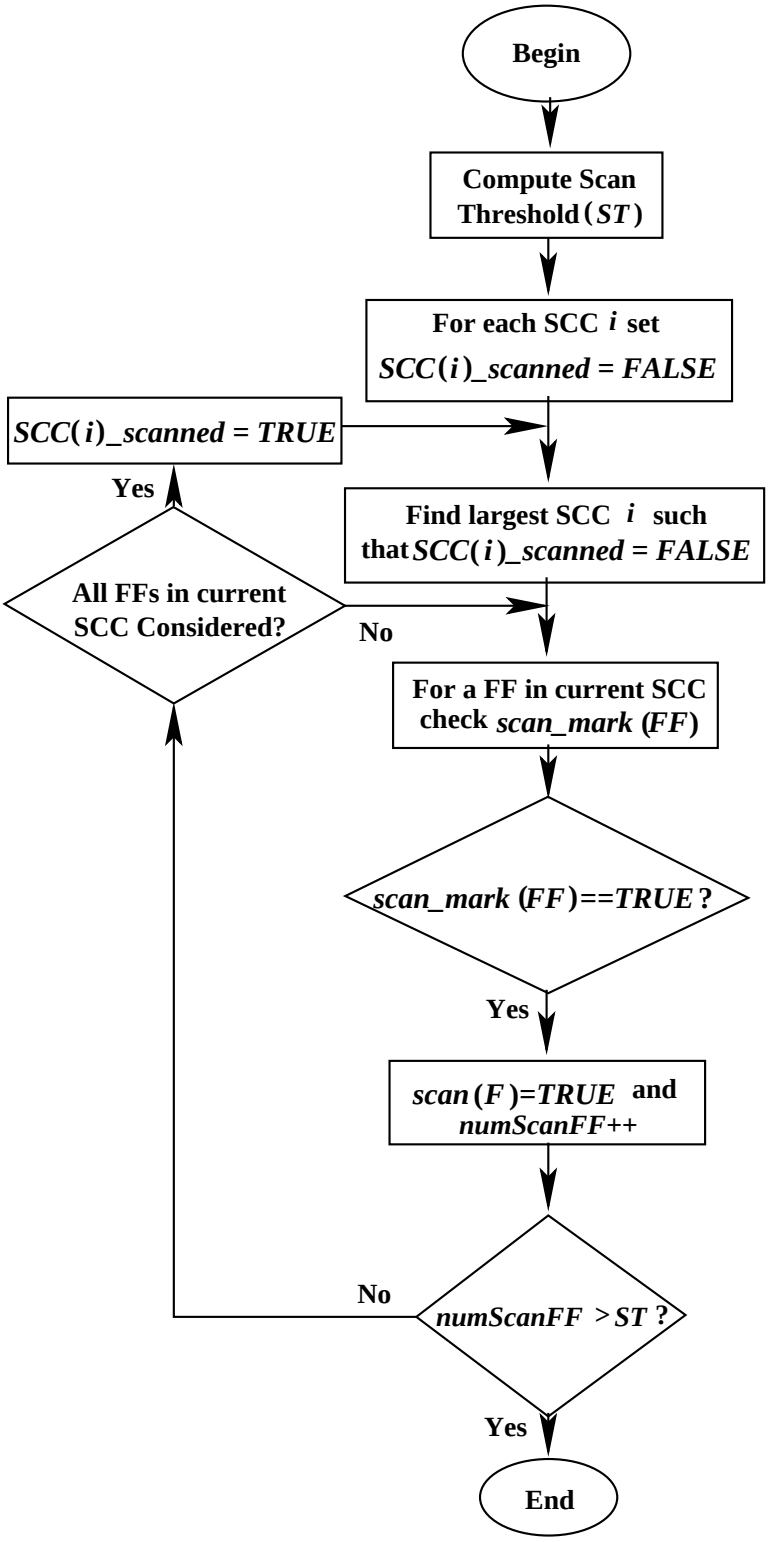

Figure 3.3: Procedure 3.1a – SPARTAN's spectral scan flip-flop selection.

each flip-flop's fanout cone, we get a total complexity of $O(F \times (16 \times P_{FO}))$. The complexity of the loop in Step 6 is $O(F)$. Therefore, the total computational complexity of Procedure 3.1 is approximately $O((N+L)+VN+(F+P)V+F \times P_{FO})$. If $V \approx N$, $F << N$, $P << N$, and $P_{FO} << N$, then the approximate complexity of Procedure 3.1 is $O(N^2)$.

Complexity Analysis of Procedure 3.1a

Assume that there are S SCCs in the circuit, F_{SCC} flip-flops in each SCC, and that ST is the scan threshold. The Step 4 loop iterates ST times. Step 4a locates the largest SCC not visited so far. Therefore, the complexity of loop in Step 4a is S. The loop in Step 4b checks for flip-flops recommended for scanning in the current SCC being visited. The loop's complexity is F_{SCC}. Therefore, the total complexity of Step 4 and Procedure 3.1a is $O(ST(S+F_{SCC}))$. If $F_{SCC} \approx N$ and $S << N$, then the approximate complexity of Procedure 3.1a is $O(ST(S))$.

3.1.2 Entropy Analysis

SPARTAN uses entropy H as a testability measure to select scan flip-flops such that the entropy of unscanned flip-flops in the CUT is enhanced. Agrawal showed that the testability of a circuit can be improved by increasing entropy at the *primary outputs* (POs) [2]. We conjecture that increasing the entropy of a line l in the circuit is akin to increasing the controllability of l. Figure 2.5 shows that entropy is maximum when logic 0 and logic 1 are equally likely, i.e., the probabilities of logic 0 ($p(0)$) and logic 1 ($p(1)$) occurring are equal ($p(0) = p(1) = 0.5$). Then, controlling l to 0 or 1 from PIs should become easier, thus increasing l's controllability. The entropy analysis uses random-pattern testability analysis since $p(1)$ and $p(0)$ are required to compute entropy [57]. Using entropy analysis is analogous to using Fourier analysis. Given the time domain data, Fourier analysis gives the spectral components of the data. Similarly, $p(0)$ and $p(1)$ of each line in the circuit are like time domain data and the entropy analysis gives us the amount of information on each line.

Cycle Breaking

Breaking cycles in the s-graph of the CUT has been shown to improve circuit testability [15, 17, 62, 77]. Entropy analysis uses the SCC information for selecting the best scan candidates because SCCs give us cycle information by capturing direct and indirect paths from a flip-flop

to every other flip-flop in the same SCC. Each flip-flop in a SCC lies on at least one cycle. Flip-flops from different SCCs cannot lie on any cycle together because if these two flip-flops were reachable from each other, they would be included in the same SCC and not different SCCs. Therefore, we perform entropy analysis on one SCC at a time in an effort to break cycles (since flip-flops from two distinct SCCs cannot be in any cycle together) in all the SCCs and improve the CUT's testability.

Probability Calculation

We use logic simulation to compute the probability values, which are then used to compute flip-flop entropy values. *Parker and McCluskey's probability expressions* (PMEs) to compute probabilities can also be used [57]. Even though PMEs provide a quick method for approximating probability values, we used probability values obtained via logic simulations because they are more accurate. Table 3.2 shows the probability and entropy values, of benchmark s820, for the deterministic method (PME) and logic simulation. The first column gives us the flip-flop number, the second column gives the probability, $p_{sim}(1)$, of a logic 1 obtained via logic simulation and the third column is the corresponding entropy value H_{sim}. Similarly, the fourth column gives the probability, $p_{det}(1)$, of a logic 1 obtained using PMEs [57] and the fifth column is the corresponding entropy value H_{det}. The sixth column gives us the difference in entropy values for the two methods. Entropy values for the two methods are not identical because PMEs assume that each line in the CUT is independent, which is not accurate. Since logic simulation accounts for correlation among signal lines in the CUT, probability values obtained via logic simulation are more accurate.

Table 3.2: Experimental and deterministic values for probability of logic 1 ($p(1)$) and entropy (H) for s820.

DFF	Simulation		Deterministic		% Entropy
	$p_{sim}(1)$	H_{sim}	$p_{det}(1)$	H_{det}	Difference
1	0.329	0.914	0.272	0.844	7.66
2	0.036	0.222	0.019	0.136	38.74
3	0.037	0.227	0.046	0.267	17.62
4	0.044	0.260	0.076	0.387	48.85
5	0.137	0.576	0.106	0.487	15.45
Avg.		0.440		0.424	25.66

Entropy for Scan Flip-Flop Selection

First, we initialize the state machine to a random state, and we logic simulate it using 512 random vectors to compute the probabilities. After all of the probabilities are computed, these values are used to compute the entropy, $H_i(Q)$, using Equation 3.6 and conditional entropies, $H_i(D|PI_j)$ and $H_i(Q|PO_k)$, using Equation 3.7 for each flip-flop i.

$$H(X) = -((1-p)\log_2(1-p) + p\ \log_2(p)) \tag{3.6}$$

$$H(X|Y) = \sum_{y \in Y} p(y)H(X|Y=y) \tag{3.7}$$

$H_i(Q)$ tells us the amount of information at flip-flop i. $H_i(D|PI_j)$ gives the amount of information (controllability) on flip-flop i's input given a logic value (0 or 1) on the j^{th} PI, PI_j, in its fanin cone. Similarly, $H_i(Q|PO_k)$ gives the amount of information (observability) on flip-flop i's output given a logic value (0 or 1) on the k^{th} PO, PO_k, in its fanout cone. The number of controllability values ($H_i(D|PI)$) for flip-flop i will depend on the number of PIs in its fanin cone and number of observability values ($H_i(Q|PO)$) will depend on the number of POs in its fanout cone. Then, for each flip-flop i, we compute the average controllability and observability values using Equations 3.8 and 3.9, where p is the number of PIs in the fanin cone and q is the number of POs in the fanout cone of any flip-flip i. We maximize the information flow through the circuit and the unscanned flip-flops. Increasing the information flow in the circuit is analogous to increasing the testability.

$$H_{i\_avg}(D|PI) = \frac{\sum_{x=1}^{p} H_i(D|PI_x)}{p} \tag{3.8}$$

$$H_{i\_avg}(Q|PO) = \frac{\sum_{x=1}^{q} H_i(Q|PO_x)}{q} \tag{3.9}$$

So far we have $H_i(Q)$, $H_{i\_avg}(D|PI)$, and $H_{i\_avg}(Q|PO)$ values for each flip-flop i. Now we need average values $H(Q)_j$, $H(D|PI)_j$, and $H(Q|PO)_j$ for any SCC_j. Equations 3.10, 3.11, and 3.12 compute the average entropy values for any SCC_j where m is the number of nodes in SCC_j.

$$H(Q)_j = \frac{\sum_{i=1}^{m} H_i(Q)}{m} \tag{3.10}$$

$$H(D|PI)_j = \frac{\sum_{i=1}^{m} H_{i\_avg}(D|PI)}{m} \tag{3.11}$$

$$H(Q|PO)_j = \frac{\sum_{i=1}^{m} H_{i\_avg}(Q|PO)}{m} \tag{3.12}$$

Similarly, other SCCs' average entropies are computed. For each SCC (of size > 1), the algorithm checks to see which flip-flops in the SCC, if scanned, will cause an increase in the average entropy values of the SCC. It does so by making the flip-flop's input a *pseudo-primary output* (PPO) and the output a *pseudo-primary input* (PPI). It then performs logic simulation to recompute the probabilities of the circuit. Since logic simulation is performed for each flip-flop, to keep the computation time tractable, logic simulation was performed with 512 random vectors. This flip-flop is marked as 'tried.' Initially all flip-flops are marked as 'not tried.' Since any flip-flop in a SCC has a direct or indirect path to every other flip-flop in that SCC, scanning a flip-flop will affect the probabilities of all of the unscanned flip-flops in that SCC and possibly other SCCs. This means scanning a flip-flop in SCC_j will change the entropies of the unscanned flip-flops and thus the average entropy of the SCC_j will change. The new average entropy values of SCC_j become $H(Q)'_j$, $H(D|PI)'_j$, and $H(Q|PO)'_j$. The changes in average entropy values are given in Equations 3.13, 3.14, and 3.15.

$$\triangle H(Q)_j = H(Q)'_j - H(Q)_j \tag{3.13}$$

$$\triangle H(D|PI)_j = H(D|PI)'_j - H(D|PI)_j \tag{3.14}$$

$$\triangle H(Q|PO)_j = H(Q|PO)'_j - H(Q|PO)_j \tag{3.15}$$

If any one of the average entropy values ($H(Q)'_j$, $H(D|PI)'_j$, and $H(Q|PO)'_j$) of SCC_j, after selecting the flip-flop (currently being considered for scanning), increases at all, the flip-flop is marked as a good candidate for scanning because it increased the controllability and observability of other flip-flops in the SCC. The remaining flip-flops in the SCC are then 'tried' for scanning, one at a time, and the ones that increase the average entropy values of the SCC are marked as good choices for scanning. After all of the flip-flops have been tried for scanning, the flip-flops that were marked for scanning are chosen as scan flip-flops (and converted into a *pseudo-primary input* (PPI) and a *psuedo-primary output* (PPO)). Procedure 3.2 is the entropy analysis algorithm and Figure 3.4 is the flow chart of Procedure 3.2.

Procedure 3.2 – Entropy Analysis:

1. Logic simulate to compute values of $p(0)$ and $p(1)$ for all flip-flops.

2. For each flip-flop f:

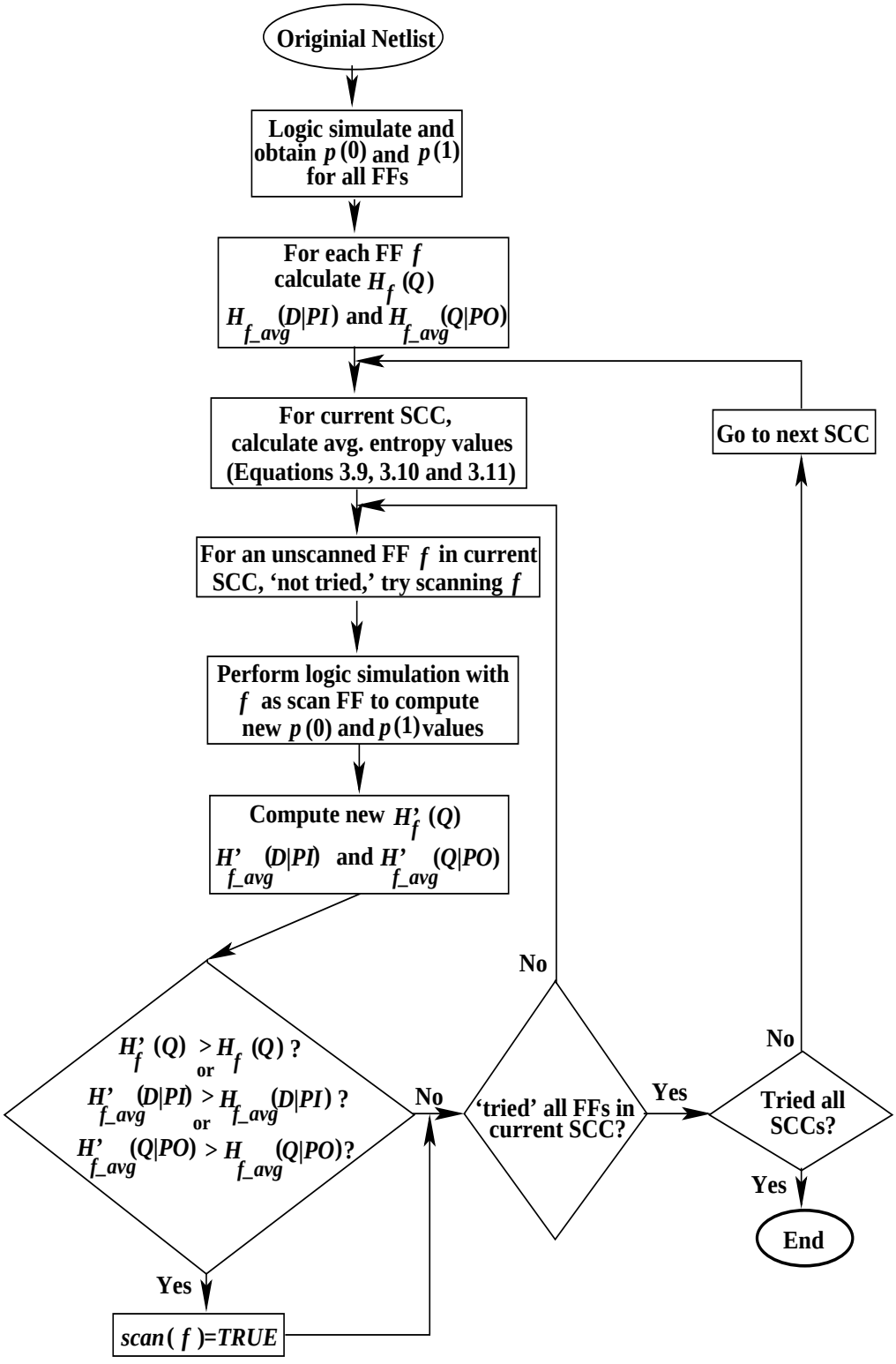

Figure 3.4: Procedure 3.2 – SPARTAN's entropy scan flip-flop selection.

(a) Use $p_f(0)$ and $p_f(1)$ to calculate entropy values $H_f(Q)$, $H_{f\_avg}(D|PI)$, and $H_{f\_avg}(Q|PO)$.

3. For each SCC S:

(a) Calculate average entropy values of SCC using Equations 3.10, 3.11, and 3.12.

i. For each unscanned and 'not tried' flip-flop f in the current SCC:

A. Try scanning f (mark it 'tried') by converting its input (output) into a PPO (PPI).

B. Perform logic simulation and compute new probabilities.

C. Backup S's original average entropy values $H(Q)_S$, $H(D|PI)_S$ and $H(Q|PO)_S$. Then compute S's new average entropy values $H(Q)'_S$, $H(D|PI)'_S$ and $H(Q|PO)'_S$ from the updated probability values.

D. If $\frac{H(Q)'_S - H(Q)_S}{H(Q)_S} > 0.0$ || $\frac{H(D|PI)'_S - H(D|PI)_S}{H(D|PI)_S} > 0.0$ || $\frac{H(Q|PO)'_S - H(Q|PO)_S}{H(Q|PO)_S} > 0.0$ by scanning f, 'flag' f as a candidate for scanning.

ii. If the number of 'flagged' flip-flops is 0:

A. No scan flip-flop is chosen from the SCC S, so try remaining untried SCCs. □

3.1.2.1 Computational Complexity

Complexity Analysis of Procedure 3.2

Assume that there are N logic gates, F flip-flops, and S SCCs in the CUT. Each SCC has F_{SCC} flip-flops and the CUT is logic simulated with V vectors. The complexity of Step 1 is $O(VN)$ and that of Step 2 is $O(F)$. For each SCC, Step 3 calculates the average entropy in Step 3(a), tries each flip-flop as a scan candidate in step 3(a)iA, and logic simulates (to compute updated probability values) in step 3(a)iB. The complexity for the average calculation is $O(F_{SCC})$ and the complexity of trying each flip-flop and logic simulating after each try is $O(F_{SCC}(VN))$. Therefore, the total complexity of Step 3 is $O(S(F_{SCC}(VN+1)))$. The total complexity of Procedure 3.2 is approximately $O(VN(SF_{SCC}+1))$. If $V \approx N \approx F_{SCC}$, then the approximate computational complexity of Procedure 2 is $O(N^3S)$.

3.1.3 Combined Analysis

Note that entropy analysis is applied to SCCs with cardinality > 1 and spectral analysis is done on all of the flip-flops. SPARTAN first performs the spectral analysis and then the entropy analysis. A flip-flop is selected for scan if it is picked by both the spectral and the entropy analysis, and if it is more than 4 levels away from a PI and more than 4 levels away from a PO. Figure 3.5 shows the SPARTAN's flow chart and Algorithm 3.1 is the SPARTAN algorithm.

Algorithm 3.1 – The SPARTAN Algorithm.

Input: Circuit netlist.
Output: Modified circuit netlist with scan flip-flops.

```
Spartan(CIRCUIT)
{
    Construct the s-graph from netlist;
    Condense s-graph to obtain SCCs;
    Perform logic simulation with 512 random vectors;
    Perform spectral analysis via Procedure 3.1 in Section 3.1.1
    to obtain SCs for all flip-flops;
    Select spectral analysis based scan flip-flops;
    Perform entropy analysis via Procedure 3.2 in Section 3.1.2
    to obtain entropy for all flip-flops;
    Select entropy analysis based scan flip-flops;
    Flip-flops selected by both methods are scanned
}
```

3.2 Summary

Section 3.1 introduces a new partial-scan algorithm that uses both spectral and entropy analysis to select scan flip-flops. Difficult to control flip-flops have an adverse effect on the testability of the circuit. The first idea here is to scan flip-flops that greatly increase the controllability of unscanned flip-flops. Low toggling flip-flops are difficult to control from PIs. Spectral analysis helps the algorithm to find flip-flops that have low activity. For observability analysis, the spectra of a flip-flop are compared to the spectra of the POs in the flip-flop's fanout cone. A flip-flop has good observability if its spectra is identical to the spectra of at least one PO in its fanout cone. The second idea involves scanning flip-flops such that information flow or entropy through unscanned flip-flops from PIs is improved. In other words, increasing information flow through the circuit improves circuit testability. If the amount of information passing through a

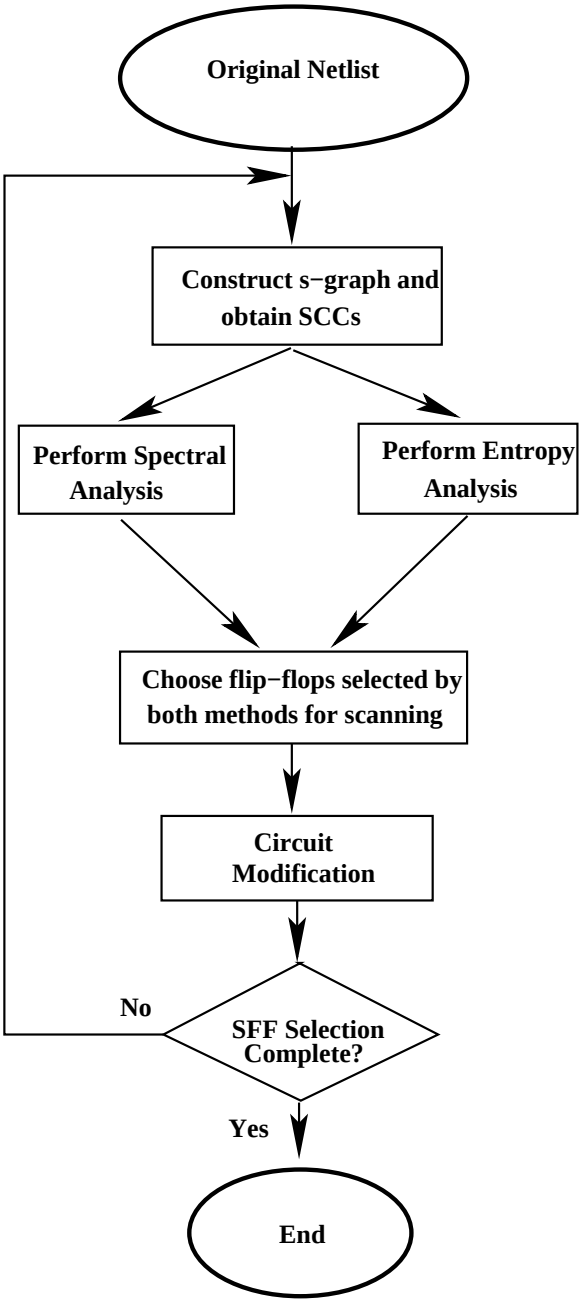

Figure 3.5: SPARTAN's scan flip-flop selection procedure.

flip-flop is high, the controllability and observability of the flip-flop are high.

Chapter 4

Spectral and Entropy Based Partial-Scan and *Test Point Insertion* (TPI)

4.1 Introduction

Test point insertion (TPI) is a commonly used *ad hoc design-for-testability* (DFT) technique for improving the controllability and observability of internal nodes in a circuit. Testability analysis is typically used to identify the internal nodes where *test points* (TPs) are needed, in the form of *control points* (CPs), *observation points* (OPs), or *complete test points* (CTPs), also know as *scan points* (SPs).

The structure of an OP (see Figure 4.1) is comprised of a *multiplexer* (MUX) and a D flip-flop. An OP is inserted on a line with low observability, by connecting the line to the 0 port of the MUX. The logic value of the line is captured in the D flip-flop. The captured value is then observed directly by shifting the value out of the OP's D flip-flop. If the circuit requires multiple OPs, the lines with low controllability are connected to the 0 input of the MUX in each OP. All OPs are connected into a shift register by connecting the *scan-in* (SI) pin of any OP, OP_n, to the *scan-out* (SO) pin of the preceding OP or to the SI pin of the circuit and the SO pin of OP_n to the SI pin of the succeeding OP or the SO pin of the circuit. Once the logic values on these lines are captured in the D flip-flops of the OPs, they are scanned out serially from the OP shift register.

The structure of a CP, shown in Figure 4.2, is slightly different from the structure of an OP. A CP is inserted on a line with poor controllability. The line in the circuit with low controllability is cut and a MUX is inserted between the source and destination ends. The 0 input of the MUX is connected to the original source and the 1 input is connected to the Q-line of the CP flip-flop. During normal operation the *test mode* (TM) is set to 0 and the value of the original source drives the destination. During test, TM is set to 1 so that the value from the D flip-flop drives the destination end. The flip-flops in the CP are connected into a shift register

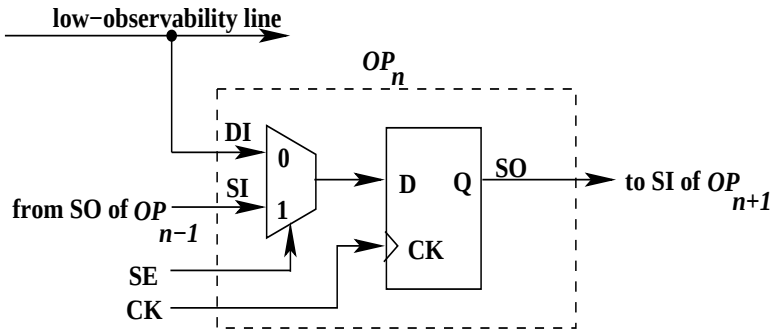

Figure 4.1: An observation point [73].

so that required logic values can be shifted into the corresponding CP. Adding CPs results in additional delay on the logic paths they are added on, hence care must be taken not to insert CPs on a critical path.

Figure 4.3 shows a *complete test point* (CTP). The 0 input of MUX1 used to capture and observe the logic value of the line into the flip-flop and the 1 input is used to scan in the values from the preceding flip-flop in the scan chain. During normal operation, SE is set to 0. Similarly, the 0 input of MUX2 is connected to the line that needs to be controlled and the 1 input is connected to the flip-flop of the CTP. During normal operation, TM is set to 0. During test mode, first SE is set to 1 and TM is set to 0 for shifting in the required logic values. Then the required logic values for each line are shifted into the corresponding flip-flops of the CTPs. The test vector is then applied with SE set to 0 and TM set to 1. TM is set to 1 so that the value in the flip-flop is used to control the value of the line and SE is set to 0 so that the new logic value on the line can be captured. After the test vector is applied, SE is set to 1 and TM to 0. The logic values captured in the flip-flops chained together are then shifted out. This process is repeated for each test vector.

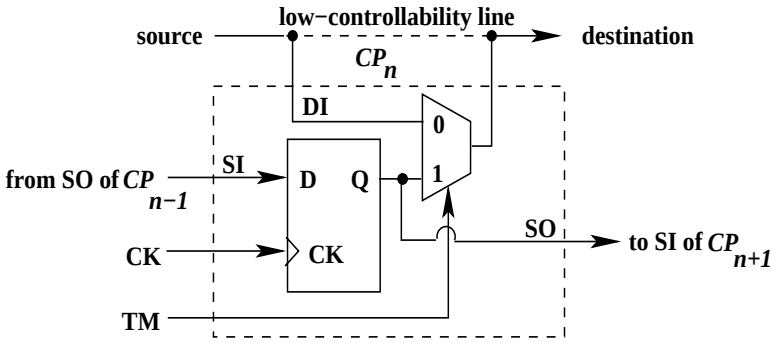

Figure 4.2: A control point [73].

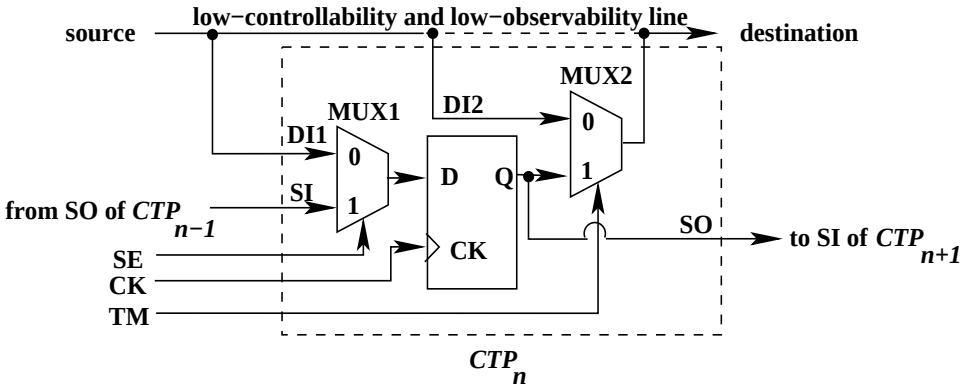

Figure 4.3: A complete test point [73].

4.2 The Spectral and Entropy-Based TPI Algorithms

The two greedy TPI algorithms proposed first construct a test point candidate line set L. Then they select good candidate lines from set L for test point insertion using spectral and entropy analysis. The initial construction of L is what differentiates the two algorithms (see Sections 4.3.1 and 4.3.2). Test point insertion is an *ad hoc* DFT technique to enhance circuit testability and allows the ATPG to attain a very high fault coverage with a shorter test set and also helps reduce ATPG time drastically. The spectral and entropy based TPI algorithms introduced use CTPs (or SPs).

4.2.1 Spectral Analysis

Section 3.1.1 explained the procedure (Procedure 3.1) to compute the *spectral correlation coefficients* (SCs) of a flip-flop. We use the identical procedure to compute SCs of each potential TP candidate line (output of a gate). The SCs of each candidate line, which are the coefficients of the candidate lines in the spectral domain, are used as a testability measure. A low toggling line will have a small average SC value and thus will have low controllability. For observability of a candidate line, the SCs of the candidate line are compared to the SCs of each PO in its fanout cone. A candidate line will have good observability if its SCs match with SCs of at least one PO in its fanout cone. Otherwise, the candidate line is categorized as low-observability line. A line with low controllability and low observability is marked as a good candidate for test point insertion. The spectral analysis procedure explains the steps taken for test point insertion.

Figure 4.4 is the flow chart for the spectral analysis procedure.

The Spectral Analysis Procedure:

1. Logic simulate the CUT with 512 random vectors to obtain valid logic values for lines in set L and all POs.

2. Compute SCs of candidate lines in L and POs.

3. Set $SC_{avg\_MAX} = 0$ and $SC_{avg\_MIN} = 0$.

4. For each candidate line l in L:

   (a) Compute the average of SCs, SC_{l_avg}, for line l with $k = 16$ using Equation 4.1. In Equation 4.1 w_j is a normalizing coefficient and it assigns lower weights to the low sequency signals and higher weights to the high sequency signals, which directly correlates to the controllability of the candidate line. Equation 4.2 is used to compute the value for w_j where φ_j is the sequency of the j^{th} transform matrix row. The sequency, which is analogous to frequency, of a row is the number of $1 \rightarrow -1$ and $-1 \rightarrow 1$ transitions in that row. For example, in Equation 3.1, the sequency of the first row of the matrix is zero, the sequency of the second row is one and so on.

$$SC_{l\_avg} = \frac{\sum_{j=1}^{k} w_j |SC_j|}{k} \tag{4.1}$$

$$w_j = \frac{(\varphi_j + 1)}{\sum_{h=1}^{k} (\varphi_h + 1)} \tag{4.2}$$

   (b) If $SC_{l\_avg} \geq SC_{avg\_MAX} \Rightarrow SC_{avg\_MAX} = SC_{l\_avg}$.

   (c) If $SC_{l\_avg} \leq SC_{avg\_MIN} \Rightarrow SC_{avg\_MIN} = SC_{l\_avg}$.

   (d) Compare the SCs of candidate line l with the SCs of each PO its fanout cone.

      i. If SCs of candidate line l == SCs of any fanout cone PO $\Rightarrow$

         $Good\_Observability(l) = TRUE$.

      Else $Good\_Observability(l) = FALSE$.

5. Compute the *SC Threshold* using Equation 4.3.

$$SC\ Threshold = \text{average}\ (SC_{avg\_MIN}, SC_{avg\_MAX}) \tag{4.3}$$

6. For each candidate line l:

   (a) If $SC_{l\_avg} < SC\ Threshold$ && $Good\_Observability(l) == FALSE \Rightarrow$ $TPI\_Flag(l) = TRUE$ (i.e., the candidate line l is a good candidate for test point insertion). □

4.2.1.1 Computational Complexity of the Spectral Analysis Procedure

Assume that there are N logic gates, L_{total} lines, and P POs. Assume that there are P_{FO}^{l} POs in a candidate test point line l's fanout cone in the CUT. Also, the CUT is logic simulated using V vectors. Step 1 performs logic simulation so the complexity of Step 1 is $O(VN)$. Step 2 computes the SCs of each candidate line l in L and the POs. To compute the SCs, a transform matrix is multiplied with the bit stream (logic values of each l or logic values of a PO) under analysis. Since we use a 16×16 RWT, the complexity of multiplying the RWT and the last 16 bits of logic values of each line l or PO response is 256. For V vectors, a total of V logic values for each l and V responses for each PO need to be analyzed. The 16×16 RWT analyzes 16 bits of logic values of each l or PO response in a clock cycle. The analysis starts on the 16^{th} clock cycle so that there are at least 16 bits to analyze. For each subsequent clock cycle (17 and so on), the RWT analyzes 16 bits, 15 bits from the previous 15 clock cycles and 1 bit from the current clock cycle. It is analogous to sliding a 16 bit window, over the logic values of a candidate line l or responses at a PO, by 1 bit in each clock cycle and analyzing the bits in the window. So, the number of times RWT performs spectral analysis on each candidate line l and PO is $V - 15$ (or $V - (16 - 1)$). Therefore, the cost of analyzing L candidate lines and P POs is $(L + P) \times (V - (16 - 1)) \times 256$. If a $R \times R$ RWT is used, the total cost of Step 2 will then be $(L + P) \times (V - (R - 1)) \times R^2$. The cost of Steps 3, 4(a), 4(b), and 4(c) is $O(1)$. The loop in Step 4(d) compares each candidate line l's SCs with SCs of all POs in its fanout cone. Since there are 16 coefficients for each PO and P_{FO}^{l} POs in each candidate line l's fanout cone, we get a total complexity of $O(L \times (16 \times P_{FO}^{l}))$. The complexity of the loop in Step 6 is $O(L)$. Therefore, the total computational complexity of the spectral analysis procedure is approximately $O(VN + (L + P)V + L \times P_{FO}^{l})$.

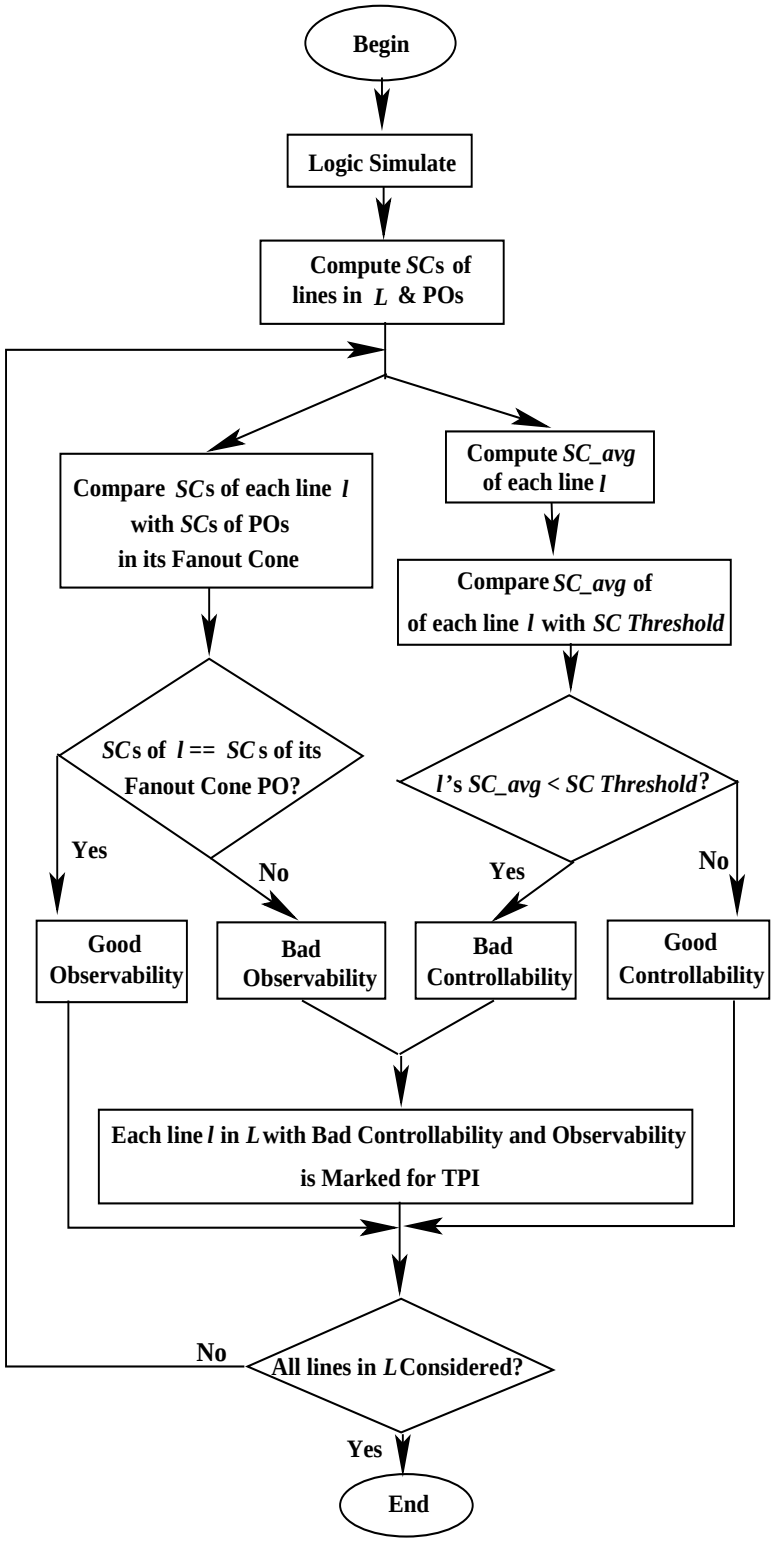

Figure 4.4: The spectral analysis procedure used for test point selection.

4.2.2 Entropy Analysis

Section 3.1.2 explained the procedure to compute entropy as a testability measure to select scan flip-flops. We use logic simulation to compute the probability values, $p(0)$ and $p(1)$, of logic 0 and logic 1 on each line (or gate output) in the circuit. These probability values are then used to compute the entropy values of each line. We use logic simulation because topology-based testability analysis such as SCOAP [30] and probability-based testability measures [57] are *static*, in the sense that they do not use input test patterns for testability analysis. The topology-based testability analysis is efficient, but the efficiency is achieved at the cost of reduced accuracy, especially for circuits that contain many reconvergent fanouts [4, 73]. Since most large circuits have a large number of reconvergent fanouts, the simulation-based testability (*dynamic*) analysis gives more accurate estimates of the probability values and the testability measures.

Entropy Analysis for Test Point Insertion

First, we initialize the state machine to a randomly selected state, and we logic simulate it using 512 random vectors to compute the probabilities. After all of the probabilities are computed, these values are used to compute the entropy, $H(l)$, using Equation 4.4 and conditional entropies, $H(l|PI_j)$ and $H(l|PO_k)$, using Equation 4.5 for each test point candidate line l.

$$H(X) = -((1-p)\log_2(1-p) + p\ \log_2(p)) \tag{4.4}$$

$$H(X|Y) = \sum_{y \in Y} p(y)H(X|Y=y) \tag{4.5}$$

$H(l)$ tells us the amount of information on a candidate line l. $H(l|PI_j)$ gives the amount of information (controllability) on line l given the logic value (0 or 1) on the j^{th} PI, PI_j, in its fanin cone. Similarly, $H(l|PO_k)$ gives the amount of information (observability) on line l given the logic value (0 or 1) on the k^{th} PO, PO_k, in its fanout cone. The number of controllability values ($H(l|PI)$) for line l will depend on the number of PIs in its fanin cone and the number of observability values ($H(l|PO)$) will depend on the number of POs in its fanout cone. Then, for each line l, we compute the average controllability and observability values using Equations 4.6 and 4.7, where p is the number of PIs in the fanin cone and q is the number POs in the fanout cone of any line l. By adding test points, we maximize the information flow through the rest of the circuit. Increasing the information flow in the circuit is analogous to increasing the

testability and improving its random pattern testability.

$$H_{avg}(l|PI) = \frac{\sum_{x=1}^{p} H_i(l|PI_x)}{p} \quad (4.6)$$

$$H_{avg}(l|PO) = \frac{\sum_{x=1}^{q} H_i(l|PO_x)}{q} \quad (4.7)$$

So far, we have $H(l)$, $H_{avg}(l|PI)$, and $H_{avg}(l|PO)$ values for each candidate line l. Now we need average values $H(L)$, $H(L|PI)$, and $H(L|PO)$ for the entire test point candidate line set L. Equations 4.8, 4.9, and 4.10 compute the average entropy values for the test point candidate set L, where m is the number of candidate lines in the set L.

$$H(L) = \frac{\sum_{i=1}^{m} H_i(L)}{m} \quad (4.8)$$

$$H(L|PI) = \frac{\sum_{i=1}^{m} H_{i\_avg}(L|PI)}{m} \quad (4.9)$$

$$H(L|PO) = \frac{\sum_{i=1}^{m} H_{i\_avg}(L|PO)}{m} \quad (4.10)$$

The TPI algorithm checks to see if converting each candidate line to a test point will cause an increase in the average entropy values of the candidate lines in the set L. It does so by cutting the candidate line and inserting the CTP hardware (see Figure 4.3) on the candidate line. The logic simulator treats the CTP flip-flop as a *pseudo-primary output* (PPO) and a *pseudo-primary input* (PPI). The algorithm then performs logic simulation to recompute the probabilities of the circuit. Since logic simulation is performed each time a test point candidate l is tried, to keep the computation time tractable, logic simulation was performed with 512 vectors. The new average entropy values $H(L)'$, $H(L|PI)'$, and $H(L|PO)'$ are compared with the original average entropy values $H(L)$, $H(L|PI)$, and $H(L|PO)$ for the entire test point candidate line set L. The changes in average entropy values are computes using Equations 4.11, 4.12, and 4.13.

$$\triangle H(L) = H(L)' - H(L) \quad (4.11)$$

$$\triangle H(L|PI) = H(L|PI)' - H(L|PI) \quad (4.12)$$

$$\triangle H(L|PO) = H(L|PO)' - H(L|PO) \quad (4.13)$$

If all of the average entropy values ($H(L)'$, $H(L|PI)'$, and $H(L|PO)'$) of the candidate line set L, after converting the line currently being considered into a test point, increase at all, the candidate is marked as a good candidate for test point insertion because it increased the controllability and observability of other lines in the circuit. The remaining candidate lines in the set L are then 'tried' for test point insertion, one at a time, and the ones that increase

the average entropy values of the set L are marked as good choices for test point insertion. After all of the candidate lines have been tried, the candidate lines that were marked for test point insertion by spectral and entropy analysis are chosen as test points (and converted into a *pseudo-primary input* (PPI) and a *pseudo-primary output* (PPO)). Figure 4.5 is the flow chart of the entropy analysis procedure.

The Entropy Analysis Procedure:

1. Logic simulate to compute values of $p(0)$ and $p(1)$ for candidate lines in L.
2. For each candidate line l in L:
   (a) Use $p_l(0)$ and $p_l(1)$ to calculate entropy values $H(l)$, $H_{avg}(l|PI)$ and $H_{avg}(l|PO)$.
3. Calculate average entropy values of set L using Equations 4.8, 4.9, and 4.10.
   (a) For each candidate line l 'not tried' in L:
      i. Try inserting a test point on l by converting its input (output) into a PPO (PPI).
      ii. Perform logic simulation and compute new probabilities.
      iii. Backup L's original average entropy values $H(L)$, $H(L|PI)$, and $H(L|PO)$. Then compute L's new average entropy values $H(L)'$, $H(L|PI)'$, and $H(L|PO)'$ from the updated probability values.
      iv. If $\frac{H(L)'-H(L)}{H(L)} > 0.0$ && $\frac{H(L|PI)'-H(L|PI)}{H(L|PI)} > 0.0$ && $\frac{H(L|PO)'-H(L|PO)}{H(L|PO)} > 0.0$ by inserting a test point on l, $TPI\_Flag(l) = TRUE$ (i.e., the candidate line l is a good candidate for test point insertion). □

4.2.2.1 Computational Complexity of the Entropy Analysis Procedure

Assume that there are N logic gates, L candidate test point candidate lines in the CUT, and that the CUT is logic simulated with V vectors. The complexity of Step 1 is $O(VN)$ and that of Step 2 is $O(L)$. For each candidate line l in L, Step 3 calculates the average entropy of the lines in L. In Step 3(a)i, each candidate line l is 'tried' as a test point and logic simulation is performed in

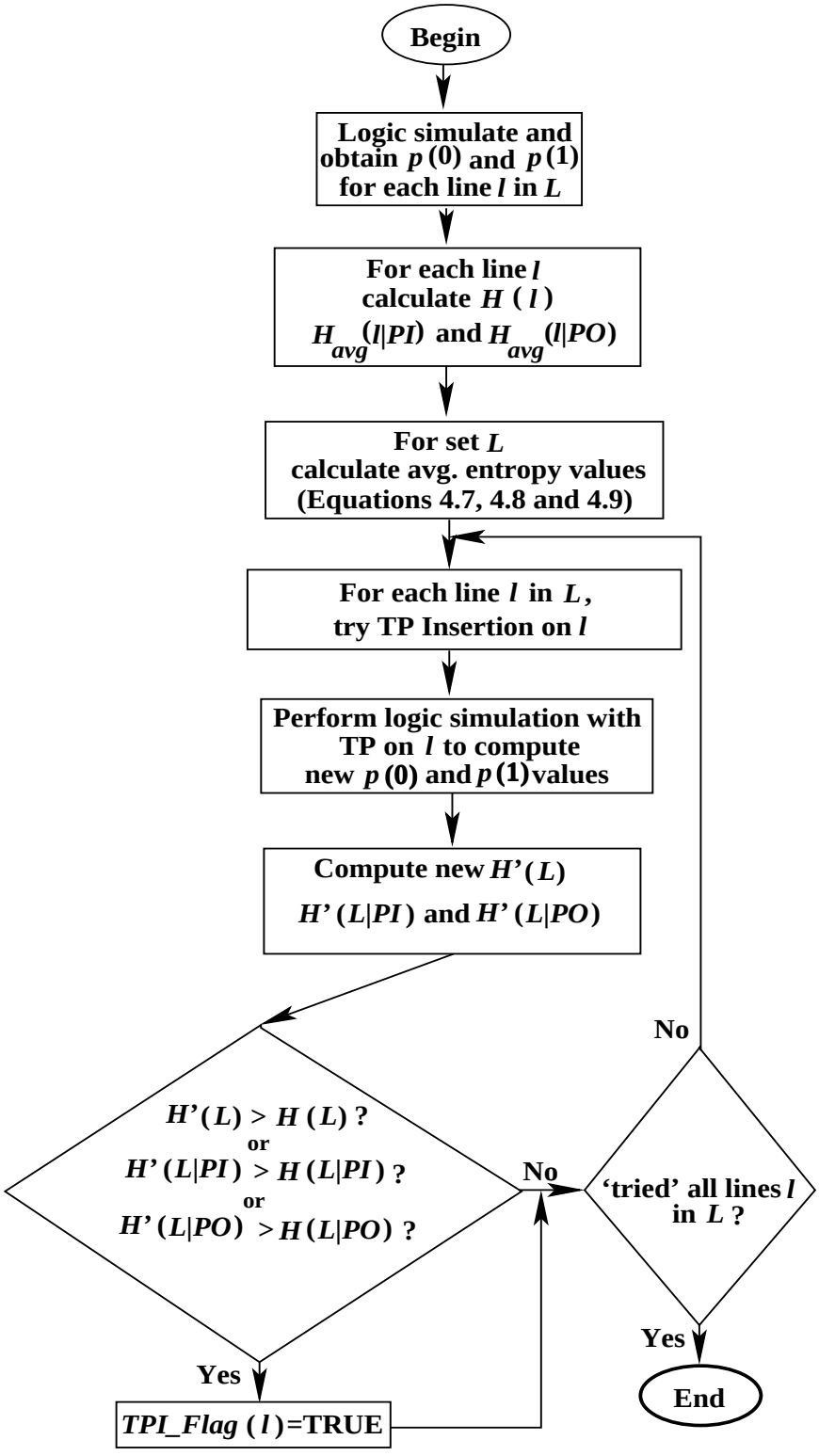

Figure 4.5: The entropy analysis procedure used for test point selection.

Step 3(a)ii (to compute updated probability values). The complexity for the average calculation is $O(L)$ and the complexity of trying each candidate line l and logic simulating after each try is $O(LVN)$. Therefore, the total complexity of Step 3 is $O(L(VN+1)))$. The total complexity of the entropy analysis procedure is approximately $O(VN(L+1)+L)$.

4.3 The TPI Algorithms

Two test point insertion algorithms, Algorithm 1 and Algorithm 2, are being proposed. Both algorithms incorporate the spectral and entropy analyses described in Section 4.2. What separates the two algorithms is the initial step used to populate the candidate test point line set L. The two algorithms are explained in detail in Sections 4.3.1 and 4.3.2.

4.3.1 The First Test Point Insertion Algorithm – TPI1

Population of Test Point Candidate List L

TPI1 first computes the variable *median*, which is half of the maximum number of levels in the circuit. The algorithm takes a user specified value, *level_range*, to find the gates that will be considered as test point candidates, where $0 < level\_range < 1$. The *level_range* specifies the fraction of the number of levels succeeding and preceding the middle of the circuit from which the initial test point candidate lines are to be selected. For example, if the maximum number of levels in the circuit is 10, *median* will be 5 and if $level\_range = 0.5$, then $median \pm \lfloor 0.5 \times median \rfloor$ will populate L with all the gates between levels 3 and 7. The variable *median* is used to locate the center of the circuit and test point candidates embedded in the circuit. Test point insertion in or near the center of the circuit is more effective because lines with low testability are usually embedded somewhere in the middle of the circuit. The variable *level_range* is used to expand the candidate selection from the circuit's center and to give the user the flexibility to choose the size of candidate set L. Ideally, we would want to try each and every line in the circuit, but this approach will be computationally infeasible for a large set L because we have to perform logic simulation each time we try a candidate test point in L. Once we are done populating the set L, we randomly pick test point candidates from L. A user specified value for the variable *numTPs* decides the number of random test point candidates picked from L. This further reduces the size of the candidate set L. The candidate test point lines that are not selected during the random selection are discarded from L.

Spectral and Entropy Analysis

Then TPI1 uses the spectral analysis procedure (see Section 4.2.1) and the entropy analysis procedure (see Section 4.2.2) on the test point candidate lines in L. After the spectral and entropy analyses are performed, the candidate lines l in L selected by both are marked as good candidates for test point insertion and the rest of the candidate lines in L are discarded.

The Location of *Observation Primary Inputs* (OPIs) – OPI Analysis

In this step, for each candidate line l in L, the algorithm locates the primary inputs that need to be specified to observe a fault effect on this line l [64]. Figure 4.6 shows a circuit with a logic cone C_1, where C_1 drives a PO and a logic cone C_2, where C_2 drives the candidate test point line l. Assume that a set of inputs I has to be specified, to control the logic gates in the shaded areas in Figure 4.6, to propagate any fault effect on line l to a PO. If we insert a test point at line l, then none of the inputs in I need to be specified to propagate the fault effects on l. Since adding a test point on l removes the requirement to specify the inputs in I, the test point on l can help reduce ATPG time and may reduce the test set length, especially if the ATPG produces don't cares in the test patterns. The set I is referred to as the *observability primary inputs* (OPIs) and is stored in $OPIs(l)$ for a line l. The goal here is to select candidate lines with large OPI sets. Therefore, the algorithm first locates the OPIs for all lines in L. The algorithm then compares and locates the number of OPIs of each line in L that overlap with the OPIs of every other line in L. For example, let us consider two candidate lines l and l' with OPI sets I and I'. If the OPI set I' is a subset of OPI set I, then we remove l' from the candidate list. This is because l' is in the neighborhood of l and selecting test points that are close to each other does not cause a significant improvement in the testability of the circuit. Also, since l is relieving a subset of PIs from being specified by the ATPG, selecting l' will not help reduce the ATPG time or test length. In case the OPI set I' is not a subset I, but has some overlap with I, the algorithm selects the candidate line with a larger OPI set and computes the amount of overlap between I and I'. If the smaller OPI set contains more than 50% overlap of the larger OPI set, the candidate line with the smaller OPI set is considered to be in the neighborhood of the candidate line with the larger OPI set. Thus, the candidate line with the smaller OPI set is discarded from L.

For two candidate lines l and l' under consideration, we compute the amount of overlap with a simple procedure using a for-loop. A flag, $pi\_acquired(I)$, is associated with each PI I

in the circuit. We set the flag $pi\_acquired(I_{FI}^{l})$ to $TRUE$ for each PI I_{FI}^{l} in l's fanin cone. Then we iterate through the PIs in l''s fanin cone and if the flag $pi\_acquired(I_{FI}^{l'})$ of PI $I_{FI}^{l'}$ in l''s fanin cone is $TRUE$, it means that this PI is also included in the fanin cone of line l. We count the total number of PIs in the fanin cone of l' that are set to $TRUE$ and this gives the amount of overlap between lines l and l'. The percentage overlap of two candidate lines is computed as the ratio of amount of overlap between the two lines and the total number of PIs in the fanin cone of the larger of the two cones.

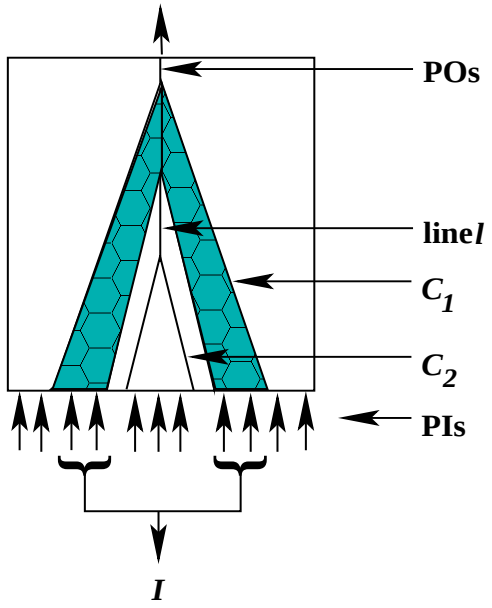

Figure 4.6: Example of circuit with observability PI set I.

After the OPI analysis in the last step, the remaining candidate lines in L are selected for test point insertion. Algorithm 4.1 shows the overview of TPI1.

4.3.1.1 Computational Complexity of TPI1

Assume that there are N logic gates, L_{total} lines, L test point candidate lines, and P POs. Assume that there are P_{FO}^{l} POs in a candidate test point line l's fanout cone in the CUT. Also, the CUT is logic simulated using V vectors. Steps 1 and 2 have a computational cost of $O(1)$. Step 3 is a while-loop and its computational complexity is $O(numTPs)$. Variable $numTPs$ decides the number of test point candidates that will be selected randomly from an initial set of candidate list. The complexity of Step 4 (Section 4.2.1.1) is $O(VN + (L+P)V + L \times P_{FO}^{l})$. The complexity of Step 5 is (Section 4.2.2.1) is $O(VN(L+1)+L)$. Step 7 locates OPIs for

Algorithm 4.1 – The First Test Point Insertion Algorithm – TPI1.

Input: Scanned version of the circuit netlist.
Output: Modified scanned circuit netlist with test points inserted.

SPARTAN_TPI1(SCANNED CIRCUIT)
{
1. Compute *median* of SCANNED CIRCUIT;
2. Use *level_range* to compute range of levels in SCANNED CIRCUIT and populate candidate list L with logic gates within this range of levels;
3. Randomly select *numTPs* test point candidates from L;
4. Perform spectral analysis procedure on L (Section 4.2.1);
5. Perform entropy analysis procedure on L (Section 4.2.2);
6. Keep candidate lines selected by spectral and entropy analyses and the remaining candidate lines are discarded from L;
7. Perform OPI analysis on remaining candidate list L;
8 Insert test points on candidate lines remaining in L;
}

a candidate line l. This involves a forward *depth-first search* (DFS) from the candidate line l to POs to locate all of the lines that need to be controlled to propagate the fault effect at line l to be observed at a PO (the off-path lines). Assume that there are $L^l_{off-path}$ lines for candidate line l. The complexity to locate all off-path lines will be $O(N+L_{total})$. Then a backward DFS is performed for each of the off-path lines to locate all of the PIs that need to specified to control the off-path lines. The complexity will be $O(L^l_{off-path}(N+L_{total}))$. This step has to carried out for each candidate line, so the total complexity of the OPI analysis will be $O(L(L^l_{off-path}(N+L_{total})))$. The total complexity of TPI1 is $O((VN+(L+P)V+L\times P^l_{FO})+(VN(L+1)+L)+(L(L^l_{off-path}(N+L_{total}))))$. If $N\approx V$, $N>>L$, $L_{total}>>L$ and $L_{total}>>L^l_{off-path}$, the complexity will be $O(N^2+(L(L^l_{off-path}(N+L_{total}))))$.

4.3.2 The Second Test Point Insertion Algorithm – TPI2

Population of Initial Test Point Candidate List L

TPI2 first computes the variable *median*, which is half of the maximum number of levels in the circuit. The algorithm takes a user specified value, *level_range*, to find the gates in and around the middle of the circuit that will be considered as test point candidates, where $0 < level\_range < 1$. Variable *level_range* specifies the fraction of the number of levels succeeding and preceding the middle of the circuit from which the initial test point candidate lines are to be

selected. The variable *median* is used to locate the center of the circuit and test point candidates embedded in the circuit. Test point insertion in or near the center of the circuit is more effective because lines with low testability are usually embedded somewhere in the middle of the circuit. The variable *level_range* is used to expand the candidate selection from the circuit's center and to give the user the flexibility to choose the size of candidate set L. Ideally, we would want to try each and every line in the circuit, but this approach will be computationally infeasible for a large set L because we have to perform logic simulation each time we try a candidate test point in L.

The Location of *Observation Primary Inputs* (OPIs) – OPI Analysis

For each candidate line l in L, the algorithm locates the primary inputs that need to be specified to observe a fault effect on this line. Assume that a set of inputs I has to be specified, in order to control the logic gates in the shaded areas in Figure 4.6, to propagate any fault effect on line l to a PO. If we insert a test point at line l, then none of the inputs in I need to be specified to propagate the fault effects on l. Since adding a test point on l removes the requirement to specify the inputs in I, the test point on l can help reduce ATPG time and may reduce the test set length especially if the ATPG produces don't cares in the test patterns. The goal here is to select candidate lines with large OPI sets. Therefore, the algorithm first locates the OPIs for all lines in L. The algorithm then compares and locates the number of OPIs of each line in L that overlap with the OPIs of every other line in L. For example, let us consider two candidate lines l and l' with OPI sets I and I'. If the OPI set I' is a subset of OPI set I, then we remove l' from the candidate list. This is because l' is in the neighborhood of l and selecting test points that are close to each other does not cause a significant improvement in the testability of the circuit. Also, since l' is relieving a subset of PIs from being specified by the ATPG, selecting l' will not help reduce the ATPG time or test length. In the case that OPI set I' is not a subset of I, but has some overlap with I, the algorithm selects the candidate line with a larger OPI set and computes the amount of overlap between I and I'. If the smaller OPI set contains more than 50% overlap of the larger OPI set, the candidate line with the smaller OPI set is considered to be in the neighborhood of the candidate line with the larger OPI set. Thus, the candidate line with the smaller OPI set is discarded from L.

4.3.2.1 Spectral and Entropy Analysis

Finally, the algorithm performs the spectral analysis procedure (see Section 4.2.1) and the entropy analysis procedure (see Section 4.2.2) on the test point candidate lines in L. After the spectral and entropy analyses are performed, the candidate lines l in L selected by both are marked as good candidates for test point insertion and the rest of the candidate lines in L are discarded.

After the last step, the candidate lines remaining in L are selected for test point insertion. Algorithm 4.2 shows the overview of TPI2.

Algorithm 4.2 – The Second Test Point Insertion Algorithm – TPI2.

Input: Scanned version of the circuit netlist.
Output: Modified scanned circuit netlist with test points inserted.

SPARTAN_TPI2(SCANNED CIRCUIT)
{
1. Compute *median* of SCANNED CIRCUIT;
2. Use *level_range* to compute range of levels in SCANNED CIRCUIT and populate candidate list L with logic gates within this range of levels;
3. Perform OPI analysis;
4. Perform spectral analysis procedure on L(Section 4.2.1);
5. Perform entropy analysis procedure on L (Section 4.2.2);
6. Keep candidate lines selected by spectral and entropy analyses and the remaining candidate lines are discarded from L;
7. Insert test points on candidate lines remaining in L;
}

4.3.2.2 Computational Complexity of TPI2

The computational complexity of TPI2 will be identical to that of TPI1, which is $O(N^2 + (L(L^l_{offpath}(N + L_{total}))))$.

4.4 Summary

Two test point insertion algorithms, TPI1 and TPI2, have been proposed. The algorithms use spectral and entropy analysis to locate the best test point candidate lines in a circuit. It is an established fact that difficult to control lines in a circuit will have an adverse effect on the testability of the circuit. The first idea here is to add test points that greatly increase the

controllability and observability of the remaining circuit. Low toggling lines are difficult to control from PIs and spectral analysis helps the algorithm to find lines that have low activity. For observability analysis, the spectra of a line are compared to the spectra of the POs in the line's fanout cone. A line has good observability if its spectra are identical to the spectra of at least one PO in its fanout cone. The second idea involves adding test points on lines such that information flow or entropy of the circuit is improved. In other words, increasing information flow through the circuit improves circuit testability. If the amount of information passing through a line is high, the controllability and observability of the line will be high. The test point algorithms introduced here help us increase the overall testability if the circuit.

Chapter 5

SPARTAN Results

In this chapter we will present fault coverage, test length, test volume, and test application time results obtained for partial-scan and partial-scan with test point insertion via the TPI1 and TPI2 algorithms. All SPARTAN results are compared with the best partial-scan results reported by the authors of *mpscan* [78] and full-scan results reported by the authors of the combinational test pattern generator TRAN [14]. The results are compared with *mpscan* because its authors report the highest fault coverage results and the lowest test length, test volume, and test application times attained by any partial-scan algorithm. Similarly, SPARTAN results are compared with TRAN because TRAN is a commercially used combinational ATPG and its authors report full-scan results for the *ISCAS*-89 benchmarks. Section 5.2.1 provides the fault coverage results for partial-scan and partial-scan with test point insertion. Sections 5.2.2, 5.2.3, and 5.2.4 present test length, test volume, and test application time results for partial-scan and partial-scan with test point insertion. Section 5.3 presents the comparison of GATEST and HITEC test lengths, test volumes, and test application times. Sections 5.4 and 5.5 present the logic gate overhead and CPU time results for partial-scan and partial-scan with test point insertion.

5.1 Experimental Conditions

ATPGs and ATPG Parameters

SPARTAN results were generated using the sequential ATPG GATEST [61]. GATEST, being a non-deterministic ATPG, has a list of parameters that can be specified and varied by the user. The parameters used by GATEST for test generation are listed in Table 5.1. We also employ a *reverse order restoration* (ROR) compactor [24]. HITEC [51] is a deterministic sequential ATPG used to obtain *mpscan* results against which we compare. The backtrack limit is the only user-specified parameter for HITEC. The SPARTAN experimental results are generated by taking the benchmark circuits with DFT hardware added by SPARTAN and performing

Table 5.1: GATEST ATPG options and values used for *ISCAS*-89 benchmark test generation.

Option [Parameter]	Parameter Description	Parameter Value	Circuit(s)
-v [vectors]	Maximum number of vectors in a test sequence	200	s298, s344, s349, s382, s386, s400, s444, s526, s641, s713, s820, s953, s1423, s1488, s1494
		300	s5378
		500	s9234, s13207, s15850, s35932
		16	s38417
		18	s38584
-p [vec_prob]	Probability of including a fault in the fault sample during test vector generation	1.0	All benchmarks
-P [seq_prob]	Probability of including a fault in the fault sample during test sequence generation	1.0	All benchmarks
-g [num_generations]	Number of generations used in Genetic Algorithm	8	All benchmarks except s38417 and s38584
		16	s38417, s38584
-G [gen_gap]	Generation gap for Genetic Algorithms	1.0	All benchmarks
-i [init_vec]	Limit on number of vectors generated for initializing flip-flops	20	All benchmarks except s38417 and s38584
		200	s38471, s38584
-S [rand_seed]	Random seed	0	All benchmarks
-M [seq_mutation_rate]	Mutation rate during test sequence generation	1/64	All benchmarks

ATPG with GATEST on each of the benchmark circuits with the DFT hardware. The vectors generated by GATEST are then compacted using a ROR compactor [24]. The results in Tables 5.5, 5.6, and 5.8 reflect the compacted vector set for each circuit.

SPARTAN Partial-Scan Parameters

The partial-scan algorithm scans flip-flops in an iterative fashion. Each iteration selects a subset of flip-flops for scanning and then the algorithm proceeds to the subsequent iteration. The number of iterations for scan flip-flop selection can be varied. We used two, three, or four iterations for all our experiments. The number of iterations can be specified by the user. The parameters used during the spectral analysis procedure for the partial-scan algorithm are the transform matrix order *MATRIXSZ*, the *scanning threshold* (ST), and the number of random vectors generated, *SPEC_NUM_RAND_VEC* (used for logic simulation to obtain valid states of the circuit for spectral analysis). The circuit is first initialized to a random state and then logic simulation is performed to obtain the valid states of the circuit.

The spectral analysis is divided into two parts (see Section 3.1.1). The first part performs spectral analysis and identifies all of the potential scan flip-flop candidates. The spectral analysis is performed by analyzing states of the circuit with a *Rademacher-Walsh transform* (RWT). The transform order *MATRIXSZ* used is 4 ($2^4 \times 2^4$ RWT) and the states are obtained via logic simulation of 512 random vectors, thus *SPEC_NUM_RAND_VEC* is set to 512. An order 4 matrix is used primarily to reduce spectra computation time. Even though larger matrices will have more spectral coefficients and more information, they will also have larger spectra computation times. The second part of spectral analysis uses ST, a percentage of the initial scan flip-flop candidate list from the first part to scan, in order to compute the number of flip-flops that will be scanned. The values of ST can range from 0-100. For example, if ST is set to 10% and there are 100 flip-flops in the candidate list recommended by the first spectral analysis step, the total number of flip-flops finally scanned will be ≤ 10. Once the first step marks all of the potential scan flip-flop candidates, the second step visits the largest SCC in the circuit and scans all of the flip-flops marked (by the first step) for scanning. If the number of scanned flip-flops is still less than ST, the second largest SCC is visited and all of the marked flip-flops in the SCC are scanned. This proceeds until the number of scanned flip-flops is equal to ST. The default ST value is set to 25% for all of the circuits, which was determined empirically. We tried ST values ranging from 10% to 90%, but 25% gave a good result for all of the benchmark circuits so we

used ST = 25%. Table 5.2 describes the parameter settings for all experiments.

The parameter used for entropy analysis is *ENT_NUM_RAND_VEC*, which specifies the number of random vectors for logic simulation. The default value is 512. Logic simulation for entropy analysis is performed separately from the logic simulation performed for spectral analysis since for entropy analysis, logic simulation is performed for each candidate test point insertion line trial, whereas for spectral analysis, logic simulation is needed only once. Entropy analysis uses logic simulation to compute probability values, $p_i(0)$ and $p_i(1)$, for each flip-flop i. Logic simulation is also used to compute conditional probabilities, $p_i(D|PI_j)$ and $p_i(Q|PO_k)$, to compute conditional entropy values for each flip-flop i (see Section 3.1.2). Here $p_i(D|PI_j)$ is the probability of a logic 0 or logic 1 occurring at the D-line of flip-flop i given the value on the j^{th} PI in its fanin cone. Similarly, $p_i(Q|PO_k)$ is the probability of a logic 0 or logic 1 occurring at the Q-line of flip-flop i given the value on the k^{th} PO in its fanout cone. The circuit is first initialized to a random state before logic simulation. The probability values are then used to compute the entropy and conditional entropy values (see Section 3.1.2). The parameter values used for partial-scan and test point insertion are shown in Table 5.2.

SPARTAN Test Point Insertion Parameters – TPI1

The spectral analysis is performed using a 16×16 RWT. Logic simulation for spectral analysis is performed using 512 random vectors. For entropy analysis, logic simulation is performed to compute the probability values of each line in the circuit with 512 vectors. Since logic simulation is performed each time a test point candidate is tried, logic simulation with 512 random vectors is performed to keep the computation times reasonable.

TPI1 takes a user specified value for parameter *level_range*, where $0 < level\_range < 1$. The parameter *level_range* specifies the fraction of the number of levels succeeding and preceding the middle of the circuit from which the initial test point candidate lines are selected to populate the initial candidate list. We used value of 0.99 for *level_range*, which is the default value as well. The parameter *numTPs* decides the number of random test point candidates to be picked from the initial candidate list. The default value is 500 and we used this value in our experiments.

TPI2 takes a user specified value for parameter *level_range*, where $0 < level\_range < 1$. The parameter *level_range* specifies the fraction of levels succeeding and preceding the middle

of the circuit from which the initial test point candidate lines are selected to populate the initial candidate list. We used the value of 0.0625 for *level_range*, which is the default value as well.

Table 5.2: SPARTAN parameters and values used for *ISCAS*-89 benchmark scanning and test point insertion.

Partial-Scan		
Parameter	Parameter Value	Value Used on Circuit(s)
# Iterations	1	s298, s400, s641, s713, s820, s953
	2	s344, s349, s382, s386, s444, s526, s1488, s5378, s35932
	3	s1494, s9234, s13207, s15850, s38584
	4	s1423, s38417
MATRIXSZ	4	All benchmarks
Scanning Threshold (ST)	25%	All benchmarks
SPEC_NUM_RAND_VEC	512	All benchmarks
ENT_NUM_RAND_VEC	512	All benchmarks
TPI1		
Parameter	Parameter Value	Value Used on Circuit(s)
level_range	0.99	All benchmarks
numTPs	500	All benchmarks
SPEC_NUM_RAND_VEC	512	All benchmarks
ENT_NUM_RAND_VEC	512	All benchmarks
TPI2		
Parameter	Parameter Value	Value Used on Circuit(s)
level_range	0.0625	All benchmarks except s15850
	0.25	s15850
SPEC_NUM_RAND_VEC	512	All benchmarks
ENT_NUM_RAND_VEC	512	All benchmarks

5.2 Results

SPARTAN results are presented in this section. Section 5.2.1 presents the fault coverage results for SPARTAN partial-scan, SPARTAN partial-scan with test point insertion, *mpscan* [78], and TRAN [14]. Section 5.2.2 presents the test length results for SPARTAN, *mpscan* [78], and TRAN [14]. SPARTAN uses the GATEST [61] ATPG for test generation, *mpscan* uses HITEC [51], and TRAN results are reported using a combinational ATPG, TRAN [14]. Section 5.2.3 reports the test volume results and Section 5.2.4 report the test application time results. GATEST uses the PROOFS fault simulator [52] to determine the fault detected by the test set generated. All of the fault coverage results produced by GATEST were verified by the HOPE fault simulator [45]. Table 5.3 shows the number of primary inputs and the number of flip-flops

in the benchmark circuits.

Table 5.3: Numbers of *primary inputs* (PIs) and *flip-flops* (FFs) in the *ISCAS*-89 benchmarks.

Circuit	PIs	FFs
s298	3	14
s344	9	15
s349	9	15
s382	3	21
s386	7	6
s400	4	21
s444	3	21
s526	3	21
s641	35	19
s713	35	21
s820	18	5
s953	16	29
s1423	17	74
s1488	8	6
s1494	8	6
s5378	35	179
s9234	19	228
s13207	31	669
s15850	77	534
s35932	35	1728
s38417	28	1636
s38584	12	1452

5.2.1 Fault Coverage Results

This section presents the fault coverage results for the *ISCAS*-89 benchmark circuits. Columns 2 and 3 in Table 5.4 report the results with partial-scan only, columns 4 and 5 in Table 5.4 report the results with the partial-scan and test point combination algorithm PS+TPI1, and columns 6 and 7 in Table 5.4 report the results with the partial-scan and test point combination algorithm PS+TPI2. Columns 8 and 9 in Table 5.4 are the partial-scan results reported for *mpscan* [78]. Column 10 in Table 5.4 is the full-scan results reported for TRAN [14]. In Table 5.4, the columns under PS report the the number of *scan flip-flops* (SFFs) and the partial-scan *fault coverage* (FC). The columns PS+TPI1 and PS+TPI2 report the number of *complete test points* (CTPs) and the FC results. To obtain the results for the partial-scan and *test point insertion* (TPI) combination, either the TPI algorithm TPI1 or TPI2 is applied to the scanned benchmark circuits produced by the SPARTAN PS algorithm. Results in columns 3, 5, and 7 of Table 5.4 were obtained with GATEST test vectors. In Table 5.4, columns 8 and 9 are the results

Table 5.4: *ISCAS*-89 benchmark fault coverage results for SPARTAN with TPI1 (Spectral and Entropy) vs. *mpscan* [78] and TRAN [14].

Ckt.	FFs	SPARTAN						*mpscan* [78]		TRAN [14]
		PS		PS+TPI1		PS+TPI2				
		SFF	FC (%)	CTP	FC (%)	CTP	FC (%)	SFF	FC (%)	FC (%)
s298	14	11	99.67	5	99.68	4	99.68	2	98.70	100.00
s344	15	7	99.71	1	99.71	3	99.71	3	99.40	100.00
s349	15	8	99.14	27	99.75	4	99.16	3	98.90	99.43
s382	21	17	99.00	1	99.00	16	99.54	6	99.00	100.00
s386	6	4	99.48	25	100.00	10	100.00	4	100.00	100.00
s400	21	16	94.60	1	94.63	16	95.42	4	95.80	98.59
s444	21	18	96.84	1	95.38	19	98.05	6	96.20	97.05
s526	21	19	99.64	1	99.82	3	99.82	10	99.30	99.82
s641	19	9	99.14	40	99.64	6	99.58	1	99.40	100.00
s713	21	9	92.94	39	95.14	6	93.09	1	92.90	93.46
s820	5	2	99.76	187	100.00	48	100.00	2	100.00	100.00
s953	29	23	100.00	3	100.00	3	100.00	3	100.00	100.00
s1423	74	63	98.75	19	98.78	3	98.75	41	98.10	99.08
s1488	6	5	100.00	389	100.00	58	100.00	2	100.00	100.00
s1494	6	4	99.20	399	100.00	26	99.42	2	99.10	99.20
s5378	179	114	98.81	2	98.87	44	98.78	50	97.20	99.13
s9234	228	199	93.32	7	93.27	102	94.81	97	93.00	93.47
s13207	669	496	97.77	21	98.02	20	97.82	58	85.60	98.46
s15850	534	407	93.08	30	92.74	7	93.33	180	94.80	96.68
s35932	1728	477	89.80	27	89.92	18	89.83	150	89.80	89.81
s38417	1636	1249	98.44	27	98.48	201	98.82	400	94.50	99.47
s38584[a]	1452	924	94.54	24	94.38	87	94.25	-	-	95.85
Avg.			97.44		97.60		97.72		-	98.16
Avg. w/o s38584			97.58		97.75		97.89		96.75	98.27

[a]The authors of *mpscan* did not provide results for s38584.

for *mpscan* and column 10 shows the TRAN results. TRAN with FS has the best FC results (98.16%) followed by SPARTAN with PS and TPI2 (97.72%), SPARTAN with PS and TPI1 (97.60%), SPARTAN with only PS (97.44%), and *mpscan* (96.75%). So, all of the SPARTAN algorithms perform better than *mpscan*, the best PS algorithm, and perform almost as well as TRAN with full-scan.

5.2.2 Test Length Results

Table 5.5: *ISCAS*-89 benchmark *test length* (TL) results for SPARTAN (partial-scan and partial-scan with TPI) vs. *mpscan* [78] and TRAN [15].

Ckt.	FFs	SPARTAN						*mpscan* [78]		TRAN [15]	
		PS		PS+TPI1		PS+TPI2					
		SFF	TL	CTP	TL	CTP	TL	SFF	TL	SFF	TL
s298	14	11	37	5	36	4	41	2	160	14	55
s344	15	7	65	1	63	3	50	3	141	15	36
s349	15	8	50	27	22	4	33	3	161	15	36
s382	21	17	215	1	164	16	93	6	168	21	54
s386	6	4	119	25	89	10	94	4	205	6	100
s400	21	16	236	1	252	16	79	4	394	21	52
s444	21	18	158	1	116	19	91	6	193	21	49
s526	21	19	131	1	112	3	105	10	273	21	116
s641	19	9	107	40	108	6	115	1	286	19	101
s713	21	9	125	39	105	6	130	1	310	21	102
s820	5	2	349	187	93	48	171	2	602	5	196
s953	29	23	98	3	90	3	90	3	339	29	132
s1423	74	63	131	19	68	3	95	41	397	74	130
s1488	6	5	136	389	56	58	131	2	639	6	214
s1494	6	4	179	399	66	26	191	2	622	6	213
s5378	179	114	535	2	556	44	517	50	1023	179	432
s9234	228	199	958	7	717	102	860	97	3114	228	666
s13207	669	496	879	21	875	20	1044	58	9805	669	744
s15850	534	407	3906	30	661	7	832	180	4527	534	722
s35932	1728	477	261	27	166	18	150	150	252	1728	79
s38417	1636	1249	3113	27	3066	201	3433	400	11573	1636	1601
s38584[a]	1452	924	1491	24	1460	87	1449	-	-	1452	1240
Avg.			749		406		445		-		321
Avg. w/o s38584			714		356		397		1675		278

[a]The authors of *mpscan* did not provide results for s38584.

Results in columns 3, 5, and 7 of Table 5.5 were obtained with GATEST test vectors compacted with a ROR compactor [24]. Table 5.5 shows that TRAN has the shortest average vector length (321 vectors) followed by SPARTAN partial-scan with TPI1 (406 vectors), SPARTAN

partial-scan with TPI2 (445 vectors), and SPARTAN with only partial-scan (749 vectors). Algorithm *mpscan* has the highest average test length (1675 vectors).

5.2.3 Test Volume Results

Table 5.6: *ISCAS*-89 benchmark *test volume* (TV) results for SPARTAN (partial-scan and partial-scan with TPI) vs. *mpscan* [78] and TRAN [14].

Ckt.	FFs	SPARTAN						*mpscan* [78]		TRAN [15]
		PS		PS+TPI1		PS+TPI2				
		SFF	TV	CTP	TV	CTP	TV	SFF	TV	TV
s298	14	11	555	5	684	4	738	2	800	990
s344	15	7	1105	1	1070	3	950	3	1692	900
s349	15	8	900	27	968	4	693	3	1932	900
s382	21	17	4515	1	3444	16	3348	6	1512	1350
s386	6	4	1428	25	3204	10	1974	4	2255	1400
s400	21	16	4956	1	5292	16	2844	4	3152	1352
s444	21	18	3476	1	2552	19	3640	6	1737	1225
s526	21	19	3013	1	2576	3	2625	10	3549	2900
s641	19	9	4815	40	9072	6	5750	1	10295	5555
s713	21	9	5625	39	8715	6	6500	1	11160	5814
s820	5	2	7329	187	19251	48	11628	2	12040	4704
s953	29	23	3920	3	3780	3	3780	3	6441	5940
s1423	74	63	10611	19	6732	3	7885	41	23026	11960
s1488	6	5	1904	389	22512	58	9301	2	6390	3210
s1494	6	4	2327	399	27126	26	7258	2	6220	3195
s5378	179	114	80250	2	83956	44	99781	50	86955	92880
s9234	228	199	209802	7	161325	102	275200	97	361224	165168
s13207	669	496	464112	21	479500	20	571068	58	872645	521544
s15850	534	407	1894410	30	339754	7	408512	180	1163439	441864
s35932	1728	477	133893	27	89474	18	79500	150	46620	139356
s38417	1636	1249	3978414	27	3998064	201	5073974	400	4953244	2664064
s38584[a]	1452	924	1397067	24	1401600	87	1482327	-	-	1815360
Avg.			405258		303212		366331		-	267937
Avg. w/o s38584			358029		250907		313188		360778	194191

[a]The authors of *mpscan* did not provide results for s38584.

Results in columns 3, 5, and 7 of Table 5.6 were obtained with GATEST test vectors compacted with a ROR compactor [24]. Test volume is the total number of bits stored in the external tester that will be fed into the circuit being tested. The total number of bits applied to a circuit per test vector is the sum of bits applied to each primary input and the number of bits shifted into the scan chain. Parallel vectors are vector sequences generated by the sequential ATPG program with SFFs as PPIs and PPOs. Then these parallel test sequences are serialized according to the scan structure and the final serialized test vectors are developed. Each serialized vector

includes the test vector that will be applied to the PIs and the scan-in and scan-out sequences for each parallel test sequence generated by the ATPG. The total number of bits applied to the primary inputs will be n_{PI}, where n_{PI} is the number of primary inputs. Similarly, the total number of bits shifted into the partial-scan chain is n_{SFF}, where n_{SFF} is the length of the scan chain. Therefore, for partial-scan, the total number of bits per vector will be $(n_{PI} + n_{SFF})$ and the total number of bits for the complete test set with $n_{vectors}$ will be $n_{vectors} \times (n_{PI} + n_{SFF} + 1)$ (see Equation 5.1). For the partial-scan and test point insertion combination, the test volume is computed with Equation 5.2. The TV results obtained using Equations 5.1 and 5.2, where $n_{vectors}$ is the total number of test vectors, n_{PI} is the number of primary inputs, n_{SFF} is the number of scan flip-flops, and n_{CTP} is the number of complete test points, are shown in Table 5.6. The 1 in Equations 5.1 and 5.2 accounts for the test mode selection bit. TRAN has the lowest test volume compared to all of the other DFT techniques. The averages for the SPARTAN algorithms are higher than *mpscan* and TRAN because of s38417. SPARTAN algorithms do not perform very well on benchmark s38417 and the high test volume is skewing the average. TRAN's average test volume for all circuits except s38417 is 153759, SPARTAN's average test volume with partial-scan only is 235107, SPARTAN partial-scan with TPI1 is 127266, and SPARTAN partial-scan with TPI2 is 142157. SPARTAN with TPI1's average test volume (excluding s38417) and SPARTAN with TPI2's average test volume (excluding s38417) is lower than full-scan with TRAN's average test volume (excluding s38417). Table 5.7 shows the test volume reduction by each of the SPARTAN algorithms compared with TRAN. Equation 5.3 is used to compute the percent reduction in test volume, where TV_{TRAN} is the test volume for TRAN and $\mathrm{TV}_{SPARTAN}$ is the test volume for a SPARTAN (PS, PS+TPI1 or PS+TPI2) algorithm. A negative value means that the test volume for the SPARTAN algorithm is larger than that of TRAN for the particular circuit.

$$\text{Test Volume (PS)} = n_{vectors} \times (n_{PI} + n_{SFF} + 1) \tag{5.1}$$

$$\text{Test Volume (PS+TPI)} = n_{vectors} \times (n_{PI} + n_{SFF} + n_{CTP} + 1) \tag{5.2}$$

$$\text{TV Reduction}(\%) = \frac{\mathrm{TV}_{TRAN} - \mathrm{TV}_{SPARTAN}}{\mathrm{TV}_{TRAN}} \times 100\% \tag{5.3}$$

Table 5.7: Percentage of test volume reduction by SPARTAN (partial-scan and partial-scan with TPI) for *ISCAS*-89 benchmarks compared to full-scan with TRAN [14].

Ckt.	SPARTAN % TV Reduction		
	PS	PS+TPI1	PS+TPI2
s5378	13.60	9.61	-7.43
s9234	-451.58	2.33	-66.62
s13207	11.01	8.06	-9.50
s15850	-328.73	23.11	7.55
s35932	3.92	35.79	42.95
s38417	-49.25	-49.98	-90.35
s38584	23.09	22.84	18.40

Table 5.8: *ISCAS*-89 benchmark test application time results for SPARTAN (partial-scan and partial-scan with TPI) vs. *mpscan* [78] and TRAN [14].

Ckt.	FFs	SPARTAN						*mpscan* [78]		TRAN [15]
		PS		PS+TPI1		PS+TPI2				
		SFF	TAT (clocks)	CTP	TAT (clocks)	CTP	TAT (clocks)	SFF	TAT (clocks)	TAT (clocks)
s298	14	11	470	5	648	7	690	2	488	857
s344	15	7	538	1	587	3	574	3	574	610
s349	15	8	470	27	866	4	457	3	654	610
s382	21	17	3908	1	3156	16	3232	6	1192	1234
s386	6	4	607	25	2732	10	1442	4	1037	176
s400	21	16	4048	1	4574	16	2675	4	1982	1190
s444	21	18	3042	1	2362	19	3536	6	1367	1124
s526	21	19	2662	1	2396	3	2463	10	3027	2598
s641	19	9	1092	40	5502	6	1874	1	578	2062
s713	21	9	1272	39	5245	6	2114	1	626	2290
s820	5	2	1055	187	18052	48	8825	2	1814	1190
s953	29	23	2402	3	2486	3	2486	3	1366	4022
s1423	74	63	8514	19	5812	3	6501	41	16760	9902
s1488	6	5	830	389	22912	58	8514	2	1925	1514
s1494	6	4	907	399	27474	26	5985	2	1874	1507
s5378	179	114	61757	2	65288	44	82523	50	52277	78122
s9234	228	199	192002	7	148835	102	260326	97	305370	152974
s13207	669	496	437859	21	454288	20	540784	58	578615	499822
s15850	534	407	1594466	30	290396	7	346112	180	819751	387342
s35932	1728	477	125716	27	84842	18	75394	150	38356	140051
s38417	1636	1249	3893752	27	3917838	201	4984187	400	4641577	2624113
s38584[a]	1452	924	1381027	24	1387440	87	1468414	-	-	1804628
Avg.			379945		293351		354960		-	259931
Avg. w/o s38584			332275		241252		301938		308152	186374

[a]The authors of *mpscan* did not provide results for s38584.

5.2.4 Test Application Time Results

Results in columns 3, 5, and 7 of Table 5.8 were obtained with GATEST test vectors compacted with a ROR compactor [24]. Equation 5.4 is used to compute the test application time for partial-scan only and Equation 5.5 is used for partial-scan with test point insertion [13]. Here $n_{vectors}$ is the total number of test vectors, n_{SFF} is the number of scan flip-flops and n_{CTP} is the number of complete test points. TRAN has the lowest average test application time compared to all of the other DFT techniques. The averages for the SPARTAN algorithms are higher than for *mpscan* and TRAN because of s38417. SPARTAN algorithms do not perform very well on benchmark s38417 and the high test application time for s38417 is skewing the average. TRAN's average test volume for all circuits except s38417 is 147351, SPARTAN's average test volume with partial-scan only is 212621, SPARTAN with TPI1 is 120757 and SPARTAN with TPI2 is 134520. SPARTAN with TPI1's average test application time (excluding s38417) and SPARTAN with TPI2's average test application time (excluding s38417) are lower than full-scan with TRAN's average test application time (excluding s38417). Table 5.9 shows the test volume reduction by each of the SPARTAN algorithms compared with TRAN. Equation 5.6 is used to compute the percent reduction in test application time, where TAT_{TRAN} us the test application time for TRAN and $\text{TAT}_{SPARTAN}$ is the test application time for a SPARTAN (PS, PS+TPI1, or PS+TPI2) algorithm. A negative value means that the test application time for the SPARTAN algorithm is larger than that of TRAN for the particular circuit.

$$\text{Test application time (PS)} = (n_{vectors}+2)n_{SFF}+n_{vectors}+4 \text{ clock periods} \tag{5.4}$$

$$\text{Test application time(PS+TPI)} = (n_{vectors}+2)(n_{SFF}+n_{CTP})+n_{vectors}+4 \text{ clock periods} \tag{5.5}$$

$$\text{TAT Reduction}(\%) = \frac{\text{TAT}_{TRAN}-\text{TAT}_{SPARTAN}}{\text{TAT}_{TRAN}} \times 100\% \tag{5.6}$$

5.3 GATEST vs. HITEC

In this section the results obtained by GATEST and HITEC obtained for some large circuits are discussed. The results show that deterministic test generators such as HITEC do better then simulation-based test generators such as GATEST for highly sequential circuits. Adding DFT hardware to a sequential circuit increases its testability and reduces its sequential nature. With DFT hardware added, the average fault coverage results obtained by GATEST are almost

Table 5.9: Percentage of test application time reduction by SPARTAN (partial-scan and partial-scan with TPI) for *ISCAS*-89 benchmarks compared to TRAN [14].

Ckt.	SPARTAN % TAT Reduction		
	PS	PS+TPI1	PS+TPI2
s5378	20.95	16.43	-5.63
s9234	-444.15	2.71	-70.18
s13207	12.40	9.11	-8.20
s15850	-311.64	25.03	10.64
s35932	10.24	39.42	46.17
s38417	-48.38	-49.30	-89.94
s38584	23.47	23.12	18.63

identical to HITEC's fault coverages. GATEST managed to obtain much shorter test lengths than HITEC. Similarly, the test volume and test application results for GATEST are much lower than those for HITEC.

5.3.1 Fault Coverage Results

The fault coverage results obtained with GATEST and HITEC are shown in Table 5.10. For all the DFT techniques, GATEST gets an equal or a better fault coverage result than HITEC for the two largest circuits, s35932 and s38417. The average fault coverage result for the seven benchmarks, HITEC does marginally better than GATEST for all the DFT techniques.

Table 5.10: *ISCAS*-89 benchmark fault coverage results for SPARTAN (partial-scan and partial-scan with TPI) with GATEST vs. HITEC.

Ckt.	GATEST FC (%)			HITEC FC (%)		
	PS	PS+TPI1	PS+TPI2	PS	PS+TPI1	PS+TPI2
s5378	98.81	98.87	98.79	98.96	99.00	99.08
s9234	93.32	93.27	94.81	93.40	93.42	94.88
s13207	97.77	98.02	97.82	98.00	98.23	97.64
s15850	93.08	92.74	92.33	95.10	95.37	95.09
s35932	89.80	89.92	89.83	89.80	89.91	89.83
s38417	98.44	98.48	98.82	96.40	98.16	98.27
Avg.	95.20	95.22	95.40	95.28	95.68	95.80

5.3.2 Test Length Results

The test lengths obtained by GATEST and ROR compaction are much shorter than test lengths obtained by HITEC. Table 5.11 shows the test length results for GATEST and HITEC. GATEST

gets the shortest average test length for SPARTAN partial-scan with TPI2 (1007 vectors) and SPARTAN partial-scan with TPI1 gets the next best result (1139 vectors).

Table 5.11: *ISCAS*-89 benchmark test length results for SPARTAN (partial-scan and partial-scan with TPI) with GATEST vs. HITEC.

Ckt.	GATEST TL			HITEC TL		
	PS	PS+TPI1	PS+TPI2	PS	PS+TPI1	PS+TPI2
s5378	535	556	517	822	686	632
s9234	958	717	860	958	847	850
s13207	879	875	1044	1823	1705	2212
s15850	3906	661	832	1383	1190	1219
s35932	261	166	150	165	214	175
s38417	3113	3066	3433	2885	3927	4168
Avg.	2142	1139	1007	1339	1428	1543

5.3.3 Test Volume Results

The test volume results for GATEST and HITEC are shown in Table 5.12. GATEST gets the lowest test volume for SPARTAN partial-scan with TPI1 (858679 bits). GATEST had a lower test volume for SPARTAN partial-scan with TPI2 (1084673 bits) compared to HITEC for SPARTAN partial-scan with TPI2 (1410797 bits).

Table 5.12: *ISCAS*-89 benchmark test volume results for SPARTAN (partial-scan and partial-scan with TPI) with GATEST vs. HITEC.

Ckt.	GATEST TV (bits)			HITEC TV (bits)		
	PS	PS+TPI1	PS+TPI2	PS	PS+TPI1	PS+TPI2
s5378	80250	83956	99781	123300	104272	122608
s9234	911040	161325	275200	209802	191422	272850
s13207	464112	479500	571068	962544	936045	1212176
s15850	1894410	339754	408512	670755	612850	599748
s35932	133893	89474	79500	84645	115560	92925
s38417	3978414	3998064	5073974	3687030	5124735	6164472
Avg.	1243687	858679	1084673	956346	1180814	1410797

5.3.4 Test Application Time Results

The test application time results for GATEST and HITEC are shown in Table 5.13. GATEST gets the lowest test application time for SPARTAN partial-scan with TPI1 (826915 clock cycles). GATEST had a lower average test application time for SPARTAN partial-scan with TPI2

(1048221 clock cycles) compared to HITEC for SPARTAN partial-scan with TPI2 (1357990 clock cycles).

Table 5.13: *ISCAS*-89 benchmark test application time results for SPARTAN (partial-scan and partial-scan with TPI) with GATEST vs. HITEC.

Ckt.	GATEST TAT (clock cycles)			HITEC TAT (clock cycles)		
	PS	PS+TPI1	PS+TPI2	PS	PS+TPI1	PS+TPI2
s5378	61757	65822	82523	94762	80498	100808
s9234	832402	148835	260326	192002	175745	257306
s13207	437859	454288	540784	907027	884228	1144640
s15850	1594466	290396	346112	565082	522098	506717
s35932	125716	84842	75394	79828	109082	87794
s38417	3893752	3917838	4984187	3608752	5017335	6050672
Avg.	1157659	826915	1048221	907909	1131498	1357990

5.4 DFT Area Overhead

Results in columns 3, 5, and 7 of Table 5.14 for SPARTAN, *mpscan*, and TRAN report the logic gate DFT overhead for each of the methods. The logic gate overhead for SPARTAN with partial-scan is calculated using Equation 5.7.

$$\text{Gate overhead of partial-scan} = \frac{4 \times n_{SFF}}{n_{gates} + 10 n_{FF}} \times 100\% \tag{5.7}$$

Here n_{SFF} is the number of scan flip-flops, n_{gates} is the number of gates in the circuit, and n_{FF} is the number of flip-flops in the circuit. A multiplexer is added to convert a regular flip-flop into a scan flip-flop, which is 4 extra logic gates shown in the Equation 5.7 numerator. Similarly, the overheads in columns 5 and 7 are calculated using Equation 5.8.

$$\text{Gate overhead of partial-scan and TPI} = \frac{4 \times n_{SFF} + 18 \times n_{CTP}}{n_{gates} + 10 n_{FF}} \times 100\% \tag{5.8}$$

In Equation 5.8, n_{SFF} is the number of scan flip-flops, n_{CTP} is the number of complete test points, n_{gates} is the number of gates in the circuit, and n_{FF} is the number of flip-flops in the circuit. A multiplexer is added to convert a regular flip-flop into a scan flip-flop, which is 4 extra logic gates. A CTP is an extra scan flip-flop added to control and observe a given line, each of which will add 14 extra logic gates. All of these overhead logic gates are accounted for in the numerator of Equation 5.8. The logic gate overhead for *mpscan* is calculated using

Equation 5.7 and the logic gate overhead of TRAN is computed using Equation 5.9.

$$\text{Gate overhead for full-scan} = \frac{4 \times n_{FF}}{n_{gates} + 10n_{FF}} \times 100\% \tag{5.9}$$

Table 5.14 shows that *mpscan* has the lowest overhead (4.10%) given that it is a purely partial-scan technique. SPARTAN with partial-scan only is next with an overhead of 10.99%. TRAN has an gate overhead of 15.98%. SPARTAN with TPI1 and SPARTAN with TPI2 have higher overheads because of the extra test points added into the circuits since each test point incurs 14 gates of overhead compared to 4 gates to convert a flip-flop into a scan flip-flop. The gate overheads in columns 5 (PS+TPI1) and 7 (PS+TPI2) of Table 5.14 include overhead gates for partial-scan and complete test points. SPARTAN with TPI2 has an overhead of 35.78% and SPARTAN with TPI1 has an overhead of 104.43%. Most of the circuits with SPARTAN partial-scan and TPI1 have a reasonable gate overhead. The average is skewed by three circuits s820, s1488, and s1494. These circuits are small circuits and adding any extra hardware has an exponential increase in the gate overhead of these circuits. For the two largest circuits (s38417 and s38584), the overheads for SPARTAN partial-scan only, SPARTAN partial-scan with TPI1, and SPARTAN partial-scan with TPI2 are lower than for TRAN with full-scan except for SPARTAN partial-scan with TPI2 for s38417. It has a slightly higher gate overhead (18.18%) than TRAN with full-scan (16.98%). Figure 5.1 shows the logic gate overhead for the 7 largest benchmark circuits. In Figure 5.1, s5378 PS and PS+TPI1 have a lower gate overhead than full-scan (TRAN) and PS+TPI2 has a slightly higher overhead than full-scan (TRAN). For s9234, PS and PS+TPI1 have a lower gate overhead than full-scan (TRAN) and PS+TPI2 has a higher overhead than full-scan (TRAN). For the other circuits, except s38417, all of the SPARTAN approaches have a lower overhead than full-scan (TRAN). In s38417, PS+TPI2 has a slightly higher gate overhead than full-scan (TRAN).

5.5 CPU Times

Table 5.15 reports the CPU times for computing which flip-flops in the circuit to scan and locate the best candidates for test point insertion. Column 2 shows the CPU time required to find the scan flip-flops in a circuit. Column 3 in Table 5.15 shows the time required by TPI1 for locating the best test point insertion candidates. Column 3 does not include the time for partial-scan. Similarly, column 4 in Table 5.15 shows the time required by TPI2 for locating the best test point insertion candidates. SPARTAN partial-scan with TPI1 has the highest CPU time (28585

Table 5.14: *ISCAS*-89 benchmark *logic gate overhead* (LGO) for SPARTAN (partial-scan and partial-scan with TPI) vs. *mpscan* [78] and TRAN [14].

Ckt.	FFs	SPARTAN						*mpscan* [78]		TRAN [15]
		PS		PS+TPI1		PS+TPI2				
		SFF	LGO (%)	CTP	LGO (%)	CTP	LGO (%)	SFF	LGO (%)	LGO (%)
s298	14	11	16.99	5	51.74	7	44.79	2	3.09	21.62
s344	15	7	9.03	1	14.84	3	26.45	3	3.87	19.35
s349	15	8	10.29	27	166.56	4	33.44	3	3.86	19.29
s382	21	17	18.48	1	23.37	16	96.74	6	6.52	22.83
s386	6	4	7.31	25	212.79	10	89.50	4	7.31	10.96
s400	21	16	17.11	1	21.93	16	94.12	4	4.28	22.46
s444	21	18	18.41	1	23.02	19	105.88	6	6.14	21.48
s526	21	19	18.86	1	23.33	3	32.26	10	9.93	20.84
s641	19	9	6.33	40	132.86	6	25.31	1	0.70	13.36
s713	21	9	5.99	39	122.80	6	23.96	1	0.67	13.98
s820	5	2	2.36	187	995.28	48	257.23	2	2.36	5.90
s953	29	23	13.43	3	21.31	3	21.31	3	1.75	16.93
s1423	74	63	18.04	19	42.52	3	21.90	41	11.74	21.19
s1488	6	5	2.81	389	984.85	58	149.23	2	1.12	3.37
s1494	6	4	2.26	399	1018.10	26	68.46	2	1.13	3.39
s5378	179	114	9.98	2	10.77	44	27.31	50	4.38	15.67
s9234	228	199	10.10	7	11.70	102	33.41	97	4.92	11.58
s13207	669	496	13.55	21	16.13	20	16.01	58	1.58	18.28
s15850	534	407	10.77	30	14.35	7	11.61	180	4.76	14.13
s35932	1728	477	5.72	27	7.18	18	6.69	150	1.80	20.73
s38417	1636	1249	12.96	27	14.22	201	22.35	400	4.15	16.98
s38584[a]	1452	924	10.94	24	12.22	87	15.58	-	-	17.20
Avg.			10.99		179.18		55.62		-	15.98
Avg. w/o s38584			10.99		187.13		57.52		4.10	15.92

[a]The authors of *mpscan* did not provide results for s38584.

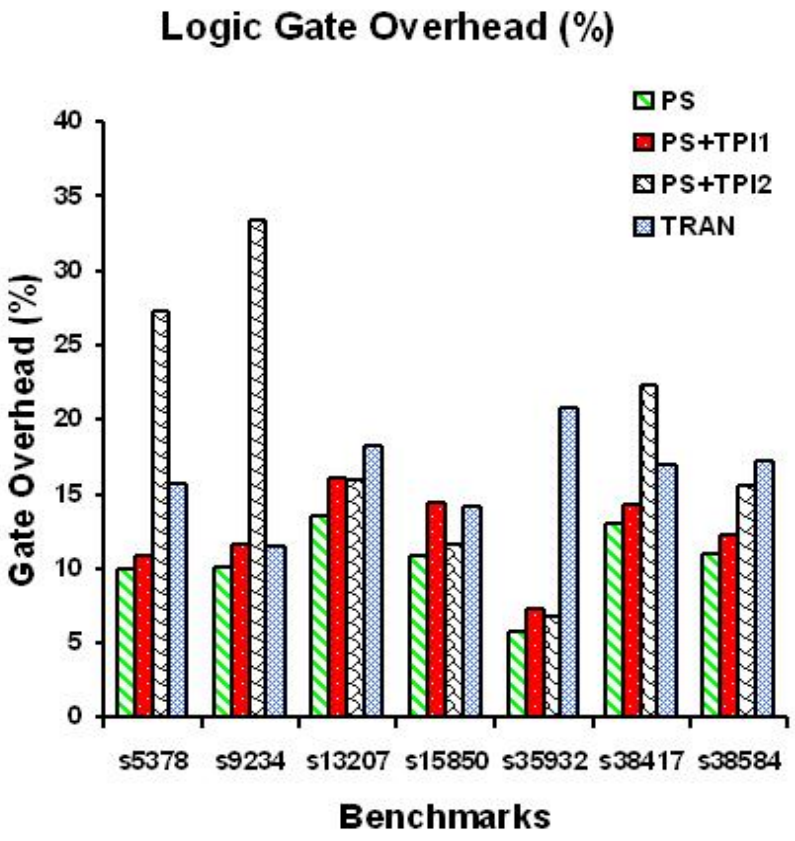

Figure 5.1: SPARTAN and TRAN logic gate overhead.

seconds) because the algorithm first randomly selects a fixed number of candidate test points (500 for our experiments) and tries each of the randomly selected lines. SPARTAN partial-scan with TPI2 first selects initial candidate lines and weeds out the lines in the initial test set via overlap analysis. This helps reduce the size of the candidate list and therefore a smaller list of candidate lines needs to be tried. Therefore, SPARTAN partial-scan with TPI2 has a lower CPU time than SPARTAN partial-scan with TPI1. CPU times for SPARTAN partial-scan with TPI1 and SPARTAN partial-scan with TPI2 are the times for test point insertion only. The CPU times for the larger circuits are lengthy because the entropy analysis for partial-scan and test point insertion assimilates logic simulation to compute the probabilities for entropy calculation. Logic simulation is performed for each candidate scan flip-flop and candidate test point line. For example, assume that there are 500 candidate scan flip-flops in a circuit. Logic simulation is performed when each candidate scan flip-flop is tried for scanning to compute new entropy values. In this case 500 logic simulations will be performed for the circuit, once for each candidate scan flip-flop. We use logic simulation because topology-based testability analysis such as SCOAP [30] and probability-based testability measures [57] are *static*, in the sense that they do not use input test patterns for testability analysis. The topology-based testability analysis is efficient, but the efficiency is achieved at the cost of reduced accuracy, especially for circuits that contain many reconvergent fanouts, since circuit line signal probabilities are not independent but the measures assume signal independence [4, 73]. Since most large circuits have a large number of reconvergent fanouts, the simulation-based testability (*dynamic*) analysis gives more accurate estimates of the probability values and the testability measures.

Table 5.16 shows the CPU time for the test generation and compaction process. Columns 2, 3, and 4 report the ATPG times for SPARTAN with partial-scan, SPARTAN partial-scan with TPI1 and SPARTAN partial-scan with TPI2. The CPU times in columns 2, 3, and 4 include the test compaction time. The CPU times for the *mpscan* partial algorithm were not reported by the authors. The Sun Microsystems Sun Fire 880™ server with four 750 MHz UltraSPARC III™ processors each with 2 GB of RAM was used for the DFT insertion, ATPG, and ROR compaction simulations.

Table 5.15: *ISCAS*-89 benchmark CPU time results for SPARTAN (partial-scan and partial-scan with TPI) DFT insertion.

Ckt.	SPARTAN CPU Time (s)		
	PS	PS+TPI1	PS+TPI2
s298	23	189	2
s344	26	197	1
s349	27	201	2
s382	33	201	3
s386	57	209	3
s400	44	219	10
s444	55	268	10
s526	57	315	11
s641	13	1050	5
s713	17	1133	5
s820	1	762	63
s953	20	1251	10
s1423	84	5382	67
s1488	10	2835	58
s1494	40	2834	94
s5378	1607	20702	135
s9234	3005	19759	57
s13207	7190	61494	92
s15850	3600	46857	2784
s35932	1296	259189	28080
s38417	1920	91762	27758
s38584	128784	112067	21704
Avg.	6723	28585	3680

Table 5.16: *ISCAS*-89 benchmark ATPG CPU times for SPARTAN (partial-scan and partial-scan with TPI) vs. TRAN [14].

Ckt.	SPARTAN ATPG CPU Time (s)			TRAN [15]
	PS	PS+TPI1	PS+TPI2	
s298	25	26	37	0.2
s344	29	54	41	0.3
s349	29	44	32	0.3
s382	36	66	38	0.3
s386	34	2	1	0.8
s400	36	71	39	0.3
s444	32	62	38	0.4
s526	56	109	44	1.0
s641	116	190	103	2.1
s713	98	181	120	3.1
s820	156	52	48	2.5
s953	4	16	4	3.7
s1423	159	352	613	8.5
s1488	4	28	12	3.4
s1494	74	46	96	3.7
s5378	3484	3846	4335	73.0
s9234	10713	8742	7899	803.4
s13207	8357	10710	725	807.2
s15850	13249	16353	10789	1117.2
s35932	36985	25433	38381	1617.2
s38417	56514	67016	70379	5077.6
s38584	128881	151211	178365	2483.8
Avg.	11776	12937	14188	549

5.6 Summary

This chapter presented results for the SPARTAN partial-scan and the SPARTAN partial-scan with test point insertion algorithms. The results of all the SPARTAN algorithms were compared with *mpscan*, because *mpscan* is the partial-scan algorithm with the best fault coverage results, and with TRAN with full-scan because TRAN is a commercially used combinational ATPG.

All of the SPARTAN algorithm average fault coverages (97.44%, 97.60%, and 97.72%) were higher than *mpscan*'s average fault coverage (96.75%). TRAN had the highest average fault coverge (98.16%) and SPARTAN partial-scan with TPI2 has a slightly lower average fault coverage (97.72%). Fault coverage results of s349 (99.75%), s713 (95.14%), s1494 (100.00%), and s35932 (89.92%) with SPARTAN partial-scan with TPI1 are higher than the corresponding fault coverages of TRAN. Fault coverage results of s444 (98.05%), s1494 (99.42%), s9234 (94.81%), and s35932 (89.83%) with SPARTAN partial-scan with TPI2 are higher than the corresponding fault coverages of TRAN.

The average test lengths of the three SPARTAN algorithms were lower than that of *mpscan*, but were a little higher than TRAN's average test length. The average test length for SPARTAN with partial-scan is 749 vectors, for SPARTAN partial-scan with TPI1 it is 406 vectors, and for SPARTAN partial-scan with TPI2 it is 445 vectors. The average test length for *mpscan* is 1675 and that of TRAN is 321. The reason for a slightly higher average test length for the SPARTAN algorithms is the sequential nature of the benchmarks. Even though the circuits are scanned to improve circuit testability, certain states in the circuit still may be difficult to reach. Long sequences are required to drive the circuit into these difficult states, thus increasing the overall test length of the circuit.

The average test volumes and test application times for the SPARTAN algorithms are higher than TRAN and *mpscan* because of s38417. This is because s38417 has a large number of reconvergent fanouts. By manually examining the circuit structure we found a gate with twenty-six fanout branches. A different transform such as a wavelet transform could provide a more accurate analysis and may prove to be a better choice for capturing the circuit behavior. Large values for test volume and test application time for s38417 skew the average. Most of the individual circuits have lower test volumes for the SPARTAN partial-scan with TPI1 and the SPARTAN partial-scan with TPI2 than the test volume of TRAN and *mpscan* for the corresponding circuits (see Table 5.6).

Section 5.4 discussed the logic gate overhead and Section 5.5 presented the CPU times for partial-scanning and test point insertion. The average overhead for SPARTAN with partial-scan only (10.99%) is lower than for TRAN with full-scan (15.98%), but slightly higher than for *mpscan* (4.10%). SPARTAN partial-scan with TPI1 and SPARTAN partial-scan with TPI2 have higher overheads than other techniques because of the extra test point hardware added by the test point insertion algorithm.

The SPARTAN partial-scan with test point insertion algorithms perform better (higher fault coverage, lower test length, lower test volume, and lower test application time) than TRAN and *mpscan* on a variety of circuits. The average values for the test length, test volume, and test application time are skewed by s38417.

Chapter 6

Conclusions

In this chapter we epitomize SPARTAN, a collection of DFT algorithms for partial-scan and test point insertion. In Chapter 3, we proposed a spectral and entropy-based partial-scan algorithm to improve the circuit testability. In an effort to further improve circuit testability, reduce test length, test volume, and test application time, we proposed two test point insertion algorithms in Chapter 4. The test generation was performed using the GATEST ATPG [61]. In Chapter 5 the results of the SPARTAN partial-scan algorithms and test point insertion algorithms were compared to *mpscan* [78], the best partial-scan algorithm, and full-scan results reported using a commercial combinational ATPG, TRAN [14].

Full-scan is most likely the DFT technique of choice for most complex industrial circuits. This is because full-scan makes each flip-flop in the circuit completely controllable and observable from primary inputs and outputs, thus making the circuit combinational in nature and reducing the test generation problem into a combinational one. But, to convert each flip-flop in a circuit into a scan flip-flop, a multiplexer is required. Addition of hardware to the circuit translates into delay, thus resulting in a lower operational clock frequency of the circuit. The multiplexer of the scan flip-flop adds delay equivalent to two gate-delays in all the clocked paths. In addition, flip-flop outputs have one extra fanout, which increases the capacitive loading of the output signal of the flip-flop. This can reduce the clock speed by 5 to 10% [13]. Also, full-scan can cause yield reduction by detecting faults in the test mode that will be untestable in functional mode.

Partial-scan provides an alternate structured DFT technique to full-scan. Partial-scan, as the name implies, scans a subset of the flip-flops in the circuit to improve the circuit testability. Partial-scan helps eliminate the extra hardware required to convert all of the flip-flops into scan flip-flops. Numerous partial-scan publications have been put forth to date. The SPARTAN partial-scan algorithm, proposed in Chapter 3, uniquely combines spectral analysis [35] using DSP methods and entropy analysis using Shannon's entropy [22]. Chapter 4 introduced two

spectral and entropy-based test point insertion algorithms TPI1 and TPI2. Each of the test point insertion algorithms, TPI1 and TPI2, is combined with partial-scan to attain fault coverage and test length results that are comparable to full-scan results. The results show that capturing toggling activity with a DSP transform is a good observability measure and comparing spectra of a candidate line or a candidate scan flip-flop to a the spectra at a primary output in its fanout cone is a reliable observability measure. Moreover, increasing information flow (entropy) of a circuit increases the testability of the circuit. Partial-scan with test point insertion performs better than partial-scan only because testability problems are often related to the combinational part of the circuit, allowing the combination of partial-scan and test point insertion to be more effective than partial-scan only.

The testability measure analysis offered by the spectral and entropy combination performs better than the existing measures such as the cardinality of the *minimum-feedback vertex* set (MFVS) [15] and SCOAP [30] and algorithms such as *mpscan* [78], PASCANT [8], and BALLAST [33]. This is because the spectral analysis uses the toggling frequency as a controllability measure and correlation of the spectral coefficients of the flip-flop with primary outputs in the flip-flop's fanout cone as an observability measure. The other methods are static without frequency information. The spectral analysis also incorporates the condensed s-graph of the circuit for cycle breaking and only considers SCC's of the condensed s-graph whose size is greater than 1. This analysis helps identify noisy flip-flops with high toggling frequency, but with low correlation with primary inputs and primary outputs, that will be good choices for scanning. The entropy analysis identifies flip-flops that improve information flow (toggling activity) through the circuit and selects flip-flops that may or may not be embedded in large cycles. The flip-flops that do not belong to a large flip-flop cycle can have bad controllability and observability and can be a source of testability problems. Also, testability problems related to the combinational part of the circuit are not considered by earlier partial-scan algorithms. Test point insertion helps with the testability issues with the combinational parts of the circuit.

The existing techniques such as the finding the MFVS of the s-graph for scanning do not work as well because finding the MFVS is a NP-complete problem and the possible solutions obtained with heuristics are not exact. Also, the MFVS method only takes into account the high-level structural information of the flip-flops only using s-graphs to decide which flip-flop needs to be scanned, disregarding the combinational part of the circuit. Combinational parts of the circuit with reconvergent fanouts and large fanout-free regions have a very adverse effect

on the circuit testability. The *mpscan* partial-scan algorithm uses a combination of the number of conflicts during test generation and circuit state information obtained via logic simulation with the cycle size that contains the flip-flop to select a scan flip-flop. If the number of states of a cycle is low, *mpscan* first tries to break this cycle and then reverts to the conflict measure to select more scan flip-flops, thus placing more emphasis on breaking large cycles. The problem here is that some flip-flop cycles will exhibit high toggling activity, but the observabilities of the flip-flops in the cycle may be low. Scanning flip-flops to increase just the toggling activity of the unscanned flip-flops only increases their controllability, but may or may not increase the observability of the unscanned flip-flops. The authors of *mpscan* do not consider the observabilities of the unscanned flip-flop in their cycle-breaking analysis. For the SCOAP measure, inaccuracies creep into the testability measures because of reconvergent fanouts. PASCANT also uses only cycle breaking as a criterion for scan flip-flop selection. BALLAST tries to convert the circuit into a balanced structure by scanning flip-flops. It lacks cycle breaking and analysis of the combinational part of the circuit.

Chapter 5 presented the results of the SPARTAN partial-scan only, SPARTAN partial-scan and TPI1 and SPARTAN partial-scan and TPI2 algorithms. All of the SPARTAN algorithm fault coverages (97.44%, 97.60%, and 97.72%) were higher than *mpscan*'s average fault coverage (96.75%). TRAN has the highest fault coverge (98.16%) and SPARTAN partial-scan with TPI2 has a slightly lower fault coverage. Fault coverage results of s349 (99.75%), s713 (95.14%), s1494 (100.00%), and s35932 (89.92%) with SPARTAN partial-scan with TPI1 are higher than the corresponding fault coverages of TRAN. Fault coverage results of s444 (98.05%), s1494 (99.42%), s9234 (94.81%), and s35932 (89.93%) with SPARTAN partial-scan with TPI2 are higher than the corresponding fault coverages of TRAN. Refer to Table 5.4 for the fault coverage results for all of the DFT techniques. The goal persued by most of the previous DFT techniques was low hardware overhead. Instead, the main goals of the DFT techniques in this dissertation were high fault coverage and low test application time. High fault coverage is important because a low fault coverage will force the designer to manually add more DFT hardware to the circuit to boost the fault coverage. This extra iteration of DFT insertion in the testing phase will delay the completion of the project and may increase production time and time-to-market for the product. We use more hardware than the previous partial-scan methods, but less then full-scan does, which is an attractive tradeoff to get reduced circuit delay and lower test application time than full-scan.

Chapter 5 also presented the test length, test volume, and test application time results of the SPARTAN partial-scan only, SPARTAN partial-scan and TPI1, and SPARTAN partial-scan and TPI2 algorithms. Most of the individual circuits have lower test lengths, test volumes, and test application times for the SPARTAN partial-scan with TPI1 and SPARTAN partial-scan with TPI2 than that of TRAN and *mpscan* (see Tables, 5.5, 5.6, and 5.8). The SPARTAN partial-scan with test point insertion algorithms perform better (higher fault coverage, lower test length, lower test volume, and lower test application time) than TRAN and *mpscan* on a variety of circuits. The average values for the test length, test volume, and test application time are skewed by results for s38417. Lower test volume and test application time help reduce the total testing time required to test each circuit. Reducing the testing time will result in reduced test cost because a larger number of circuits can be tested in the same amount of time. The test length, test volume, and test application time can be further reduced by test point insertion techniques that help insert don't cares (X's) in the test vectors because don't cares improve the compaction. Also, the test length, test volume, and test application time for SPARTAN partial-scan can be reduced by applying compression techniques (code-based, linear-decompression-based, or broadcast-scan-based schemes) that are usually incorporated with full-scan [73].

In this dissertation, we presented a DFT insertion algorithm called SPARTAN, which is a unique combination of partial-scan and test point insertion. Even though the SPARTAN algorithm has illustrated the potential to replace full-scan completely, the ability of full-scan to shift out the complete state of the circuit for post-silicon debug can be very instrumental for DFT engineers. Since full-scan incurs increased delays on the clock path, the tradeoff can be using partial-scan for timing-critical parts of the circuit and full-scan in the remaining circuit. Partial-scan can be the DFT technique of choice for very high-end microprocessors used in speed-critical servers.

Chapter 7

Future Work

A greedy algorithm, SPARTAN, for partial-scan and test point insertion is proposed in this dissertation. It requires user-specified parameters to perform DFT hardware insertion. Even though greedy algorithms are efficient, they do not always converge to the optimal global solution. A list of projects for extending this work further are briefly discussed below.

A Fast Optimization Algorithm

To improve the SPARTAN algorithm further, there is a need for a fast optimization algorithm that can quickly converge to the most optimal global solution to determine the best values for the parameters. The new algorithm may be formulated as a linear program, a non-linear program, a genetic algorithm, or an artificial neural network. There are many more optimization techniques that may be used here. However, the optimization complexity must remain $O(n^2)$.

Extend Partial-Scan and Test Point Insertion to Other Fault Models

The SPARTAN algorithm can be used to discover the effects of partial-scan and test point insertion on different models, such as the delay fault model, current tests, *n*-detect tests, resistive bridging faults, and crosstalk faults, other than the traditional stuck-at fault model.

Use Different Transforms

The spectral analysis can be done using other transforms such as the wavelet transform.

Fanout-Free Region (FFR) Analysis

A fanout-free region in a circuit is a single-output subcircuit that has the following properties:

1. Its output is either a fanout stem or a primary output.

2. Its inputs are either fanout branches or primary inputs.

3. It does not include any fanout stems except possibly its output.

A circuit with very large FFRs will have high *test counts* (TCs) and will require longer test sets. TCs of a single line give a lower bound on the number of fault effects that have to pass through that signal line such that all of the faults in its fanin cone can be covered [42]. Large FFRs require large test sets to test because all faults in that FFR have to pass through the single output of FFR and many test patterns will be needed to make sure that all of the faults in the FFR become observable on the FFR output at least once. This can have a significant impact on the total size of the ATPG generated test set. The high TCs can be reduced by TPI, which splits large FFRs into two FFRs with fewer inputs. This is achieved by inserting a CTP on the line where the test point is to be inserted [31]. The splitting of a large FFR into two smaller FFRs by inserting a CTP on line C is shown in Figure 7.1 [31]. We can use this FFR analysis to locate and break large FFRs in the circuits by locating a line for TPI that is likely to divide the FFR into two smaller and possibly equal-sized FFRs. We will locate a line that has roughly half the number of PIs in its fanin cone as the number of PIs in the fanin cone of the FFR that needs to be divided into smaller FFRs.

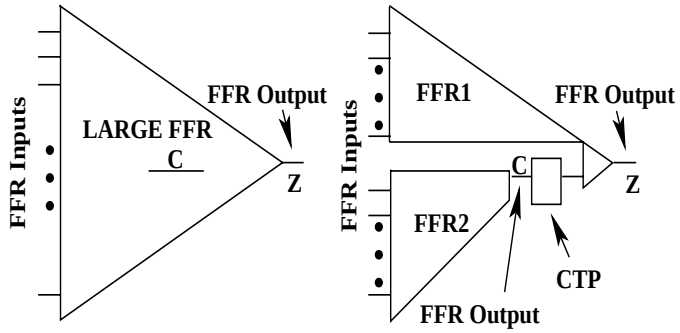

Figure 7.1: Splitting of an FFR.

Delay and *Critical Path Analysis* (CPA)

CPA and delay effects of adding DFT hardware in a circuit need to be addressed for the SPARTAN algorithm. Accurate delay models for each gate can be used to estimate the delays of

paths to locate the critical paths in the circuit. Once the critical paths are located, SPARTAN will not be allowed to scan flip-flops or insert test points on these paths. The critical path analysis algorithm can also be used to estimate the amount of delay incurred by the DFT hardware inserted.

Power Estimation

An estimate of the power savings achieved by the SPARTAN partial-scan and test point insertion algorithms is needed.

Power-Aware Partial-Scan and Test Point Insertion

The SPARTAN partial-scan and test point insertion algorithms can be made power-aware such that insertion of DFT hardware helps reduce power consumption during test mode.

Combined SFF Selection and Test Point Insertion

The partial-scan algorithm proposed in this dissertation makes a decision to scan a flip-flop based on the testability improvement of the circuit compared to other candidate scan flip-flops only. Then the test point insertion is performed and a test point is inserted based on the testability improvement of the circuit compared to other candidate test points only. A new algorithm is needed that weighs the testability improvement of scanning a flip-flop or adding a test point with other potential scan flip-flop candidates and test point candidates. This can help eliminate unnecessary flip-flop scanning because the algorithm will determine whether scanning a certain flip-flop is more beneficial than adding a test point within a certain distance (logic levels) of the flip-flop.

Appendix A

Users' Guide

The SPARTAN DFT tool set includes the partial-scan program (*SPARTAN_PS*), two test point insertion programs (*SPARTAN_TPI1* and *SPARTAN_TPI2*), and a shell script file as described below:

- *genSPARTANParams* – This shell script generates *SPARTAN_PS.params* for running the partial-scan program, *SPARTAN_TPI1.params* for running the TPI1 program, and *SPARTAN_TPI2.params* for running the TPI2 program. Please note that *SPARTAN_PS*, *SPARTAN_TPI1*, and *SPARTAN_TPI2* programs need their corresponding parameter files *SPARTAN_PS.params*, *SPARTAN_TPI1.params*, and *SPARTAN_TPI2.params* to run. They will not run without the parameter files so the parameter files have to be generated first.

- *SPARTAN_PS circuit_name* – Shell script file will perform partial-scan on *circuit_name.rutmod*. The output of the program will be the scanned netlists *circuit_nameS.rutmod* and *circuit_nameS.bench*.

- *SPARTAN_TPI1 circuit_name* – Shell script file will perform TPI using the TPI1 algorithm on *circuit_name.rutmod*. The output of the program will be the scanned netlists *circuit_nameT1.rutmod* and *circuit_nameT1.bench*.

- *SPARTAN_TPI2 circuit_name* – Shell script file will perform TPI using TPI2 on *circuit_name.rutmod*. The output of the program will be the scanned netlists *circuit_nameT2.rutmod* and *circuit_nameT2.bench*.

The *.bench* version of the netlist generated is needed to run the ATPG. To run the tools, place all the executables in a directory, and include the path name of the directory in your search path. Then create the parameter files *SPARTAN_PS.params*, *SPARTAN_TPI1.params*, and *SPARTAN_TPI2.params* using *genSPARTANParams*. For example, run the partial-scan program on

*s*27 as follows:

>> *genSPARTANParams*

>> *SPARTAN_PS* s27

Several parameters are available for the *SPARTAN_PS* program. All the parameters with default values will be specified in the *SPARTAN_PS.params* file. The user can modify the parameter values in this file. Default values will be used for parameters that remain unmodified by the user. The partial-scan parameters are:

- MATRIXSZ – Determines the order of the RWT. For example, if MATRIXSZ = 3, a 8×8 RWT matrix will be used for spectral analysis. The default value is set to 4.
- ST – ST is the scanning threshold. It determines the percentage of total flip-flops that will be allowed to scan by the program. The default value is set to 0.25.
- SPEC_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for spectral analysis. The default value is set to 512.
- ENT_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for entropy analysis. The default value is set to 512.

Several parameters are available for the *SPARTAN_TPI1* program. All the parameters with default values will be specified in the *SPARTAN_TPI1.params* file. The user can modify the parameter values in this file. Default values will be used for parameters that remain unmodified by the user. The TPI1 parameters are:

- levelrange – Determines the fraction of levels of the number of levels succeeding and preceding the middle of the circuit from which the initial test point candidate lines are to be selected. The default value is 0.99.
- numTPs – Decides the number of random test point candidates picked from the initial test point candidate list. The default value is 500.
- MATRIXSZ – Determines the order of the RWT. For example, if MATRIXSZ = 3, a 8×8 RWT matrix will be used for spectral analysis. The default value is set to 4.
- ST – ST is the scanning threshold. It determines the percentage of total flip-flops that will be allowed to scan by the program. The default value is set to 0.25.

- SPEC_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for spectral analysis. The default value is set to 512.
- ENT_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for entropy analysis. The default value is set to 512.

Several parameters are available for the *SPARTAN_TPI2* program. All the parameters with default values will be specified in the *SPARTAN_TPI2.params* file. The user can modify the parameter values in this file. Default values will be used for parameters that remain unmodified by the user. The TPI2 parameters are:

- levelrange – Determines the fraction of levels of the number of levels succeeding and preceding the middle of the circuit from which the initial test point candidate lines are to be selected. The default value is 0.99.
- MATRIXSZ – Determines the order of the RWT. For example, if MATRIXSZ = 3, a 8×8 RWT matrix will be used for spectral analysis. The default value is set to 4.
- ST – ST is the scanning threshold. It determines the percentage of total flip-flops that will be allowed to scan by the program. The default value is set to 0.25.
- SPEC_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for spectral analysis. The default value is set to 512.
- ENT_NUM_RAND_VEC – Determines the number of random vectors used for logic simulation for entropy analysis. The default value is set to 512.

References

[1] M. Abramovici, J. J. Kulikowski, and R. R. Roy. The Best Flip-Flops to Scan. In *Proc. of the Int'l. Test Conf.*, pages 166–173, 1991.

[2] V. D. Agrawal. An Information Theoretic Approach to Digital Fault Testing. *IEEE Transactions on Computers*, C-30(8):582–587, Aug. 1981.

[3] V. D. Agrawal, K-T Cheng, D. Johnson, and T. Lin. Designing Circuits with Partial Scan. *IEEE Design & Test of Computers*, 5(1):8–15, Mar. 1988.

[4] V. D. Agrawal and M. R. Mercer. Testability Measures: What Do They Tell Us? In *Proc. of the Int'l. Test Conf.*, pages 391–396, Nov. 1982.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley Publishing Co., Reading, MA, 1974.

[6] P. Ashar and S. Malik. Implicit Computation of Minimum-Cost Feedback Vertex Sets for Partial Scan and Other Applications. In *Proc. of the ACM/IEEE Design Automation Conf.*, pages 77–80, 1994.

[7] R. G. Bennetts, C. M. Maunder, and G. D. Robinson. CAMELOT: A Computer-Aided Measure for Logic Testability. *IEE Proc. E. Computers and Digital Techniques*, 128(5):177–189, Sept. 1981.

[8] S. Bhawmik, C. Cheng, K.-T. Cheng, and V. D. Agrawal. PASCANT: A Partial Scan and Test Generation System. In *Proc. of the IEEE Custom Integrated Circuits Conf.*, pages 17.3/1–17.3/4, May 1991.

[9] V. Boppana and W. K. Fuchs. Partial Scan Design Based on State Transition Modeling. In *Proc. of the Int'l. Test Conf.*, pages 538–547, 1996.

[10] S. Boubezari, E. Cerny, B. Kaminska, and B. Nadeau-Dostie. Testability Analysis and Test Point Insertion in RTL VHDL Specifications for Scan-Based BIST. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1327–1340, Sept. 1999.

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

[12] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagram. *ACM Computing Surveys*, 24(3):293–318, Mar. 1992.

[13] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Norwell, MA, 2000.

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

[15] S. Chakradhar, A. Balakrishnan, and V. D. Agrawal. An Exact Algorithm for Selecting Partial Scan Flip-Flops. In *Proc. of the* 31^{st} *ACM/IEEE Design Automation Conf.*, pages 81–86, June 1994.

[16] K-T. Cheng. Single-Clock Partial Scan. *IEEE Design and Test of Computers*, 12(2):24–31, Summer 1995.

[17] K-T Cheng and V. D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Trans. on Computers*, 39(4):544–548, April 1990.

[18] V. Chickermane and J. Patel. An Optimization Based Approach to the Partial Scan Problem. In *Proc. of the Int'l. Test Conf.*, pages 377–386, Oct. 1990.

[19] V. Chickermane and J. Patel. A Fault Oriented Partial Scan Design Approach. In *Proc. of the Int'l. Conf. on Computer-Aided Design*, pages 400–403, Nov. 1991.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill Book Co., New York, NY, 2002.

[21] F. Corno, P. Prinetto, M. Sonza Reorda, and M. Violante. Exploiting Symbolic Techniques for Partial Scan Flip-Flop Selection. In *Proc. of the IEEE Design Automation and Test in Europe (DATE) Conf.*, pages 670–677, 1998.

[22] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons Inc., Hoboken, New Jersey, 2006.

[23] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[24] S. K. Devanathan, O. I. Khan, and M. L. Bushnell. PROR Compaction Scheme for Larger Circuits and Long Vector Sequences. Unpublished, 2007.

[25] J. A. Dussault. A Testability Measure. In *Proc. of the IEEE 1978 Semiconductor Test Conf.*, pages 113–116, Oct. 1978.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

[27] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and H. Wunderlich. A Modified Clock Scheme for a Low Power BIST Test Pattern Generator. In *Proc. of the VLSI Test Symp.*, pages 306–311, April 2001.

[28] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. In *Proc. of the* 25^{th} *Int'l. Symp. on Fault-Tolerant Computing*, pages 337–343, June 1995.

[29] L. H. Goldstein. Controllability/Observability Analysis of Digital Circuits. *IEEE Trans. on Circuits and Systems*, 26(9):685–693, Sept. 1979.

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

[31] M. J. Guezebroek, J. Th. 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 Conference*, pages 138–147, Oct. 2002.

[32] R. Gupta, R. Gupta, and M. A. Breuer. BALLAST: A Methodology for Partial Scan. In *Proc. of the IEEE Int'l. Fault-Tolerant Computing Symp.*, pages 118–125, 1989.

[33] R. Gupta, R. Gupta, and M. A. Breuer. The BALLAST Methodology for Structured Partial Scan Design. *IEEE Trans. on Computers*, 39(4):538–544, April 1990.

[34] M. Hsiao, G. S. Saund, E. M. Rudnick, and J. H. Patel. Partial Scan Selection Based on Dynamic Reachability and Observability Information. In *Proc. of the Int'l. Conf. on VLSI Design*, pages 174–180, Jan. 1998.

[35] S. L. Hurst, D. M. Miller, and J. C. Muzio. *Spectral Techniques in Digital Logic*. Academic Press Inc., London, 1985.

[36] V. Iyengar and D. Brand. Synthesis of Pseudo-Random Pattern Testable Designs. In *Proc. of the Int'l. Test Conf.*, pages 601–508, Aug. 1989.

[37] N. Jiang, R. M. Chou, and K. K. Saluja. Synthesizing Finite State Machines for Minimum Length Synchronizing Sequences Using Partial Scan. In *Proc. of the IEEE Int'l. Fault-Tolerant Computing Symp.*, pages 41–49, 1995.

[38] P. Kalla and M. J. Ciesielski. A Comprehensive Approach to the Partial Scan Problem Using Implicit State Enumeration. In *Proc. of the Int'l. Test Conf.*, pages 651–657, 1998.

[39] P. Kalla and M. J. Ciesielski. A Comprehensive Approach to the Partial Scan Problem Using Implicit State Enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):810–826, July 2002.

[40] O. I. Khan, M. L. Bushnell, S. K. Devanathan, and V. D. Agrawal. SPARTAN: A Spectral and Information Theoretic Approach to Partial-Scan. In *Proc. of the Int'l. Test Conf.*, pages 21.1.1–21.1.10, Oct. 2007.

[41] K. Kim and C. Kime. Partial Scan by Use of Empirical Testability. In *Proc. of the IEEE Int'l. Conf. on Computer-Aided Design*, pages 314–317, 1990.

[42] B. Krishnamurthy. A Dynamic Programming Approach to the Test Point Insertion Problem. In *Proc. of the* 24^{th} *ACM/IEEE Design Automation Conf.*, pages 695–705, June 1987.

[43] A. Kunzmann and H. J. Wunderlich. An Analytical Approach to the Partial Scan Design Problem. *J. of Electronic Testing: Theory and Applications (JETTA)*, 1(2):163–174, May 1990.

[44] D. Lee and S. Reddy. On Determining Scan Flip-Flops in Partial Scan Designs. In *Proc. of the Int'l. Conf. on Computer-Aided Design*, pages 322–325, Nov. 1990.

[45] H. K. Lee and D. H. Ha. HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1048–1058, Sept. 1996.

[46] H. C. Liang and C. L. Lee. Effective Methodology for Mixed Scan and Reset Design Based on Test Generation and Structure of Sequential Circuits. In *Proc. of the* 8^{th} *IEEE Asian Test Symp.*, pages 173–178, 1999.

[47] X. Lin, I. Pomeranz, and S. M. Reddy. Full Scan Fault Coverage with Partial Scan. In *Proc. of the IEEE Design Automation and Test in Europe (DATE) Conf.*, pages 468–472, 1999.

[48] E. L. Lloyd, M. L. Soffa, and C. C. Wang. On Locating Minimum Feedback Vertex Sets. *J. of Computer and System Sciences*, 37(3):292–311, Dec. 1988.

[49] D. Marculescu, R. Marculescu, and M. Pedram. Information Theoretic Measures for Power Analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):599–610, June 1996.

[50] S. Narayanan, R. Gupta, and M. A. Breuer. Optimal Configuring of Multiple Scan Chains. *IEEE Trans. on Computers*, 42(9):1121–1131, Sept. 1991.

[51] T. Niermann and J. Patel. HITEC: A Test Generation Package for Sequential Circuits. In *Proc. of the European Conf. on Design Automation (EDAC)*, pages 214–218, Feb. 1991.

[52] T. M. Niermann, Wu-Tung Cheng, and J. H. Patel. PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(2):198–207, Feb. 1992.

[53] C. H. Papdimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[54] P. Parikh and M. Abramovici. Testability-Based Partial Scan Analysis. *J. of Electronic Testing: Theory and Applications (JETTA)*, 7(8):47–60, Aug. 1995.

[55] I. Park, D. S. Ha, and G. Sim. A New Method for Partial Scan Design Based on Propagation and Justification Requirements of Faults. In *Proc. of the Int'l. Test Conf.*, pages 413–422, Oct. 1995.

[56] S. Park. A Partial Scan Design Unifying Structural Analysis and Testabilities. *Int'l. J. on Electronics*, 88(12):1237–1245, Dec. 2001.

[57] K. Parker and E. J. McCluskey. Probabilistic Treatment of General Combinational Networks. *IEEE Trans. on Computers*, 24(1):668–670, Jun. 1975.

[58] I. M. Ratiu, A. Sangiovanni-Vincentelli, and D. O. Pederson. VICTOR: A Fast VLSI Testability Analysis Program. In *Proc. of the Int'l. Test Conf.*, pages 397–401, Nov. 1982.

[59] J. Rearick. The Case for Partial Scan. In *Proc. of the Int'l. Test Conf.*, pages 1032–1032, Oct. 1997.

[60] M. Riley, N. Chelstrom, M. Genden, and S. Sawamura. Debug of the CELL Processor: Moving the Lab into Silicon. In *Proc. of the Int'l. Test Conf.*, pages 1–9, Oct. 2006.

[61] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann. A Genetic Algorithm Framework for Test Generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(9):1034–1044, Sept. 1997.

[62] J. Saund, M. Hsiao, and J. Patel. Partial Scan Beyond Cycle Cutting. In *Proc. of the IEEE Int'l. Fault-Tolerant Computing Symp.*, pages 320–328, 1997.

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

[64] R. Sethuram, S. Wang, S. T. Chakradhar, and M. L. Bushnell. Zero Cost Test Point Insertion Technique to Reduce Test Set Size and Test Generation Time for Structured ASICs. In *Proc. of the* 15^{th} *IEEE Asian Test Symp.*, pages 339–348, Nov. 2006.

[65] S. Sharma and M. Hsiao. Combination of Structural and State Analysis for Partial Scan. In *Proc. of Int'l. Conf. on VLSI Design*, pages 134–139, Jan. 2001.

[66] J. E. Stephenson and J. Grason. A Testability Measure for Register Transfer Level Digital Circuits. In *Proc. of the IEEE Int'l. Fault-Tolerant Computing Symp.*, pages 101–107, June 1976.

[67] S. E. Tai and D. Bhattacharya. A Three Stage Partial Scan Design Method to Ease ATPG. *J. of Electronic Testing: Theory and Applications (JETTA)*, 7(11):95–104, Nov. 1995.

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

[69] K. Thearling and J. Abraham. An Easily Computed Functional Level Testability Measure. In *Proc. of the Int'l. Test Conf.*, pages 381–390, Oct. 1989.

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

[71] E. Trischler. Incomplete Scan Path with an Automatic Test Generation Methodology. In *Proc. of the Int'l. Test Conf.*, pages 153–162, Oct. 1980.

[72] H.-C. Tsai, K.-T. Cheng, C.-J. Lin, and S. Bhawmik. An Efficient Test Point Selection for Scan-Based BIST. *IEEE Trans. on VLSI Systems*, 6(4):667–676, Dec. 1998.

[73] L.-T. Wang, C.-W. Wu, and X. Wen. *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann, San Francisco, CA, 2006.

[74] M. J. Williams and J. B. Angell. Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic. *IEEE Trans. on Computers*, C-22(1):46–60, Jan. 1973.

[75] H.-J. Wunderlich and S. Hellebrand. The Pseudo-Exhaustive Test of Sequential Circuits. In *Proc. of the Int'l. Test Conf.*, pages 19–27, Oct. 1989.

[76] D. Xiang, M.-J. Chen, J.-G. Sun, and H. Fujiwara. Improving Test Effectiveness of Scan-Based BIST by Scan Chain Partitioning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):916–927, June 2005.

[77] D. Xiang and J. Patel. A Global Partial Scan Design Algorithm Using State Information. In *Proc. of the Int'l. Test Conf.*, pages 548–557, Oct. 1996.

[78] D. Xiang and J. Patel. Partial Scan Design Based on Circuit State Information and Functional Analysis. *IEEE Trans. on Computers*, 53(3):276–287, Mar. 2004.

[79] D. Xiang, S. Venkataraman, W. K. Fuchs, and J. H. Patel. Partial Scan Design Based on Circuit State Information. In *Proc. of the ACM/IEEE Design Automation Conf. (DAC)*, pages 807–812, June 1996.

[80] D. Xiang, Y. Xu, and H. Fujiwara. Non-Scan Design for Testability of Synchronous Sequential Circuits Based on Conflict Analysis. In *Proc. of the Int'l. Test Conf.*, pages 520–529, Oct. 2000.

[81] D. H. Younger. Minimum Feedback Arc Sets for a Directed Graph. *IEEE Trans. on Circuits and Systems*, CT-10(2):238–245, June 1963.

[82] Y. Zorian. Testing the Monster Chip. *IEEE Spectrum*, 36(7):54–60, July 1999.

Vita

Omar Ilyas Khan

Education

Ph.D. in Computer Engineering, Rutgers University, October 2007

M.S. in Computer Engineering, Rutgers University, October 2003

B. S. in Computer Engineering, Rutgers University, May 2001

Professional Experience

Feb. 2002 - June 2007
Rutgers University, Department of Electrical and Computer Engineering, Piscataway, NJ.
Position: Graduate/Research Assistant

June 2007 - Jul. 2007
Rutgers University, School of Engineering, Piscataway, NJ.
Position: Instructor

June 2006 - Jul. 2006
Rutgers University, School of Engineering, Piscataway, NJ.
Position: Teaching Assistant

July 2002 - Aug. 2002
Virage Logic Corporation, Hampton, NJ.
Position: Summer Intern

Publications

- Omar Khan, "*Statistical Response Compaction for Built-In Self-Test Systems Using Spectral Techniques.*" M.S. Thesis, ECE Department, Rutgers University, October 2003.
- O. Khan and M. L. Bushnell, "Spectral Analysis for Statistical Response Compaction during Built-In Self Test," in *Proc. of the Int'l. Test Conf.* (ITC), pp. 67-76, October 2004.
- O. Khan and M. L. Bushnell, "Aliasing Analysis of Spectral Statistical Response Compaction Techniques," in *Proc. of the* 19^{th} *Int'l. Conf. on VLSI Design*, pp. 801-806, Jan. 2006.

- R. Sethuram, O. Khan, H. Venkatanarayanan and M. L. Bushnell, “Power Reduction using Neural Net Branch Predictors,” in *Proc. of the* 20^{th} *Int’l. Conf. on VLSI Design*, pp. 679-684, Jan. 2007.
- O. Khan, M. L. Bushnell, S. Devanathan and V. D. Agrawal, “SPARTAN: A Spectral and Information Theoretic Approach to Partial Scan,” in *Proc. of the* 16^{th} *IEEE North Atlantic Test Workshop* (NATW), pp. 19-25, May 2007.
- O. Khan, M. L. Bushnell, S. Devanathan and V. D. Agrawal, “SPARTAN: A Spectral and Information Theoretic Approach to Partial Scan,” in *Proc. of the Int’l. Test Conf.* (ITC), pp. 21.1.1-21.1.10, Oct. 2007