Design for Testability Techniques to Optimize VLSI Test Cost

Swapneel B. Donglikar

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Dr. Michael S. Hsiao, Chair
Dr. A. Lynn Abbott
Dr. Chao Huang

June 26, 2009
Blacksburg, Virginia

Keywords: DFT, Illinois Scan, Random Access Scan, Test Data Volume

Design for Testability Techniques to Optimize VLSI Test Cost

Swapneel B. Donglikar

ABSTRACT

High test data volume and long test application time are two major concerns for testing scan based circuits. The Illinois Scan (ILS) architecture has been shown to be effective in addressing both these issues. The ILS achieves a high degree of test data compression thereby reducing both the test data volume and test application time. The degree of test data volume reduction depends on the fault coverage achievable in the broadcast mode. However, the fault coverage achieved in the broadcast mode of ILS architecture depends on the actual configuration of individual scan chains, i.e., the number of chains and the mapping of the individual flip-flops of the circuit to the respective scan chain positions. Current methods for constructing scan chains in ILS are either ad-hoc or use test pattern information from an a-priori automatic test pattern generation (ATPG) run. In this thesis, we present novel *low cost techniques* to construct ILS scan configuration for a given design. These techniques efficiently utilize the circuit topology information and try to optimize the flip-flop assignment to a scan chain location without much compromise in the fault coverage in the broadcast mode. Thus, they eliminate the need of an a-priori ATPG run or any test set information. In addition, we also propose a new scan architecture which combines the broadcast mode of ILS and Random Access Scan architecture to enable further test volume reduction on and above effectively configured conventional ILS architecture using

the aforementioned heuristics with reasonable area overhead. Experimental results on the ISCAS'89 benchmark circuits show that the proposed ILS configuration methods can achieve on an average *5%* more fault coverage in the broadcast mode and on average *15%* more test data volume and test application time reduction than existing methods. The proposed new architecture achieves, on an average, *9% and 33%* additional test data volume and test application time reduction respectively on top of our proposed ILS configuration heuristics.

to my family

Acknowledgments

I would like to dedicate this section to all the people who have been instrumental in the completion of not only this thesis but the graduate life as a whole. Nothing would have been possible without the support and encouragement from these people.

First of all, I would like to thank my advisor Dr. Michael S. Hsiao for his continued guidance and insightful suggestions throughout the duration of my research. I will always cherish my interactions with him for his methodical way of tackling the problems and ability to foresee the troublemakers.

I am also thankful to Dr. A. Lynn Abbott and Dr. Chao Huang for agreeing to serve on my committee.

I am deeply indebted to my colleagues Prashant Shrivastava, Suresh Kumar, Prasanna Kaliamoorthy and Sadiq Ahmed who introduced me to the challenging yet wonderful world of DFT during my stay at Open-Silicon India. I am also grateful to Dr. Karim Arabi, Mr. Craig Borden and Mr. Joseph Fang of Qualcomm Inc. for giving me an opportunity to intern at a world-class DFT team and to work on the Qualcomm's next generation DFT architecture.

Also my heartfelt thanks to friends in the PROACTIVE Lab for enriching my research experience - Maheshwar Chandrasekar, Mainak Banga, Shrirang Yardi, Anupam Shrivastava, Ankur Parikh, Karthik Channakeshava, Sandesh Prabhakar, Harini Jagadeesan, Neha Goel, Min Li, Nannan He, Xueqi Cheng by their constructive suggestions and timely help.

I am also grateful to my former and current apartment-mates Saurabh Kulkarni, Raghunandan Nagesh, Abhijit Pattewar, Rohit Rangnekar, Nikhil Rahagude and Percy Dadabhoy

for their help and making my stay at Virginia Tech a memorable one.

Last but not the least, I would like to thank my parents, sister and other family members and friends for their unconditional love, support and encouragement.

Finally, I am grateful to God for his countless blessings.

Swapneel B. Donglikar

June 26, 2009.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The famous Moore's Law [1] which predicted that the number of transistors on the integrated circuits will double every two years succinctly captures the tremendous progress the semiconductor industry has made in last few years. Today's Integrated Circuits (ICs) contain tens of millions of gates and can perform many complex tasks which a few years back can only be dreamed of. Electronic gadgets built upon these ICs have become an integral part of our lives and one just can't imagine a day without the gadgets like computers, cell phones, medical instruments and so on. To assure the correct functionality of these devices, a thorough testing of the underlying ICs for manufacturing defects is a must. As the ICs have packed more and more functionality, they have become more and more difficult to test. The naive way of testing of the ICs is by applying the functional patterns and verifying that we get the correct, expected outputs. Such a method suffers from numerous drawbacks - the functional patterns applied may not exercise all the parts of the logic and thus, fail to provide high degree of confidence about the absence of defects. Further, it is not always possible to visualize all possible operating conditions an ICs may be subjected to and thus, leading to hard-to-catch corner cases. Also a quality test - which can uncover large number of defects with minimal amount of testing - will require a deep understanding of the dataflow and control flow in the design which may be very complex. Since, most of the times the group which designs the logic and the group which develops the test cases are two different

entities, such a requirement is hard to meet. In addition, such basic testing has very limited diagnosability power and provides very little or no information about the possible location of the defect if a chip is found to be defective. All these factors have led to the wide-spread adoption of the *structural testing* to verify that the chip is defect free.

In *structural testing,* as the name suggests, we are concerned about verifying structural correctness of the design. In other words, we are interested in establishing that the actual logic gates, memories, flip-flops etc. are working as expected and do not have any defects associated with them. Towards this, we model the actual *physical defect* by some logical *fault* like line stuck-at some value, line late to transition etc. Then we consider all possible faults and generate *test patterns* (using software programs called as ATPG engine which stands for Automatic Test Pattern Generator engine) which can detect the presence of such faults. Furthermore, we assume that there is *only one* fault present in the circuit under test (CUT) at any given instance of time and generate a pattern for it. This is called a *single fault model* and it has been shown that this model can indeed model and detect most of the multiple defects that may occur. The most widely used fault model today is the *single stuck-at (SSA)* fault model which models a defect as some corresponding line in the circuit constantly stuck at either logic 1 or at logic 0.

Even with such a simplistic fault model, structural testing for the sequential circuits containing storage elements like flip-flops remains extremely challenging. It is due to the fact that in sequential circuits, the storage elements may need to be set to some specific values in order to detect the fault. But this requirement of setting the flip-flops to any arbitrary required value - also known as state justification - is very difficult to solve. To reach a particular state B from another state A may require many intermediate state transitions which may not be simple[1]. For example, it may be a case that a desired state cannot be reached as it is an *unreachable state* for the CUT; then in this case the faults which require this particular state in order to be detected become undetectable. This results in a lower *fault coverage* which is simply the percentage of the detected faults to the total modeled faults.

[1]In fact, the state justification itself is a vast area of research in the digital circuits verification domain.

Lower fault coverages directly translates into lower defect coverages and thus, test quality suffers. Another case may be that a fault may be detectable but it requires multiple states to be traversed and thus, needs greater test data volume and test application time. In general, most of the faults will fall under this case and thus, the required test data volume may be too large to fit on the Automated Testing Equipment (ATE) requiring expensive multiple ATE memory loads and/or the time required to apply these test patterns may be too high. Since ATEs are typically very expensive, together with high operating costs, the test cost may become too high for practical purposes if we have large test sets. Most of these issues can be easily sorted out if we can somehow abstract away the sequential nature of the CUT and this is the idea behind the *serial scan* design - the current de-facto *Design-for-Testability* (DFT) technique.

DFT techniques, in general, try to ease the VLSI testing process by adding some extra logic in the CUT exclusively for the test purposes. Serial Scan design adds a multiplexer in front of the flip-flops in the CUT so that, during the test mode, all of such flip-flops can be stitched together in one long shift-register. Thus, all of the flip-flops are now fully controllable and can be assigned to any desired state by shifting in the required values through the scan input(s). This eliminates the need for justification of states as would be needed in sequential ATPG. Also, the serial scan design has tolerable area and routing overhead. Hence, it has become a de-facto DFT solution.

Until recently, the serial scan design was sufficient to tackle the test costs issues regarding the test data volume and test application time as the circuit sizes were moderate and the single stuck-at fault model was able to model and capture most of the physical defects occurring on the ICs. But with the explosion in the circuit sizes and with multi-core chips, the number of flip-flops in the CUT has gone up significantly and also the number of patterns required to achieve the needed fault coverage. These two when combined lead to very large amount of test data volume and require longer test application time and thus, elevate the test cost. The situation gets even worse when we consider the deep-submicron effects - *SSA* alone cannot model all possible defects faithfully and we need to consider other advanced fault

models like transition delay faults, path delay faults, bridging faults etc. These additional fault models require additional test data volume and test application time.

The aforementioned test data volume and test application time explosion issues are due to the serial nature of the scan chain. Typically for a given fault, it is sufficient to specify only a few flip-flops for detection. If each of the faults is tested individually using a single scan chain, the test data volume would be prohibitively large. In addition, the test application time rises because the values have to be shifted in for all the flip-flops in the circuit irrespective of their role in the target fault detection. To address these drawbacks, scan chain partitioning is a viable solution. One effective architecture that has been popularized is the Illinois Scan (ILS) Architecture[3, 2], in which the partitioned chains can receive the values simultaneously via the broadcast mode. The effectiveness of the ILS architecture depends on the actual configuration of individual scan chains, i.e., the number of chains and the mapping of the individual flip-flops of the circuit to the respective scan chain positions. However, current methods for constructing scan chains in ILS architecture are either ad-hoc or require test pattern information from an a-priori ATPG run. An open question, thus, is whether it is possible to quickly configure the scan chains without an ATPG run before-hand, such that the fault coverage obtained in the broadcast mode would not be compromised?

The use of test patterns for the configuration of the scan chains ensures that *incompatible* flip-flops will not be grouped together to occupy the same position among different chains. By *incompatibility* we mean loading different values to the flip-flops in different scan chains simultaneously in the same scan shift cycle. In case of scan chain partitioning with an a-priori ATPG run, this partitioning depends on the values assigned to the flip-flops in the generated test set. A topology based method has no such constraints. Although a topology based approach may result in placing two flip-flops which have conflicting values in a given test set to the same position, the fault may still be detected. This is because in most cases, a fault is detectable by more than one vector and so a suitable pattern can be used instead.

Contributions of this thesis:

Our goal is to reduce the cost of scan chain partitioning such that the overall test data volume and test application time are reduced without compromising the fault coverage achievable in the broadcast mode. **The first contribution** of this thesis is three new low-cost heuristics to partition and arrange the flip-flops into chains so that high fault coverage in the broadcast mode can be achieved. Our heuristics efficiently utilize the circuit topology information and try to optimize the flip-flop assignment, thus, eliminating the need of an a-priori ATPG run. While we have demonstrated the usefulness of our approach in an ILS based setting, it is equally helpful to any on-chip compression technique. Our proposed approach has no additional hardware overhead apart from that in the normal ILS architecture i.e. the scan in and scan out pins, the multiplexers for selecting the scan inputs and an output response compactor. Experimental results show that the scan chain configurations obtained for ILS using our methods achieve high stuck-at fault coverage ($\geq 95\%$) for most of the big ISCAS'89 benchmarks [14] using the broadcast mode alone - on an average *5%* more than the existing methods and achieve on average *15%* more reduction in test data volume and test application time.

While the proposed heuristics do improve the broadcast mode fault coverage, there will still be some faults that cannot detected owing to the ATPG constraints associated with the broadcast mode. To cover these faults, in conventional ILS, another test mode with serial scan is used. The experimental results show that even though the number of such faults may be very small, the test data volume and test application time required to detect these can still be very high due to the need to shift through the entire chain. This prompted us to look out for an alternate way to detect these faults more efficiently. We propose the use of broadcast-enabled partial Random Access Scan, explained in Chapter 2.2.2, as a solution - this is our **second contribution**. We call the resulting scan architecture as the ILS-RAS Hybrid Architecture and describe its configuration, test generation scheme and test application scheme. The experimental results show that the proposed new architecture is a promising candidate that can help to further reduce the very high test data volume and

test application time associated with the current complex, multi-million gate designs with reasonable area overhead (refer to Chapter 4.1). The best case results for the bigger circuits indicate that, with just *1.69%* more area, we can achieve *27%* and *37%* additional reduction in the test data volume and test application time relative to the *effectively*[2] configured conventional ILS.

Organization of this thesis:

The rest of the thesis is organized as follows:

- Chapter 2: This chapter introduces various concepts in VLSI testing and describes various Design-for-Test (DFT) techniques used currently in the industry for quality and cost-effective test generation. It also surveys the various compression architectures proposed in the literature to reduce the test cost by reducing the test data volume and/or test application time. It describes in detail the Illinois Scan Architecture and Random Access Scan Architecture - these are the two scan architectures that we deal with in this thesis.

- Chapter 3: This chapter describes our proposed circuit topology based heuristics to configure the scan chains in the Illinois Scan Architecture to maximize the test cost reduction.

- Chapter 4: This chapter describes our proposed new ILS-RAS Hybrid Scan Architecture to further minimize the test data volume and test application time to reduce test cost.

- Chapter 5: This chapter concludes the thesis, describes limiting cases and outlines the future work that can enhance the current work.

[2]Here and in the sequel, the term *effective configuration of ILS* indicates the configuration that yields empirically best test data volume and test application time reduction.

Chapter 2

Background

This chapter introduces basic concepts in VLSI testing and various Design-for-Test (DFT) techniques used currently in the industry for quality and cost-effective test generation. It also gives a brief overview of the various compression architectures proposed in the literature to tackle test data volume and test application time issues. It describes in details the Illinois Scan Architecture (ILS) and Random Access Scan Architecture - two scan architectures that we deal with in this thesis with special emphasis on ILS. Specifically, it elaborates the Illinois Scan Architecture, discusses its underlying principles, elaborates its inherent limitations and leads to the need for the *effective* configuration. It also describes the current methods of configuring the scan chains in the broadcast mode.

2.1 Fundamental Concepts in VLSI Testing

Some of the fundamental concepts in the VLSI testing are defined below[17]:

- *Defect:* It is a flaw or physical imperfection that may cause a circuit to fail to perform in an expected manner. These defects occur due to imperfections in the materials and the manufacturing process.
- *Fault Model:* It is a logical representation of a defect like line permanently stuck-at some value, line late to transition etc. There are numerous fault models proposed in

the literature but unfortunately none of these can exactly model all possible defects that may occur. The most widely used traditional fault model is *stuck-at* fault model. For the deep-submicron devices, in addition to *stuck-at* model, more advanced fault models like *transition delay, path delay, small delay* etc. are used.

- *Fault Coverage:* It is a ratio of the number of detected faults to the total number of faults and is generally expressed in percentage. It quantifies the fault detection capability of a given test set for a given fault model. Fault coverage is associated with the yield and the detect level by the expression:

$$Defect\,level = 1 - yield^{(1-fault\,coverage)}$$

- *Test Vector:* An input pattern or a sequence of input patterns that can produce different output responses for the fault-free and faulty circuit.

- *Automatic Test Pattern Generation (ATPG):* It is the process of automatically generating the set of test patterns to detect a target fault for a given fault model using a computer program.

- *Single Fault Model:* A given circuit has n possible fault sites for a given choice of fault model. *Single fault model* assumes that there is only one fault present in the defective circuit. Obviously, normally it may not be the case and *multiple-fault model* is more accurate. But it has been shown that high fault coverage with single fault model results in high fault coverage in multiple-fault model too. Hence, the single fault model is typically used for test generation and evaluation purposes.

- *Equivalent Faults:* It is a set of faults which has identical faulty behavior for all input patterns and cannot be distinguished from each other.

- *Undetectable Fault:* It is a fault for which there exists no test to distinguish a fault-free circuit from a faulty circuit containing that fault. Such faults are also called as *redundant* or *untestable* faults in combinational circuits.

- *ATPG Undetectable Fault (AU Fault):* It is a fault which becomes *undetectable* due to the ATPG constraints in effect. In general, such constraints are due to the constraints that are needed to put the design into the test mode.

- *Test cube:* It is a deterministic test vector in which the bits that are not specified by the ATPG are left as *don't cares* (X).

- *Automatic Test Equipment (ATE):* It is a computer controlled equipment used for production testing of the ICs. It stores both - test patterns and their expected responses for a fault-free circuit.

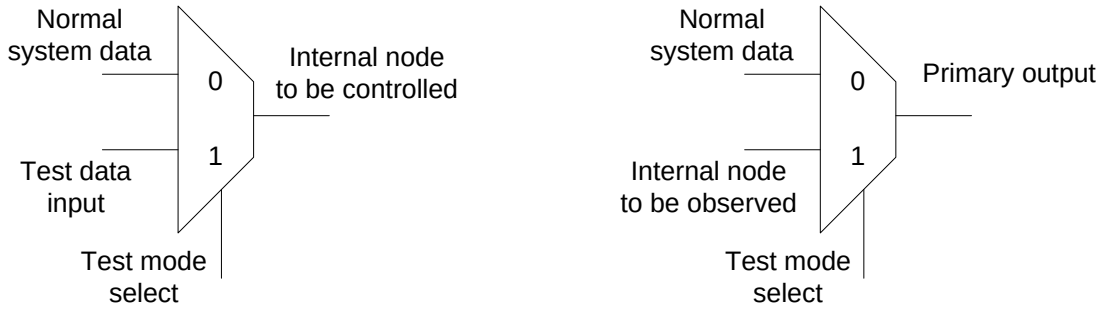
- *Test Data Volume (TDV):* It is the amount of data required in number of bits to store all the test patterns and their expected responses on the *ATE.*

- *Test Application Time (TAT):* It is the amount of time an *ATE* needs, expressed as number of test clock cycles, to apply all the test patterns.

- *ATE Test Cost:* It is the cost incurred to test a single IC on the *ATE.* It is determined by number of test patterns that need to be applied, test application time to apply each pattern, time required to reload the *ATE* memory if not all of the test data volume fits on *ATE* and the number of ICs that can be tested simultaneously. It also known as simply *test cost.*

2.2 Design for Testability (DFT)

As explained in the Introduction chapter, structural testing is essential for quality test patterns that can achieve high fault coverage with small test data volume and test application time. To minimize the amount of time and efforts required to derive high-quality structural testing patterns, augmenting the circuit during the design phase so that the design becomes more testable was proposed in 1970s. This approach adds some extra logic in the design exclusively for the testing purposes and is commonly known *Design for Testability.* The



(a) controllability test point (b) observability test point

Figure 2.1: Ad-Hoc DFT testpoints

main goal of the DFT logic is to greatly enhance the controllability and observability of the signals in the circuit. The DFT techniques can be broadly classified into three categories described next.

2.2.1 Ad-Hoc DFT

These were the initial DFT techniques proposed targeted only towards the *untestable* portion of the circuit. Here by the term *untestable* we mean the parts of the circuit which are very difficult to control or observe. These techniques add DFT logic, typically called as *testpoints,* to improve the controllability and observability of such hard-to-test regions. In general, *testpoints* enable access to the internal nodes directly from the circuit top-level. An example of a multiplexer based *testpoints* is shown in the Figure 2.1.

2.2.2 Scan Design

It is obvious that the *ad-hoc* methods described above are not scalable. As the circuit size grows, large numbers of *testpoints* and control signals become impractical. This led to the idea of using the storage elements (flip-flops) as the control and observe points by making them directly controllable and observable. This is called as *scan design.* Note that since the scan flip-flop value can be directly controlled and observed, we can consider the output of scan flip-flop as *pseudo primary input (PPI)* and the input as *pseudo primary output (PPO).*

Thus, for test generation purposes, the sequential nature of scan flip-flops is abstracted away and any logic which is fully controlled and observed by these scan flip-flops can be considered to be combinational logic.

Based on how many of flip-flops are made scanable, we can classify the scan design technique as:

- *Partial Scan:* In this, only some of the flip-flops are made scanable. So, the area and routing overhead can be reduced. But the presence of non-scan flip-flops will call for sequential ATPG which has higher computational complexity compared to the combinational ATPG. Also, typically, test data volume and test application requirements of sequential ATPG patterns is higher than combinational ATPG patterns.

- *Full Scan*: In this, all of the flip-flops are made scanable. Thus, the circuit becomes fully combinational from the ATPG perspective. This allows very high fault coverage. The combinational nature also facilitates ATPG using complex fault models. Thus, full-scan method enables quick, very high quality and cost effective testing with reasonable hardware overhead. This is the reason why full-scan DFT has become the de-facto standard in the contemporary IC design.

There are two major techniques to make the flip-flops scanable:

- *Serial Scan*: In this method, a multiplexer is added in front of the data input of the flip-flops with its select line controlled directly from the top-level. The select line is normally called as *scan enable (SE)* and the other input of the multiplexer is called as *scan input (SI)* . A shift register is built from the scan flip-flops by feeding the output a flip-flop to the *scan input* of the next flip-flop as shown in the Figure 2.2. The shift register so realized is popularly known as *scan chain.* A top-level pin feeds the first flip-flop and another top-level pin observes the output of the last flip-flop in the chain. During the test mode, the *scan enable* is asserted and the scan chain is used to shift in (also known as *scanning in*) a test vector to be applied to the combinational logic.

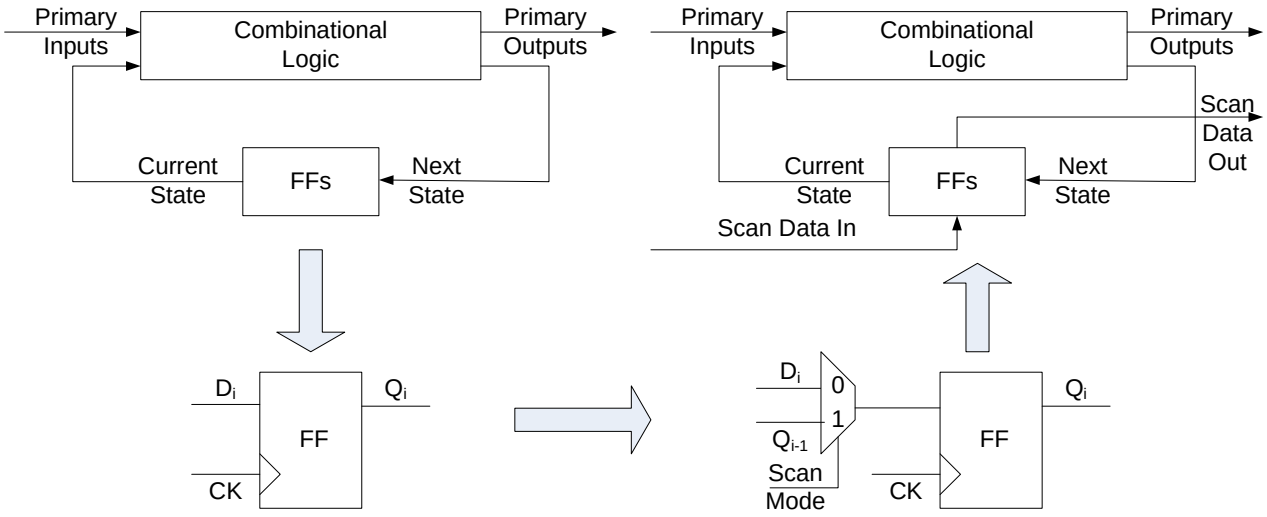


Figure 2.2: Abstract view of Serial Scan Architecture

Then, *scan enable* is deasserted so that the functional data has a path to the data-input of the scan flip-flops and the clock is pulsed once. This results in application of the test vector to the combinational logic and capturing the circuit responses into the flip-flops. Again, *scan enable* is asserted to shift out (*scan out)* these responses while simultaneously shifting in the next vector.

- *Random Access Scan*: In this method, each flip-flop is assigned an address so that it can be directly accessed from the top-level. This is similar to addressing a location in the memory space and hence the name (Figure 2.3). Using this address, we can set the flip-flop to the required value or observe its current value. Thus, in this method, we can directly set and observe only the required flip-flops for a given vector while in the serial scan we need to shift through entire scan chain. Hence, test data volume and test application time to apply a single vector for this method may be lower than the serial scan method if only a few flip-flops need to be set. But, this method incurs significant area overhead due to:

  – the big address decoder involved (requires lot of gates)

  – routing required to route the *scan input* to each individual flip-flop

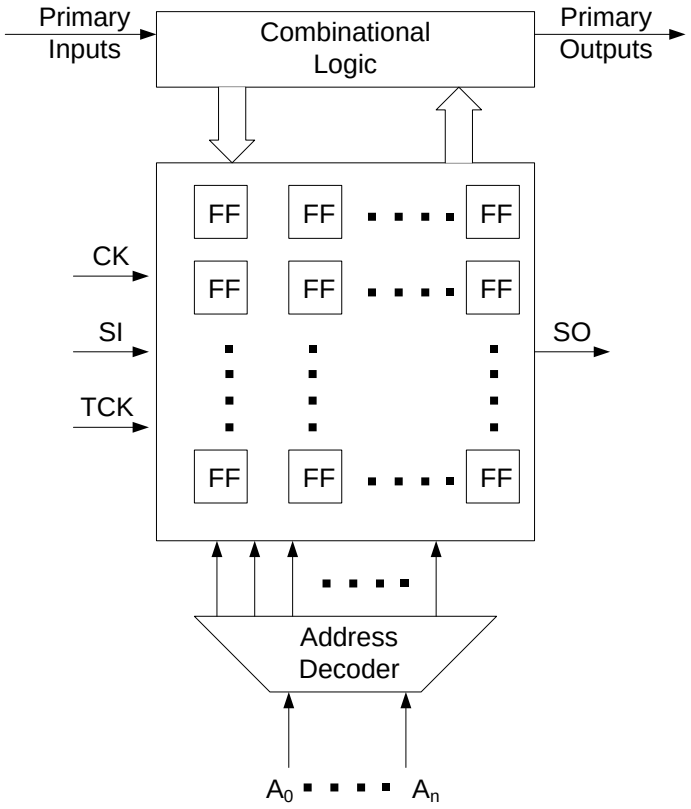


Figure 2.3: Abstract view of Random Access Architecture

– logic and routing required to observe output of each individual flip-flop

Before the era of nanometer feature size, silicon area cost per gate was quite significant and so, the additional gates required for the address decoder and flip-flop observation logic meant the overall RAS cost was too high compared to the serial scan. Hence, serial scan has been the preferred choice for implementing scan designs. But with current feature sizes of 45nm and below, the cost per gate has reduced considerably and this has led to the revived interest in the RAS.

2.2.3 Built-In Self-Test (BIST)

This DFT technique integrates a *test-pattern generator (TPG)* and an *output response analyzer (ORA)* on the IC itself to perform the testing internal to the IC. Figure 2.4 illustrates this concept. Since there is no need to apply test patterns and verify responses external to

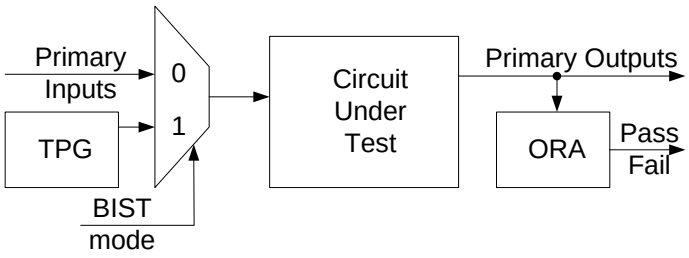


Figure 2.4: Basic BIST Architecture

the chip, BIST can be used at field also. But it requires adherence to a lot of strict design rules like blocking of unknown value propagation, isolation of analog and digital circuitry etc. In addition, BIST cannot efficiently test all faults, especially hard-to-detect faults due to the limited set of vectors the TPG can generate and hence, may result into lower fault coverage than the scan design. Also, it provides limited diagnosability compared to the scan design as we can only know whether all the test vectors have passed or failed and not which ones individually.

2.3 Illinois Scan Architecture (ILS)

Traditionally, in a full-scan circuit, a single scan chain consists of all the N flip-flops of the circuit (Figure 2.5(a)). To reduce the test application time, parallel scan chains can be used such that multiple chains can be loaded simultaneously (Figure 2.5 (b)). However, this is not a scalable solution as it does not reduce the test data volume as we need to store scan-load values for each of the scan-input pin and scan-unload values for each of the scan-output pin. Additionally, it requires more test pins for scan-inputs and scan-outputs. Hence, to reduce both test application time and test data volume simultaneously, various compression schemes have been proposed. These compression schemes add logic before scan chains to decompress the test stimulus coming from the ATE and after the scan chains to compress the response data going to the ATE [17, 26]. These compression schemes can be broadly classified as:

- *Code based schemes:* These schemes encode the *test cubes* using various data compres-

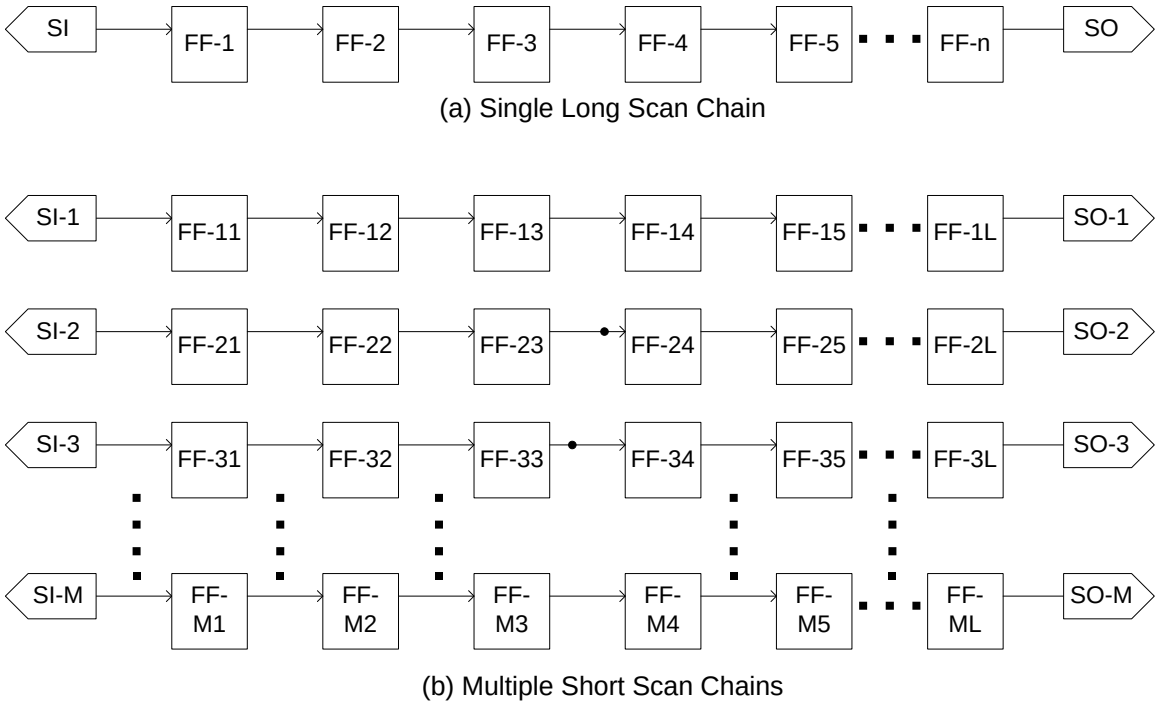


Figure 2.5: Conventional Serial Scan Architectures

sion codes. Examples are Dictionary code [45], Huffman code [46], Run-Length code [47], Golomb code [48].

- *Linear Decompression based schemes:* These schemes use linear operations performed by *linear feedback shift registers and XOR networks* to expand the data coming from the ATE to feed the scan chains. Examples are Combinational Linear Decompressors [49], Fixed-Length Sequential Linear Decompressors [50, 43], Variable Length Sequential Linear Decompressors [51] and Combined Linear and Nonlinear Decompressors [52].

- *Broadcast scan based schemes:* These schemes broadcast the same value to multiple scan chains. Examples are Broadcast scan [5, 4], Illinois Scan [3, 2], Multiple Input Broadcast Scan [11], Reconfigurable Broadcast Scan [53, 9]and Virtual Scan [29, 21].

Of these, ILS architecture, which was proposed in 1999 [3] at the University of Illinois at Urbana-Champaign (and hence, the name), is of particular interest because of its simplicity, very low area overhead, and potential for high compression ratios.

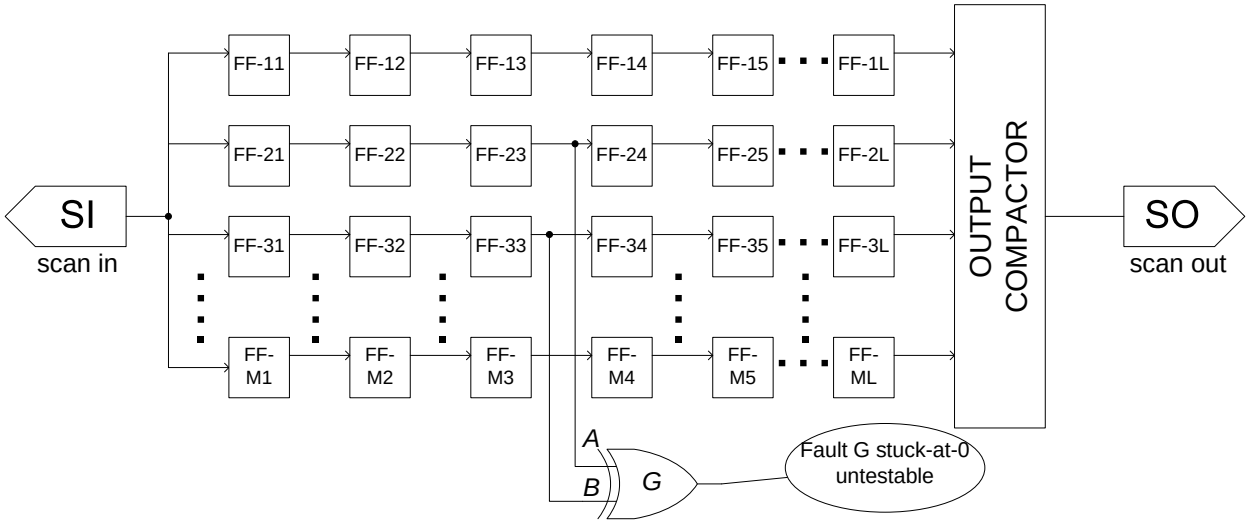


Figure 2.6: Illinois Scan Architecture

Let us consider a full-scan circuit with N flip-flops. In the ILS architecture, the N flip-flops are partitioned into M short chains - also called as *stumps* or *segments*[1]. These stumps can be connected to the common scan-in pin and can be loaded and unloaded simultaneously during the scan shift phase - known as the *Broadcast Mode*[2] (shown in Fig. 2.6) or can be concatenated together to form one single long scan chain - known as the *Serial Mode*. Toggling between the broadcast mode and serial mode is controlled by the multiplexers present at the head of the stumps (not shown in Fig 2.6). The select line of these multiplexers can be a top-level primary input or a dedicated control scan flip-flop. When in the broadcast mode, all the stumps are fed through the common scan-in pin causing the flip-flops at the same distance from the head of the *stumps* getting the same value. The tails of the *stumps* feed into an *Output Compactor* which compacts the responses from the individual *stumps* and produces a short *signature* which is then scanned out via the scan-out (SO) port. This output compactor can be sequential or combinational.

[1]In the sequel, in the context of broadcast mode, we will use the terms *chains*, *stumps* or *segments* interchangeably.

[2]In the sequel, we will use the term *configuration of ILS* to indicate *configuration of stumps in the broadcast mode of ILS.*

2.3.1 Underlying Principle and Inherent Limitation of ILS

The ILS architecture has its roots in the *Broadcast Scan* architecture [5, 4]. In the *Broadcast Scan* architecture, two or more *independent* circuits, each with its own single scan chain, are driven with the same shared scan-in input. This results in these circuits being tested simultaneously which translates to reduction in test data volume and test application time.

ILS extends this concept to a single circuit - it breaks the single scan chain into multiple stumps and drives all of these by the same shared scan-in. But since the logic blocks driven by these scan chains are not *independent*, not all of the circuit states would be achievable when using a shared scan-in pin. For instance, flip-flops at the same distance from the scan chain head in each of the chains are now constrained to receive the same value. This reduced set of achievable circuit states now causes some of the faults to be undetectable and leads to somewhat lesser fault coverage in the broadcast mode when compared to conventional single scan chain mode.

Fig. 2.6 exemplifies the problem of untestable fault in the broadcast mode due to reduced number of allowed circuit states. In the Figure, the *XOR* gate G in the design derives its inputs from the flip-flops *FF-23* and *FF-33*. To test the stuck-at-0 fault at the output of this gate G, we need either a 0-1 or a 1-0 combination at the input pins A and B of the *XOR* gate. But since A and B are connected to the flip-flops which are at the same distance from the head of their respective scan chains, it is impossible to achieve this combination in this configuration. So this fault which would otherwise be testable in the *serial mode*, now becomes undetectable in the broadcast mode. We refer to this unachievable combination as *unreachable state in the broadcast mode*. It follows that to minimize the degradation in fault coverage in the broadcast mode, there should be lesser number of such unreachable states in the broadcast mode.

In addition, in the *Broadcast Scan Architecture,* each circuit shares only the scan-in but has its own scan-out pin. In ILS, the multiple chains must share the scan-out pin(s) also. To achieve this, an *Output Compactor* is used. All the scan chains feed into this compactor which then performs compaction and scans out a short signature. In the compactor, it is

possible that the fault effect at one chain is masked by a controlling value or the fault effect at another chain. This is called as *aliasing* and may result in further drop in fault coverage in the broadcast mode. The number of *aliased* faults is dependent on the compactor circuit and can be reduced to be minimal by proper selection of the compactor circuit like Multiple Input Signature Register (MISR)[3], X-Compact [16] etc.

2.3.2 Factors Affecting Performance of ILS

While the ILS architecture does not dictate the number of flip-flops in each individual chain, in order to maximize the parallelism in the scan chains shifting process, it is desirable to have *balanced* scan chains (i.e., all *stumps* have nearly the same number of flip-flops). Then, in the ideal case, each *stump* will have

$$L = \lceil (N/M) \rceil \tag{2.1}$$

number of flip-flops. Intuitively, in the broadcast mode, since all the M *stumps* get loaded simultaneously, the test data volume and hence, test application time, should also reduce by a factor of M. But, in general, it is not the case. This is due to the aforementioned fact that the number of circuit states reachable in a given broadcast mode pattern is much smaller compared to a given serial scan pattern. Hence, for a given circuit, the number of broadcast mode patterns with ILS, in general, will always be higher than the number of single scan chain patterns without ILS.

The scan chains configuration parameters that affect the ILS performance are:

***Number of Chains*:** As the number of chains increases, parallelism increases and both the test data volume and the time to apply a single pattern reduces proportionally. However, at the same time, it leads to an increased number of untestable faults in the design because of the reduced number of circuit states that can be allowed in the ILS architecture, as discussed earlier.

***Positional Assignment of flip-flops in individual chains*:** It follows from the preceding discussion that the location of flops in the individual *stumps* determines the number of

ILS induced broadcast mode untestable faults. But this problem of architecting the *stumps* in ILS is NP-hard [3]. So, we need to rely on some heuristic to map the flip-flops to the scan chains and need to decide what should be the position of a given flip-flop in it, ensuring that the final scan chain configuration achieves a high fault coverage in the broadcast mode.

2.3.3 Existing Methods to Configure ILS

2.3.3.1 Ad-Hoc

This is the simplest and most prevalent way to configure the stumps in the broadcast mode of ILS. In this approach, first a single scan chain is built and then it is broken into the required number of chains. Generally, the single scan chain is built based on the alpha-numeric order of the flop instance name. This ensures that the flops in the same module are at the successive positions and helps reduce scan chain routing. Note that this approach does not explicitly try to minimize the *unreachable states* in the broadcast mode.

2.3.3.2 Compatibility Classes Based

This is the *effective* heuristic proposed by the inventors of the ILS in [3]. This heuristic uses a pre-computed test set information to find out the set of flops which when placed at the same column in ILS would lead to lesser number of broadcast mode undetectable faults.

In this heuristic, the authors first generate the partially specified ATPG vectors (i.e. *non-care bits* are left as don't care) for all the stuck-at faults in the design assuming a full-scan setup and without any pattern compaction. Then, with respect to this test set, they build the *compatibility classes* of the flip-flops with the maximum cardinality of each class equal to the number of chains desired (say M). Here, the *compatibility class* is defined as the group of flip-flops which are pairwise compatible for the specified test set. Two flip-flops are said to be *Compatible* in a given test set T if for each pattern in T, they have either same value or at-least one of them is unspecified (don't care), else they are declared to be *Incompatible*. It follows that the flip-flops in a compatible group do not introduce any broadcast mode

untestable faults in the circuit. Thereafter the scan chains are constructed by assembling the flip-flops from a *compatibility class* to the same position (i.e. column) in the chains. If the cardinality of a *compatibility class* A is smaller than M, then some flops from the other *compatibility class* B are put at the same position along with the original class A. The maximum number of such *incompatible* flops at any given position is limited to $\frac{M}{3}$ so as to control number of broadcast mode untestable faults induced.

For example, consider a circuit with 20 flip-flops. An ATPG run on the circuit yields two vectors

$$V_1 = 1X01\_XXXX\_XXXX\_XXXX\_XXXX$$

and

$$V_2 = 010X\_XXXX\_XXXX\_XXXX\_XXXX$$

where an X represents a don't care and the underscores have been added for better readability. The first bit represents the value on the first flip-flop, the second bit represents the value on the second flip-flop and so on. It is clear that the first and the third flip-flops have conflicting values in the vectors shown and so they cannot be located in the same *compatibility class*. On the other hand, the first and the fourth flip-flop, which do not have conflicting values can belong to the *same compatibility* class provided that they do not have conflicting values in the rest of the test set.

Since the test patterns generated by ATPG depend on the test generation algorithm, *compatibility classes* in the above approach need not be unique. To illustrate this point, consider the aforementioned vectors V_1 and V_2. In this case, the first and the third flip-flop could not be placed at the same location in different scan chains because their value conflicted in V_1. If the first vector can be replaced by another vector

$$V_1' = 10X1\_X0XX\_XXXX\_XXXX\_XXXX$$

which detects the same fault as V_1, the fault coverage will still remain the same but apart from that the first and the third flip-flops become *Compatible* and hence can be put in the

same *compatibility class*. In other words, *the compatibility classes are test-set dependent*.

2.3.3.3 Graph Coloring Based

This is another pre-computed test set based heuristic to configure the ILS. In [8], the authors model the problem of configuring the scan chains as a graph coloring problem - the number of colors corresponds to the number of *compatible classes*. The *compatible* flip-flops are defined in the same way as in Section 2.3.3.2 in terms of the test set derived from an a-priori ATPG run. An *incompatibility graph* with flip-flops as nodes and *Incompatibility* relation between two flip-flops denoted by an edge between them is constructed. A graph coloring algorithm is used to assign colors to the nodes of the graph such that none of the adjacent nodes are assigned the same color. In the paper, the authors use a greedy heuristic to achieve this. Note that this approach also suffers from the fact that it needs a pre-computed test set and the *compatibility classes* are test set dependent.

Chapter 3

Topology-based ILS Configuration

In this chapter, we propose 3 new low-cost heuristics to partition and arrange the flip-flops into chains so that high fault coverage in the broadcast mode can be achieved. Our heuristics efficiently utilize the circuit topology information and try to optimize the flip-flop assignment, thus, eliminating the need of an a-priori ATPG run. While we have demonstrated the usefulness of our approach in an ILS based setting, it is equally helpful to *any on-chip broadcast based compression technique*. Our proposed approach has no additional hardware overhead apart from that in the normal ILS architecture i.e. the scan in and scan out pins, the multiplexers for selecting the scan inputs and an output response compactor. Experimental results show that the scan chain configurations obtained for ILS using our methods achieve very high stuck-at fault coverage ($\geq 95\%$) for most of the big ISCAS'89 benchmarks using the broadcast mode alone. The broadcast mode fault coverage achieved is on an average 5% more than the current *effective* method. On top of this, on an average, we are able to achieve additional *15%* reduction in the total test data volume and test application time (combination of broadcast mode and serial mode patterns) with respect to the current test-set based *effective* method.

3.1 Motivation

From the discussion in the previous Chapter (2.3.1, 2.3.2), it follows that proper scan chain configuration plays an important role in deciding the effectiveness of ILS in achieving high compression ratios. The broadcast mode fault coverage of ILS suffers if a large number of incompatible flip-flops are assigned the same position in different *stumps*, resulting in an increased number of serial scan patterns. Although the aforementioned heuristics in Section 2.3.3 help to architect the scan chains in ILS with minimum incompatibility, they need a pre-computed test set, which may itself be a limiting factor. An open question, thus, is whether it is possible to quickly configure the scan chains without an ATPG run before-hand, such that the fault coverage obtained in the broadcast mode would not be compromised? So, our goal is to eliminate the dependency of the scan chain configuration on the test set and devise a fast, low cost heuristic that can deliver an *effective configuration*. Stated differently, we want to reduce the cost of scan chain partitioning such that the overall test data volume and test application time are reduced maximally without compromising the fault coverage achievable in the broadcast mode. In addition, such a heuristic should be scalable and easy to integrate into any existing scan insertion flow utilizing off-the-shelf commercial tools without incurring much run-time overhead. In this chapter, we propose circuit structure based heuristics which address these issues.

The use of test patterns for the configuration of the scan chains helps to ensure that there will be minimal number of *incompatible* flip-flops being grouped together to occupy the same position in different chains. By *incompatibility* we mean requirement of assigning different values to the flip-flops in order to detect the faults. In case of scan chain partitioning with an a-priori ATPG run, this partitioning depends on the values assigned to the flip-flops in the generated test set. A topology based method has no such constraints. Although a topology based approach may result in placing two flip-flops which have conflicting values in a given test set to the same position, yet the fault may still be detected. This is because in most cases, a fault is detectable by more than one vector and so a suitable vector can be used instead.

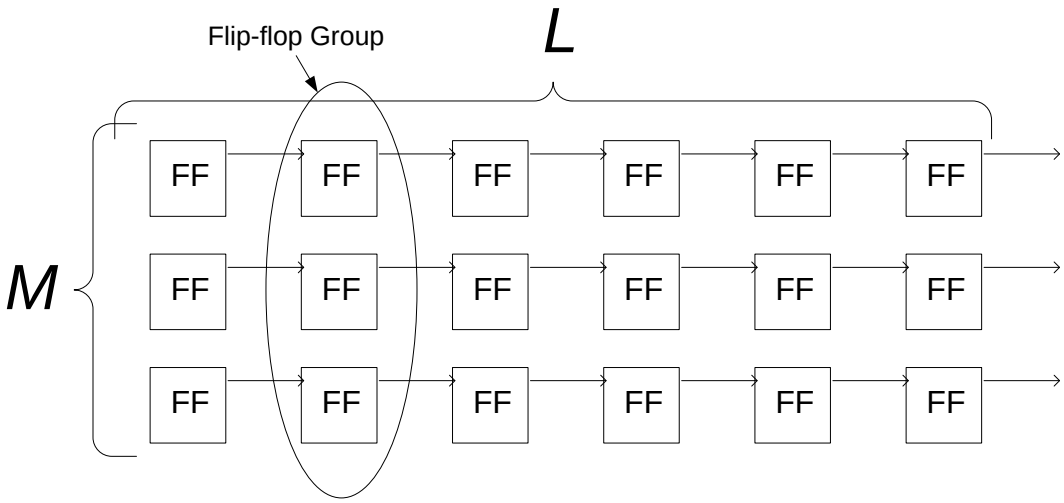


Figure 3.1: Relation of *Groups* and scan chains

3.2 Problem Formulation

In the following discussion we will use the term *Group* to denote the set of flip-flops assigned the same position in individual *stumps*. For example, in Fig. 3.1, the encircled flip-flops is a *Group*. The assignment of a particular value on a flip-flop determines whether a fault in its fanout cone can be detected. So if two flip-flops have disjoint sets of fanout cones, they can independently have the same value and may still be able to detect faults at their respective cones. Extending this concept to the *Groups* in the broadcast mode, if we have *Groups* with the minimum overlapping fanout cone between its member flip-flops, it will help to reduce the non-reachable states in the broadcast mode compared to the *serial mode*. This will lead to the higher broadcast mode fault coverage.

Referring to the Figure 3.2 (a), in this case there is a large overlap in the fanout cones of the flip-flops 2 and 6. This implies that there are lot of gates in the circuit which have both flip-flop 2 and flip-flop 6 in their fanin cone. This in turn means that many faults in the overlap region may depend on the values of flip-flops 2 and 6, and so we would prefer them to be independently controllable by placing them in different *Groups*. On the other hand, in Fig. 3.2 (b), flip-flops 3 and 7 have a very small overlap between them; few faults in the circuit would depend on both of them. So, assigning them to the same *Group* would not have

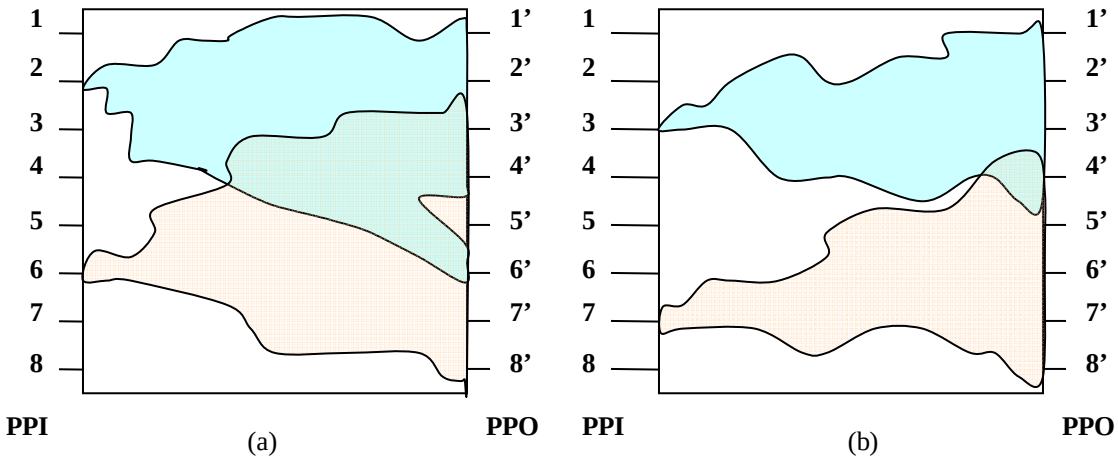


Figure 3.2: Fanout cone structure for a circuit
(a)Fanout cone overlap is more (b)Fanout cone overlap is less

a major impact on the fault coverage. In fact, it is possible that the faults in the common overlap region are independently detectable without any assignment on either flip-flop 3 or flip-flop 7. In such a case, these two flip-flops are entirely compatible.

We can indicate the fanout cone of a *Group* by keeping track of all unique gates in the fanout cone of each flip-flop in the *Group*. But this approach is too cumbersome to be scalable for bigger circuits. Instead, we use the set of unique *flip-flops* in the fanout cone as an indicator for it and call it as the *Dependency List* of the *Group*. The cardinality of the *Dependency List* is called as the *Dependency* of the associated *Group*. *Incompatibility* of one group with respect to another group is defined as the number of *unique* flip-flops in the intersection of their *Dependency Lists*. The *Incompatibility* value is used as a measure to quantify the overlap between two groups. If the *Incompatibility* between two groups is zero, then we call the flip-flops in the set of union of these two groups as *Compatible* flip-flops.

In general, it is difficult to find a large set of flip-flops that are compatible among themselves. So, the idea is to create the *Groups* for the broadcast mode in such a way so as to minimize the *Incompatibility* between the group member flip-flops. In our work, we assume that the number of *stumps* M is given, and we want to have stumps as *balanced* as possible. So, if the number of flip-flops in the design is N, then the desired length of each stump L is

given by Equation 2.1. Based upon these input constraints and the preceding discussion, we formulate the problem of configuring scan chains in ILS as generating L *Groups* with maximum group size of M with the minimum *Incompatibility* within members of each group. Note that here we want to minimize the *Incompatibility* only between the members of an individual *Group* and not across the members of different *Groups*. The *stumps* are then configured by first sorting the *Groups* in the descending order of their cardinality and then placing a member flip-flop from each group at the same relative position in the *stumps*.

3.3 Overview of Group Formation Procedure

In context of explaining our heuristics, we will use the term *Active Group List* for the list of *Groups* available for merging at any given iteration. Let N be the number flip-flops in the design. We start with an initial N *Active Groups* and reduce the number of *Active Groups* iteratively by merging an *Active Group A* with another *Active Group B* if the combined group size does not exceed M (i.e., the number of chains required). Here we call active group A as the *Host Group* and B as the *Guest Group*. After merging, the *Host Group A* is expanded to contain the members of *Guest Group*, and its *Dependency List* and *Dependency* are updated. The merged *Guest Group* is then marked as *non-active* and is not considered in any of the further merging iterations. Merging continues until there are no more *Groups* to be merged, i.e., when *Active Group List* becomes empty.

We start by constructing a $N \times N$ matrix *FCM* (Fanin/out Cone Matrix) in which both the row and column indices represent the corresponding flip-flop in the circuit. Any location $FCM[i][j]$ is set to 1 if flip-flop j is in the fan-in cone of flip-flop i, otherwise the value is set to 0. A sample FCM matrix for a 20-FF circuit is shown in the Figure 3.4. The fanout relationship for flip-flops in the circuit is shown in Fig. 3.3. In Fig. 3.3, the nodes 0, 1, 2, ... represent the source flip-flops (*PPIs*) and the primed nodes (0', 1', 2', ...) represent the destination flip-flops (*PPOs*). An arrow between a source flip-flop and a destination flip-flop indicates that the destination flip-flop is in the fanout cone of the source flip-flop. It then

follows that a *column* in the FCM corresponds to the fanout cone of the associated flip-flop.

Figure 3.5 shows the same circuit structure in the tabular form along with the *Dependency* of each source flip-flop.

We start by considering each column in the FCM as a *Group*. Thus, initially we have N *Groups*. We then iteratively reduce the number of *Groups* to L by merging two *Groups* in each iteration. The candidate *Groups* selected for merging are chosen based on the *Incompatibility* values between the *Groups* and the size of the resulting *Group*. Using the *FCM*, *Incompatibility* between the i^{th} *Group* and j^{th} *Group* can be defined as the dot-product of their respective columns. For example, the *Incompatibility* of *Group0* and *Group2* is

$$I_{0X2} = [01010000000000000000] \cdot [10100000000000000000] = 0$$

while that of *Group8* and *Group11* is

$$I_{8X11} = [00000000110100000000] \cdot [00000000010100000000] = 2$$

This means that while *Group0* and *Group2* are entirely compatible, *Group8* and *Group11* are not. This is because flip-flop 0 (corresponding to *Group0*) and flip-flop 2 (corresponding to *Group2*) have disjoint fanout cones while flip-flop 8 and flop-flop 11 do not.

The algorithm for the above methodology is shown in Algorithm 3.1 and the functions used in it are described in Table 3.1. The *inputs* to this algorithm are the full scan circuit, our *heuristics* and the number of scan chains M. The output is the flip-flop *Groups* with maximum cardinality M.

Note that Algorithm 3.1 is "*Evolutionary*" i.e. it first populates large number of small but fully *self-compatible groups* [1] and then later in the final stages, merges these small *Groups* to generate bigger *Groups* with required maximum cardinality. Hence, in a certain iteration, there may be a case that, for a given value of incompatibility, all choices of the *Guest Group* lead to resulting *Group* size exceeding maximum allowed size M. In such a case, we skip

[1] Here the term *self-compatible group* implies that all the members in the group are fully compatible

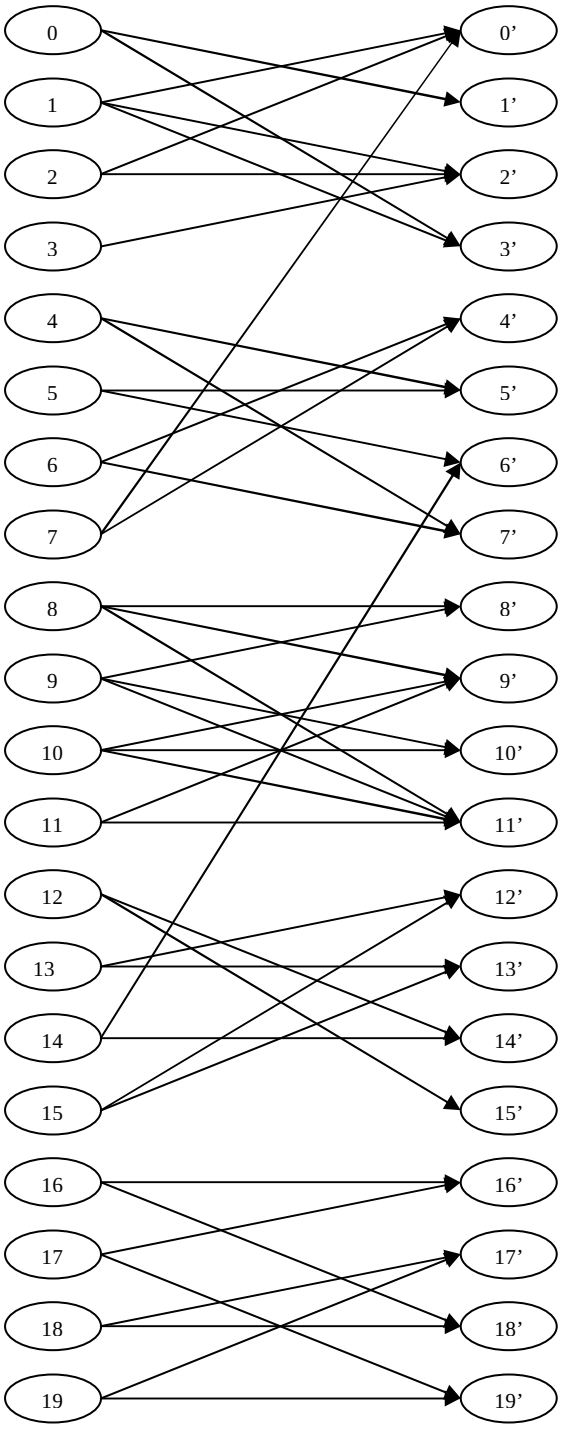


Figure 3.3: Graph representing fanout cone structure of a circuit

FF	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 3.4: Fanout cone matrix for the circuit in Figure 3.3

Flip-Flop ID	Fanout Cone	Dependency
0	1, 3	2
1	0, 2, 3	3
2	0, 2	2
3	2	1
4	5, 7	2
5	5, 6	2
6	4, 7	2
7	4, 0	2
8	8, 9, 11	3
9	8, 10, 11	3
10	9, 10, 11	3
11	9, 11	2
12	14, 15	2
13	12, 13	2
14	6, 14	2
15	12, 13	2
16	16, 18	2
17	16, 19	2
18	17, 18	2
19	17, 19	2

Figure 3.5: Fanin/Fanout Table with *Dependency* values for the circuit in Figure 3.3

Algorithm 3.1 Topology-based Partitioning(Circuit, N, M)

1: $GroupCount = N$
2: $ScanChainCount = M$
3: $FanoutConeList = DependencyList = \emptyset$
4: $IncompatibilityList = ActiveGroupList = \emptyset$
5: $GroupSize = 1$
6: **for** $i = 1 to N$ **do**
7: $Group[i] \Leftarrow \{i\}$
8: $FanoutConeList = FanoutConeList \bigcup FanoutCone(i)$
9: **end for**
10: $DependencyList \Leftarrow CreateDependencyList(FanoutConeList)$
11: $ActiveGroupList \Leftarrow GetActiveGroups(Groups)$
12: **while** $ActiveGroupList \neq \emptyset$ **do**
13: $HostGroup \Leftarrow SelectHostGroup(Heuristic)$
14: $ComputeIncompatibilityList(HostGroup)$
15: $GuestGroup \Leftarrow SelectGuestGroup(Heuristic, IncompatibilityList)$
16: **if** $Size(HostGroup) + Size(GuestGroup) < M$ **then**
17: $HostGroup \Leftarrow MergeGroups(HostGroup, GuestGroup)$
18: $UpdateActiveGroupList(GuestGroup)$
19: **else**
20: $MarkTerminalGroup(HostGroup)$
21: **end if**
22: **end while**

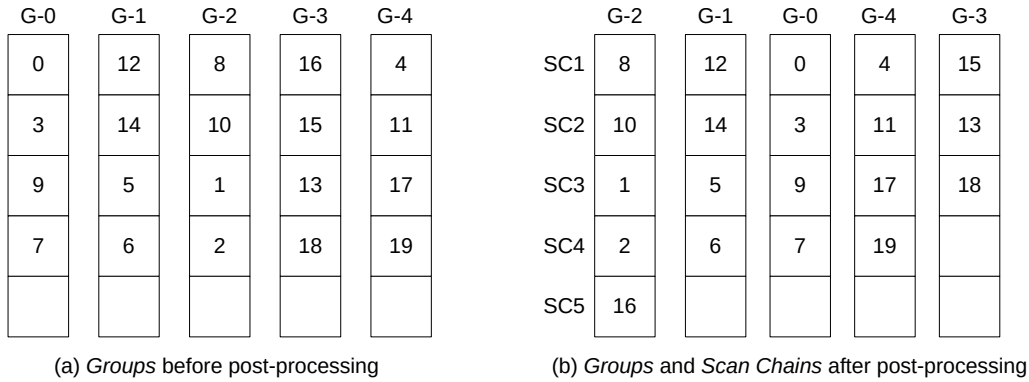


Figure 3.6: Terminal Groups

the currently selected *Guest Group* and try to find another *Guest Group*. This process is repeated until we find a valid *Guest Group* or we have exhausted all *Groups* in the current *Active Group List*. If we cannot find any valid *Guest Group*, then we declare the current *Host Group* as a *Terminal Group*; it is not expanded further. So, we may end up *overshooting* with more number of *Groups* than the target count of L. This will ensure that we never exceed the targeted maximum cardinality (i.e scan chain count) M; but this might lead to the case where the maximum *Group* size is less than M. So, in such a case, a final post-processing iteration is performed. In this iteration, the current biggest *Group* is expanded with the required number of members to boost its size to M. The algorithm for this post-processing step is given in Algorithm 3.2.

This case is illustrated in the Figure 3.6 for the circuit shown in Figure 3.3. Assume that we need 5 chains (i.e. $M = 5 \Rightarrow L = \frac{20}{5} = 4$) and the Figure 3.6 (a) indicates the groups at some stage in the *Groups* formation process. Then, it follows that no matter which *Group* is selected as the *Host Group,* none of the other groups can be merged into it. Hence, all of these *Groups* would be declared as *Terminal Groups* in the subsequent iterations. This results in the maximum *Group* cardinality of 4 and overshoots the target chain length by 1. So, in this case the post-processing described in Algorithm 3.2 is needed to find out the best *SubGroup* of cardinality 1 to boost the maximum *Group* cardinality to 5 and have 5 chains. The resulting *Groups* after the post-processing are shown in Figure 3.6 (b).

Table 3.1: Functions of Algorithms [3.1, 3.2]

Function	**Purpose**
FanoutCone(*i*)	Compute the fanout cone of flip-flop *i*
CreateDependencyList(*FanoutConeList*)	Create the dependency list from the *FanoutConeList*
GetActiveGroups(*Group, DependencyList*)	Get the list of *Groups* available for merging
ComputeIncompatibilityList(*Group*)	Populate the list of *incompatibility* values of active *Groups* wrt to *Group*
SelectHostGroup(*Heuristic*)	Select the *Host Group* based on the *Heuristic*
SelectGuestGroup(*Heuristic*)	Select the *Guest Group* based on the *Heuristic*
Size(*Group*)	Compute size of a *Group*
MergeGroups(*Group1, Group2*)	Merge *Group 2* into *Group 1*, update the size and *Dependency* of *Group 1*
UpdateActiveGroupList(*Group*)	Update *Active Group List*, mark *Group* as non-active
MarkTerminalGroup(*Group*)	Tag *Group* as *Terminal group*
Sort(*ActiveGroupsList)*	Sort the *ActiveGroupsList* wrt decreasing size and increasing *Dependency*
GetSubGroup(*Group, Count*)	Get first *Count* number of elements from the *Group*
ComputeSubGroupDependency(*SubGroup*)	Compute *Dependency* of *SubGroup*
ComputeSubGroupIncompatibility(*SubGroup*)	Compute *Incompatibility* of *SubGroup* wrt *Host Group*
SelectSubGroup(*SubGroups*)	Select the sub group with minimum *Incompatibility* and *Dependency*

Algorithm 3.2 Post-processing Generated Groups(Circuit, N, M)

1: $GroupCount = N$
2: $ScanChainCount = M$
3: $ActiveGroupList \Leftarrow GetActiveGroups(Group)$
4: $Sort(ActiveGroupsList)$
5: $HostGroup = ActiveGroupList[0]$
6: $Deficiency = M - Size(HostGroup)$
7: **for** $i = 1\,to\,N$ **do**
8: $SubGroup \Leftarrow GetSubGroup(ActiveGroupList[i],\, deficiency)$
9: $ComputeSubGroupDependency(SubGroup)$
10: $ComputeSubGroupIncompatibility(SubGroup)$
11: **end for**
12: $SubGroup \Leftarrow SelectSubGroup(SubGroupsList)$
13: $HostGroup \Leftarrow MergeGroups(HostGroup, SubGroup)$

3.4 Configuration Heuristics

In this Section we will look at 3 different topology-based heuristics to configure ILS. All of these heuristics share the same underlying group formation procedure described in Section 3.3; but differ in the *FCM* matrix and its usage.

3.4.1 Dependency-based Heuristic

This is the computationally least expensive configuration heuristic. In this heuristic, the *FCM* has only binary entries - a location $[i][j]$ is set to 1 if *PPO* i is in the fanout cone of *PPI* j else it is set to 0. Recall that the objective is to form the groups such that the fanout cones of the member flip-flops have least overlap, i.e., they should have minimum *Incompatibility* between them. To achieve this, we find and merge the *Groups* with as disjoint fanout cones as possible. At each iteration, we select the *Group* with lowest *Dependency* as the *Host Group* and the *Group* with minimum *Incompatibility* and lowest *Dependency* as the *Guest Group*. Selecting the *Host Group* in the proposed way will ensure that we always expand a *Group* which has highest potential to accommodate another *Group* without introducing any incompatibility between its members. Selecting the *Guest Group* in the proposed way will ensure that after merging *Guest Group* with the *Host Group*, the resulting *Host Group* will

have as small fanout cone as possible.

Thus, the sequence of steps in this heuristic is as follows: We start with the *Active Group List* sorted in ascending order of *Dependency* values. The *Group* with the lowest *Dependency* is selected as the *Host Group*. We calculate the *Incompatibility* of all other *Groups* in the *Active Group List* with respect to the *Host Group*. The first *Group* in the sorted *Active Group List* with minimum *Incompatibility* is selected as the *Guest Group*. We check if after the proposed merging, the *Host Group* size is still less than or equal to M. If yes, the *Guest Group* is merged into the *Host Group* and *Dependency List* and *Dependency* of the *Host Group* is updated. If the *Host Group* size reaches M, it is declared as a *Terminal Group* and is removed from the *Active Group List*. The *Guest Group* is tagged as merged and is not considered in any future merging iterations. This process is repeated until we have desired L number of *Groups*.

Iterative application of this algorithm for our example circuit (Figure 3.3) is shown in Figure 3.7 for the case of desired chains count = 5. Columns represent the *Sorted Active List* according to the increasing *Dependency* values and *I-k* stands for the k^{th} iteration. In first iteration we choose the *Group* with minimum *Dependency*, i.e., *group 3*, as the *Host Group*. We compute *Incompatibility* of all other groups in the *Active Group List* with respect to *group 3*. This is listed under the column *I-0*. From this list, we select *group 2* (with minimum *Incompatibility* value but maximum *Dependency*) as the *Guest Group*. Since the combined group size of *group 3* and *group 2* does not exceed the maximum *Group* size limit M, we merge *group 2* into *group 3* and remove *group 2* from the list of active groups. Then we update the *Dependency* values and continue this process of merging *Groups* until the *Active Group List* becomes empty.

3.4.2 Fanout Estimate Based Heuristic

In the previous *Dependency*-based heuristic, value of 1 in the *FCM* location $[i][j]$ just signifies that there is a path from PPI j to PPO i. It does not give any notion about the *actual* size of the fanout cone involved. There may be cases that for a chosen *Host Group,* there are multiple

I-0	**I-1**	**I-2**	**I-3**	**I-4**	**I-5**	**I-6**	**I-7**	**I-8**	**I-9**	**I-10**	**I-11**	**I-12**	**I-13**	**I-14**
3	**2**	**5**	**7**	**12**	**14**	**16**	**17**	**1**	**{3,0}**	**10**	**{2,4}**	**{7,11}**	**{16,19}**	**{17,18}**
2(1)	4(0)	6(0)	11(0)	13(0)	15(0)	17(1)	18(0)	{3,0}(2)	9(0)	{5,6}(0)	{7,11}(1)	{14,15}(0)	{17,18}(4)	{3,0,9}(0)
0(0)	5(0)	7(0)	12(0)	14(1)	16(0)	18(1)	8(0)	8(0)	10(0)	{7,11}(2)	{12,13}(0)	{16,19}(0)	{1,8}(0)	{10,5,6}(0)
4(0)	6(0)	11(0)	13(0)	15(0)	17(0)	19(0)	9(0)	9(0)	{7,11}(0)	{2,4}(0)	{14,15}(0)	{17,18}(0)	{3,0,9}(0)	{2,4,12,13}(0)
5(0)	7(1)	12(0)	14(0)	16(0)	18(0)	10(0)	10(0)	10(0)	{2,4}(1)	{12,13}(0)	{16,19}(0)	{1,8}(3)	{10,5,6}(0)	{7,11,14,15}(0)
6(0)	11(0)	13(0)	15(0)	17(0)	19(0)	1(0)	1(0)	{2,4}(2)	{5,6}(0)	{14,15}(0)	{17,18}(0)	{3,0,9}(1)	{7,11}(0)	{16,19,1,8}(4)
7(0)	12(0)	14(1)	16(0)	18(0)	{3,0}(0)	{3,0}(0)	{3,0}(0)	{5,6}(0)	{12,13}(0)	{16,19}(0)	{3,0,9}(1)	{10,5,6}(3)	{2,4}(0)	-
11(0)	13(0)	15(0)	17(0)	19(0)	8(0)	8(0)	{7,11}(0)	{7,11}(1)	{14,15}(0)	{17,18}(0)	{1,8}(2)	{2,4}(1)	-	-
12(0)	14(0)	16(0)	18(0)	9(0)	9(0)	9(0)	{12,13}(0)	{12,13}(0)	{16,19}(0)	{3,0,9}(2)	{10,5,6}(2)	-	-	-
13(0)	15(0)	17(0)	19(0)	10(0)	10(0)	{14,15}(0)	{14,15}(0)	{14,15}(0)	{17,18}(0)	{1,8}(2)	-	-	-	-
14(0)	16(0)	18(0)	10(0)	1(0)	1(0)	{2,4}(0)	{16,19}(2)	{16,19}(0)	{1,8}(2)	-	-	-	-	-
15(0)	17(0)	19(0)	{3,0}(0)	{3,0}(0)	{7,11}(0)	{5,6}(0)	{2,4}(0)	{17,18}(0)	-	-	-	-	-	-
16(0)	18(0)	10(0)	1(1)	8(0)	{12,13}(1)	{7,11}(0)	{5,6}(0)	-	-	-	-	-	-	-
17(0)	19(0)	{3,0}(0)	8(0)	{7,11}(0)	{2,4}(0)	{12,13}(0)	-	-	-	-	-	-	-	-
18(0)	10(0)	1(0)	9(0)	{2,4}(0)	{5,6}(1)	-	-	-	-	-	-	-	-	-
19(0)	1(2)	8(0)	{5,6}(1)	{5,6}(0)	-	-	-	-	-	-	-	-	-	-
8(0)	{3,0}(1)	9(0)	{2,4}(1)	-	-	-	-	-	-	-	-	-	-	-
9(0)	8(0)	{2,4}(1)	-	-	-	-	-	-	-	-	-	-	-	-
10(0)	9(0)	-	-	-	-	-	-	-	-	-	-	-	-	-
1(1)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
{3,0}	{2,4}	{5,6}	{7,11}	{12,13}	{14,15}	{16,19}	{17,18}	{1,8}	{3,0,9}	{10,5,6}	{2,4,12,13}	{7,11,14,15}	{16,19,1,8}	{17,18,3,0,9}

Figure 3.7: Scan Chain Formation
(values in {} denote the group and value in () denote the *Incompatibility*)

potential *Guest Groups* with the same minimum number of overlapping flip-flops in their fanout cone i.e. *minimum Incompatibility.* These groups may differ in the *actual* number of gates in the overlapping fanout cone region and ideally, we would like to choose the *Group* with the smallest *actual* overlapping fanout cone. But, unfortunately, the *Dependency*-based heuristic cannot accomplish this as it has no information about the *actual* fanout cone size involved. It simply picks the first *Group* with the minimum *Incompatibility.* Thus, it is intuitive that a heuristic which has information about the *actual* overlapping fanout cone involved may yield better results. This is the motivation behind the *Fanout Estimate based* heuristic.

Note that, as explained earlier, the exact calculation of the overlapping fanout cone region between a *Host Group* and all candidate *Guest Groups* is computationally very expensive as it will call for first finding the individual fanout cones of the *Host* and *Guest Groups* and then intersecting them. Hence, we need some heuristic measure which is easy and inexpensive to compute yet is reliable enough. Note that, we are interested in finding the relative overlapping fanout cone size between a *Host* and the candidate *Guests.* We don't really need an exact number of gates in the overlap region - it is sufficient if we can figure out the relative order of candidate *Groups* with respect to the overlap region size. In this heuristic, we propose and use the *cumulative level count* as a measure to come up with this relative order.

To compute the *cumulative level count,* we first *reverse levelize* the given circuit by assigning a *level* to each gate in the circuit. The *level* of the primary outputs and pseudo primary outputs is set to 1 and then the levels of all other gates are computed backwards towards the primary inputs, in the *breadth first manner*, using the following relation:

$$level(gate\,G) = \sum(G's\,all\,successor's\,level + 1) \tag{3.1}$$

Since we add up the *levels* of all of its successors to compute the *level* for a given gate, we call the *level* assigned to a gate as the *cumulative level-count.* It follows that, for a fanout free circuit, this *cumulative level count* for a given gate will correspond to the total number of

gates in it's fanout. Since the circuit under consideration is assumed to be a *full-scan* circuit, i.e., each flip-flop is replaced by a pair of pseudo primary input (PPI) and pseudo primary output (PPO), it is reasonable to assume that most of the circuit under consideration is fanout free and hence, the *cumulative level count* measure will give us reasonably accurate notion about the fanout cone of primary or pseudo primary inputs.

The overview of the steps in this heuristic is as follows: At each iteration, we select the *Group* with minimum *cumulative level count* as the *Host* Group and the *Group* with least fanout overlap with the *Host Group* as the *Guest Group.* To estimate the fanout cone overlap between the *Host Group* and a candidate *Guest Group,* the *cumulative level count* is recomputed for all the gates with the *level* of all primary outputs and pseudo primary outputs *in the fanout cone of the Host Group* set to 1. The *level* for rest other primary outputs and pseudo primary outputs is set to *infinity.* For a gate G, if the *level* for a successor is *infinity,* then it is not added during the *cumulative level count* computation. This ensures that only the gates, in particular the primary and pseudo primary inputs, in the fanin cone of the *Dependency List* of the *Host Group* have non-infinity *level* values. Then, the *Group* with the least *cumulative level count* is selected as the *Guest Group* to be merged. All other things like checks about the permissible *Group Size* and identification of *Terminal Groups* remain the same as in *Dependency*-based heuristic. It is expected that choosing the *Host* and *Guest Groups* in this way will lead to final *Groups* with minimum number of gates in the overlap region.

An instance of the *Guest Group* selection procedure described above is illustrated below for the smallest ISCAS 89 benchmark circuit $s27$. The Figure 3.8 is the full-scan version of it. Here the input gates $5, 6, 7$ are the pseudo primary inputs (PPIs) and the output gates $12, 17, 21$ are pseudo primary outputs. Suppose that currently there are 3 *Groups*, each with pseudo primary input $5, 6$ *and* 7 respectively. Let the *Group* with the PPI 5 is the *Host Group.* Then, the output gates in its fanout cone are $17, 20, 21$. Hence, for these output gates, *level* is initialized to 1 and for the remaining output gate 12, it is initialized to *infinity.* Then, the Table 3.2 shows the calculated *cumulative level count* values for all the

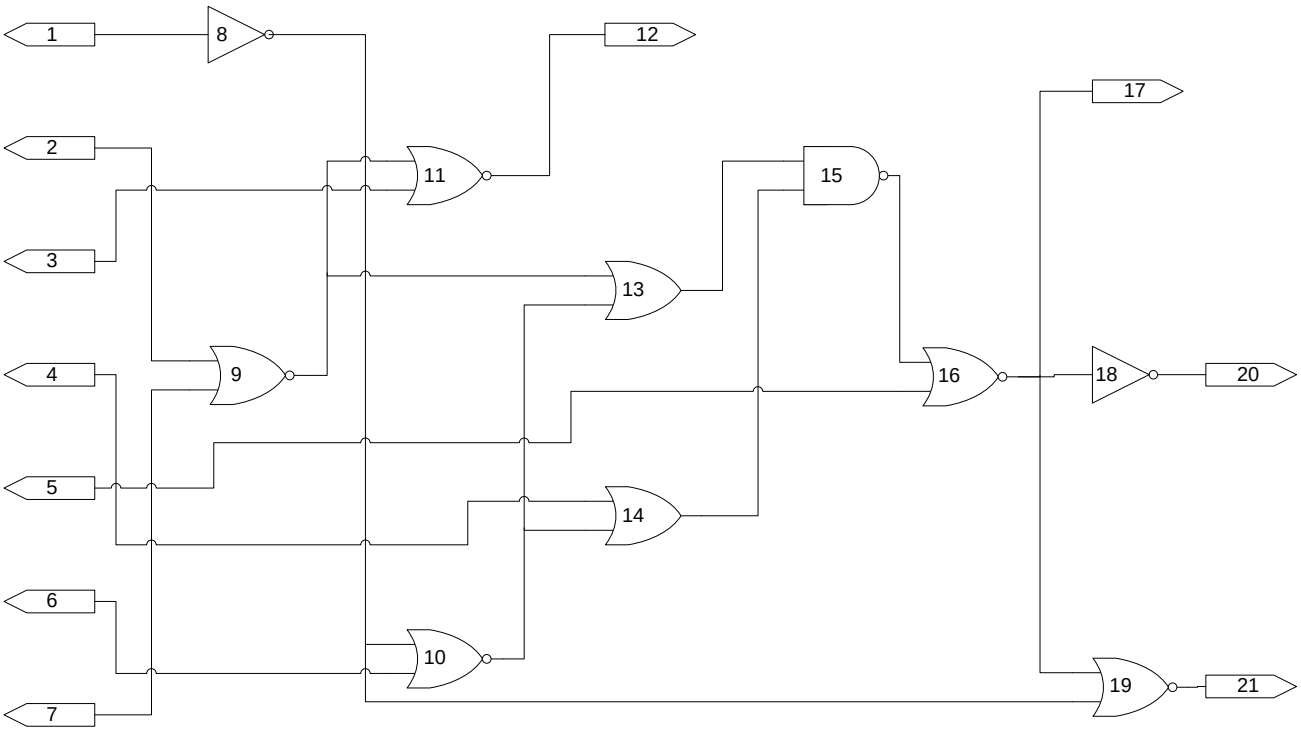


Figure 3.8: Example benchmark circuit *s27* (full-scan version)

gates. Since the *cumulative level count* for the PPI 7 is smaller than that for the PPI 6, we select the *Group* with the PPI 7 as the *Guest Group.*

3.4.3 SCOAP [30, 31] Based Heuristic

The previous fanout estimate based heuristic tries to minimize the absolute overlapping fanout cone between the *Host Group* and the *Guest Group*; but it does not really take into account how much testable the overlapping fanout really is. There may be a case that the overlapping fanout cone might be large but is very difficult to test in the broadcast mode owing to the dependencies between the scan flip-flops. We may be able to find the *Host* and *Guest Groups* with absolutely no overlapping fanout in the initial iterations of the *Groups* formation; but this will not be the case in the later iterations. Last few iterations are the *Groups* consolidation iterations in which two relatively big *Groups* will be merged together to form a one bigger *Group* and in general, these two *Groups* being merged will not be fully

Gate #	Fanout Estimate	Gate #	Fanout Estimate	Gate #	Fanout Estimate
1	27	8	26	15	9
2	15	9	14	16	8
3	3	10	22	17	1
4	11	11	2	18	2
5	9	12	1	19	2
6	23	13	10	20	1
7	15	14	10	21	1

Table 3.2: Fanout cone estimate for gates in the circuit in the Figure 3.8

compatible, i.e., they will have some overlapping fanout cone. Hence, in such scenarios it may be okay to merge two *Groups* with large overlapping fanout provided this overlapping region is *not* easily testable. This is the motivation behind this heuristic and we use a variation of the well known SCOAP testability measure [30, 31] to compute the testability of the overlapping region.

Combinational SCOAP measures define two *controllability* and one *observability* value for each line in the circuit. *CC0 / CC1* (combinational controllability to 0 /1) values correspond to the number of signals in the circuit that must be manipulated to control the line to 0 / 1 from the primary inputs. *CO* (combinational observability) value corresponds to the number of signals that must be manipulated to observe the current value of the line on the primary outputs. The controllability values are first computed from the primary inputs towards the primary outputs and then observability values are computed in the opposite direction. *CC0, CC1* and *CO* values range from 0 to ∞ and lower values mean the line is easy to control / observe. While the SCOAP values are easy to compute (O(n)), they suffer when a reconvergent fanout is encountered - SCOAP computations assume that all the lines in the circuit are uncorrelated and hence, end up with lower *optimistic* testability values. Figure 3.9 illustrates this problem for the simple case of a two input AND gate. In the Figure the tuple a/b/c above a line indicates the CC0, CC1 and CO values for the line. The controllability values for output Y and observability values for inputs A and B

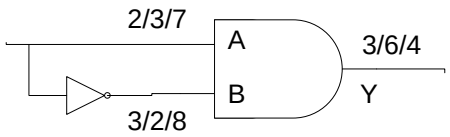


Figure 3.9: Misleading SCOAP measures for reconvergent fanout

are computed as follows:

$$\begin{aligned} CC0(Y) &= min(CC0(A), CC0(B)) + 1 \\ CC1(Y) &= \sum(CC1(A), CC1(B)) + 1 \\ CO(A) &= CO(Y) + CC1(B) + 1 \\ CO(B) &= CO(Y) + CC1(A) + 1 \end{aligned} \tag{3.2}$$

In our case, this problem gets even more aggravated as now the pseudo primary inputs which belong to the same *Group* are correlated and SCOAP computations can't account for this. So a simple workaround for this problem is to consider the SCOAP values in the opposite sense - note that when the SCOAP computations declare a signal to be difficult to control / observe, that is indeed the case! Hence, in this heuristic, we use the combinational SCOAP measures for the full-scan circuit to estimate how difficult it is to observe a value of a given PPI at the given PPO.

Thus, now the *FCM* matrix is augmented to contain the above described information - a location $[i][j]$ in the matrix now indicates how difficult it is to observe the PPI j at the PPO i. To obtain this information, we first compute the SCOAP controllability values for all signals in the regular way. Then, we calculate the *relative observability* of all PPIs with respect to *a single PPO* by setting the observability value for the PPO in the consideration to 0 and for rest other PPOs and POs to some fixed high value. We normalize the calculated relative observability values for all PPIs with respect to the smallest value on the scale of 1 to 10. This normalization maps the varying range of observability values to the more uniform range. We then invert these normalized values so that the smaller value means more difficult to observe. The location $[i][j]$ is then set to the corresponding inverted normalized value. If

a PPI cannot be observed at a given PPO (i.e. there is no path from a PPI to the PPO), the corresponding location is set to 0. This procedure is repeated for all the PPOs.

In this heuristic, we select the *Host Group* and *Guest Group* in the same way as that in the *Dependency-based* heuristic i.e. the *Host Group* is the *Group* with the least *Dependency* and the *Guest Group* as the *Group* with the least *Incompatibility* (i.e. based on the smallest dot-product of the associated columns). If there are multiple flop-flops in the *Group,* while calculating the dot-product of the columns, the non-zero entries in a row are replaced by the cumulative inverted relative observability values for flops in the *Group* for the PPO associated with that row. Other *Group* formation steps also remain the same.

3.5 Experimental Setup

In our experiments, we use a commercial ATPG engine to generate the ATPG patterns and verify the effectiveness of the partitioning across multiple heuristics using the ISCAS 89 [14] benchmark circuits[2]. All the experiments were carried out on a 3GHz Linux machine with Intel Dual core processor and 8GB RAM. We will compare the results for the following 5 ILS configuration heuristics:

1. Random (H0): flip-flops are assigned randomly to the chains
2. Test set based from UIUC (H1): refer to Section 2.3.3.2
3. Dependency based (H2): refer to Section 3.4.1
4. Fanout Estimate based (H3): refer to Section 3.4.2
5. SCOAP based (H4)[3]: refer to Section 3.4.3

All of the above heuristics were implemented in C++. In all of our experiments, we do not implement the output compactor (similar to [2, 28]) as the presence or absence of output compactor doesn't affect any of the heuristics considered in an unfair manner.

[2]We consider only the circuits with at least 50 flip-flops
[3]In Figures 3.10 to 3.21 this heuristic is represented as H4_staticCO

For ATPG purposes, initially, we had chosen to model the ILS architecture by adding suitable constraints to the commercial ATPG tool. Suppose we want to simulate the ILS configuration achieved using our *Dependency-based* heuristic. Then, we simply add the ATPG constraints stating that flip-flops in the same group (say 0, 3, 2, 5, 6) should be loaded with the same scan-shift value at the end of scan shift procedure. While this method is easy to implement and allowed us to explore a large number of ILS configurations quickly, it places severe constraints on the underlying commercial ATPG tool used. Most of the combinational ATPG systems, including the one we used in our experiments, are not designed to work under such constraints. This resulted in the performance of the ATPG suffering badly. It is not able to generate compact test set thereby resulting in a very high number of broadcast mode patterns. For example, for the circuit s38584, the ATPG time using the standalone single scan chain (without ILS) to achieve 95.96% fault coverage is 8.39 seconds and number of patterns generated is 174. For the case of broadcast mode coverage for configuration with 6 chains, the ATPG time is 163.07 seconds to reach the fault coverage of 95.54% but with 3910 patterns! Such a very high number of broadcast mode patterns nullifies the potential test data volume reduction. It follows that this result can be improved significantly if we input a Verilog netlist with actual ILS implementation and let the tool do unconstrained ATPG. Similar observation is reported in [2], which also used a commercial ATPG to model the ILS. Hence, this lead us to our second setup described next.

We use a two step procedure for the ATPG based heuristic comparison. We write out two Verilog files for a given circuit - one for the broadcast mode with the required number of scan chains configured based on the *Groups* formed and another for the *serial mode* with a single scan chain containing all the flip-flops in the design. Then, we first perform ATPG with all the faults added for the broadcast mode Verilog and dump out the list of undetected faults. Then we perform ATPG with the *serial mode* Verilog but target only those faults which are undetected in the broadcast mode. For each of the ATPG runs, we set the backtrace limit to 1000, employ random-filling for the unspecified bits in the patterns generated to maximize the fault coverage and perform *reverse fault simulation with fault dropping* (i.e.

fault simulate the patterns in the reverse order of their generation and drop the faults which are detected) to remove any *redundant* patterns. A pattern is declared as redundant if it does not detect any faults in such reverse simulation.

In our experiments, for each circuit Ckt we considered six different ILS configurations - $C1$, $C2$,...,$C6$. These configurations were obtained, respectively, by setting the number of scan chains to the first six multiples of k. $k = 6$ if number of flip-flops in $Ckt < 200$; otherwise $k = 16$. The value of k for the given number of flip-flops indicates the optimal number of chains found empirically [3]. Hence, the configuration $C1$ can be considered as the *best case* configuration and all other configurations serve as *stress test* to verify the applicability of our proposed methods under the increasing level of constraints.

In all of our test data volume (TDV) and test application time (TAT) calculations, we calculate test data volume and test application time for serial patterns using the Equations 3.3, 3.4

$$Serial\ Mode\ TDV\ (bits) = (SPC) \times (2 \times FF + PI + PO) \tag{3.3}$$

$$Serial\ Mode\ TAT\ (cycles) = FF + (SPC) \times (FF + 1) \tag{3.4}$$

where SPC indicates the *Number of Serial Mode Patterns*, FF refers to the number of flip-flops, PI refers to the number of primary inputs and PO refers to the number of primary outputs. In test data volume calculations, the number of flops is multiplied by 2 as we need to store both the scan load and scan capture value on the ATE. Similarly, test data volume and test application time for the broadcast mode patterns are calculated using the Equations 3.5, 3.6

$$Broadcast\ Mode\ TDV\ (bits) = (BPC) \times (LSCL + PI + PO) \tag{3.5}$$

$$Broadcast\ Mode\ TAT\ (cycles) = LSCL + (BPC) \times (LSCL + 1) \tag{3.6}$$

where *BPC* indicates the *Number of Broadcast Mode Patterns* and *LSCL* indicates the *Longest Scan Chain Length*. Here we do not multiply the number of flip-flops in the longest scan chain by 2 because in this mode we just need to store the scan load value for each flip-flop. Assuming that an M stage MISR present at the output and the signature is sampled for comparison only after all the broadcast mode patterns have been applied, we can estimate the MISR test data volume as simply M bits and MISR test application time as simply M cycles. In general, *MISR test data volume/test application time* of M bits/cycles is negligible when compared to either *Broadcast Mode test data volume/test application time* or *Serial Mode test data volume/test application time* and hence, we ignore it in our calculations.

3.6 Results

We will compare the efficiency of our heuristics with respect to the following two aspects:

- Effectiveness in improving the broadcast mode
- Effectiveness in reducing total test data volume and test application time

We will use the parameters *Fault Coverage* and *Number of ATPG Untestable Faults* for the first aspect (Figures 3.10 - 3.15) and the parameters *Total Test Data Volume (in bits)* and *Total Test Application Time (in cycles)* for the second aspect (Figures 3.16 - 3.21). In each of the plots, the value in the parenthesis for each circuit shows the number of scan chains in the broadcast mode.

In general, with the increasing number of scan chains in the broadcast mode, the fault coverage achievable in the broadcast mode may decrease. However, from the plots in the Figures 3.10 - 3.15, we can observe that our heuristics ($H2$, $H3$ *and* $H4$) obtain a significant gain in the fault coverage for increasing number of scan chains over the other two methods - Random ($H0$) and test-set based from UIUC ($H1$). For instance, for $s9234$, $s13207$ and 15850 runs, our heuristics achieve a fault coverage gain in the range of 10%-20%. This is because the other two methods may be significantly suffering from the *unreachable state*

issue (refer to Section 2.3.1) in the broadcast mode. However, for such cases, our heuristics are able to efficiently avoid such states and thereby obtain higher fault coverage.

Tables 3.3-3.6 compare the average gain achieved for each of the heuristics - $H0$, $H1$, $H2$, $H3$, $H4$ and $H5$ - across different configurations. The first column indicates the configuration as explained in Section 3.5. The next five columns indicate the average value of parameter of interest for all the benchmark circuits considered[4]. Next column shows the relative average gain of $H1$ over $H0$. We choose the *Dependency based* heuristic $H2$ as the representative among our proposed heuristic and the last two columns show the relative average gain of $H2$ with respect to $H0$ and $H1$ respectively.

An an interesting observation, from all of the plots, is that the test-set based heuristic $H1$ fares poorly for the original benchmark circuit (say $s13207$) than its *corrected* version ($s13207\_1$). Since we do not know what is the error in the original circuit and how it is corrected, we simply consider both the versions and report results for them. In some cases, the difference in performance is quite significant and hence, this difference affects $H1$'s average performance measure. This explains why average Fault Coverage and number of AU faults for $H1$ for configuration in *C1* is worse than $H0$. Such an effect is not seen for any of our proposed topology based heuristics. This further suggests that circuit topology based guidance is better than the test-set based as the *incompatibility* measure in a topology based approach is more comprehensive than in a test-set based approach. This is simply due to the fact that the *incompatibility* relations computed using a test-set based method are just the subset of the actual *incompatibility* relations.

From the Tables 3.3 and 3.4, it can be observed that $H1$ achieves minor reduction in the broadcast mode #AU faults and corresponding minor increase in the fault coverage over the $H0$ (random) method. This fails to justify the computational efforts required by $H1$ in processing the test patterns (without any test set compaction) for generation of scan chain configurations. Further, from the last two columns in the corresponding tables, it can be observed that our method achieves reduction of around 800 AU faults and gain

[4]results for the circuit $s5378$ are not available for the heuristics $H3$ and $H4$

of approximately 5% (over both $H0$ and $H1$) across different scan chain configurations. Note that our heuristic $H2$ just processes the fanout cone of each flip-flop to obtain the compatibility information. Further, we do not require an a-priori ATPG run (unlike $H1$). This justifies the efficiency and the low-cost nature of our method.

If we compare the average performance between proposed circuit topology based heuristics, we see that the SCOAP based heuristic $H4$ performs better (albeit marginally) than the Fanout Estimate based heuristic $H3$. This is in accordance with the motivation described for $H4$ in Section 3.4.3. When compared with $H3$ and $H4$, again $H2$ performs little bit better. This can be explained with the following reasoning: The heuristic techniques used by both $H3$ and $H4$ work flawlessly for the fanout free regions but may cause a *bit-off* guidance when reconvergent fanout region is encountered. The number of such reconvergent fanout regions may not be significant in the initial grouping iterations by virtue of smaller fanout cones involved; but it may be significant in the last iterations when *Groups* with relatively bigger fanout regions are merged. In such cases, the guidance accuracy may dip a bit and we might end up a bit more AU faults and consequently, with a bit less fault coverage. In contrast, the heuristic measure for $H2$ doesn't have any such limitation and hence, its guidance is always at the best possible level.

The fact that all three proposed methods yield nearly similar results also points to the inherent efficiency of the underlying *Group* formation process which first builds large number of small but fully *self-compatible Groups* and then merges the *Groups* with minimum *incompatibility* together to form bigger *Groups* with the required cardinality. The experimental results show that this approach is quite effective and thus, can achieve very good quality of results even with the simplest and computationally least expensive *Dependency*-based heuristic $H2$.

Tables 3.5 and 3.6 show the similar comparison between the heuristics for the average percentage reduction achieved in total test data volume and test application time. Since applying a broadcast mode pattern is more efficient than a serial mode pattern, it is intuitive that the higher broadcast mode fault coverage will also lead to the higher savings in the test

data volume and test application time. The plots in the Figures 3.16 - 3.21 and the average data presented in the Tables 3.5 - 3.6 confirm that it is indeed the case. Again we can observe that, relative to $H0$, the average reduction achieved using the test-based heuristic $H1$ is meager around 2% while, for $H2$, it is whopping 17%.

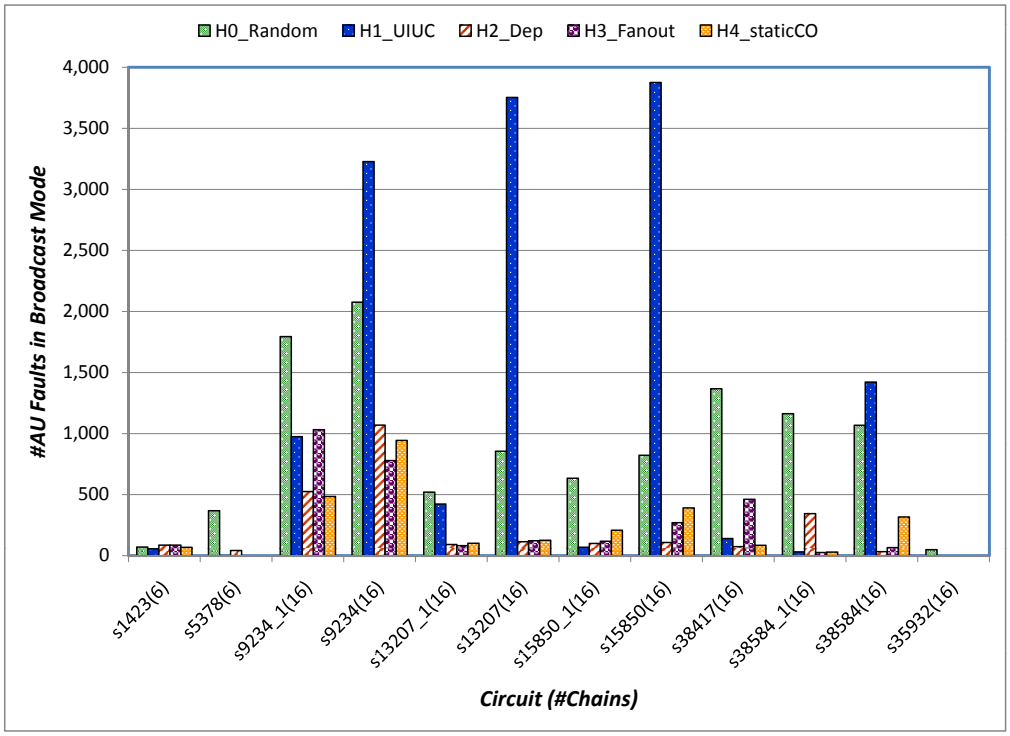
Also note that for $H1$, the gain in the percentage reduction in test data volume and test application time realtive to *H0* is comparable to the gain in the broadcast mode fault coverage; but for $H2$, the percentage gain in test data volume and test application time is almost $3X$ than the gain in the broadcast mode fault coverage. This implies that the configurations generated using our method are of better quality as the configurations not only just detect more number of faults but also detect them using fewer patterns. This further proves the effectiveness of configurations achieved using our proposed methods.

Table 3.3: Average Broadcast Mode #AU Faults

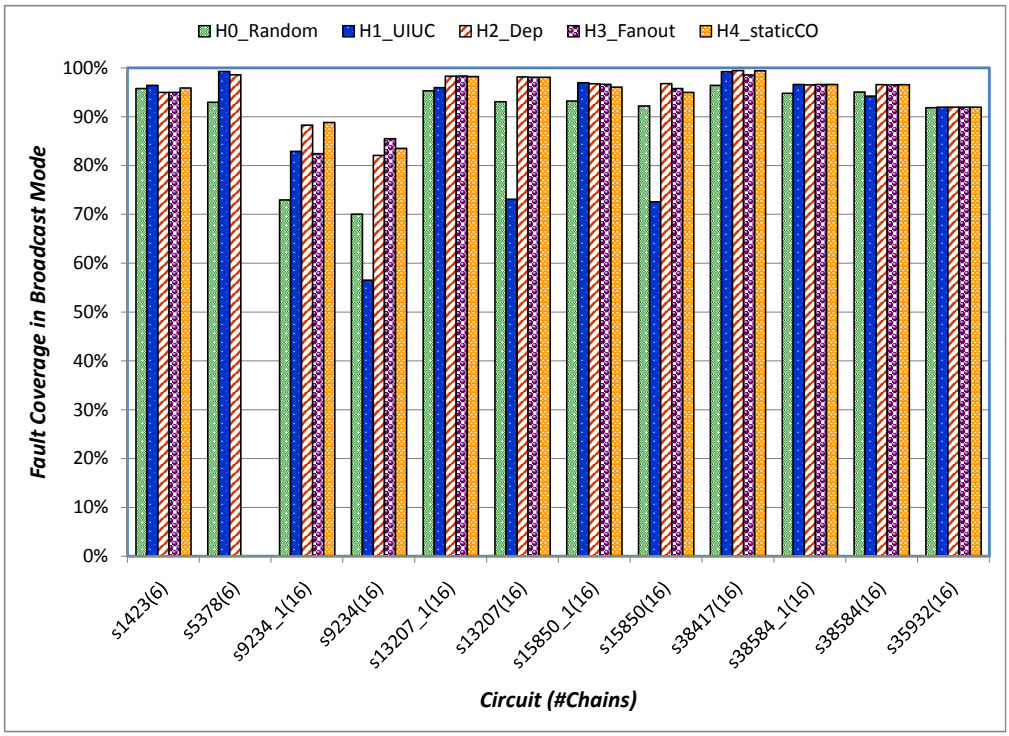
Conf	H0_Rand	H1_UIUC	H2_Dep	H3_Fanout	H4_SCOAP	H0-H1	H0-H2	H1-H2
C1	899	1,165	216	277	250	-266	683	949
C2	1,460	1,363	590	851	606	97	870	773
C3	1,927	1,833	865	1,411	1,125	94	1,062	968
C4	2,207	2,006	1,147	1,847	1,418	200	1,060	860
C5	2,605	2,138	1,510	1,960	1,788	466	1,094	628
C6	2,831	2,356	1,744	2,384	2,050	474	1,087	612

Table 3.4: Average Broadcast Mode Fault Coverage

Conf	H0_Rand	H1_UIUC	H2_Dep	H3_Fanout	H4_SCOAP	H1-H0	H2-H0	H2-H1
C1	90.31%	87.98%	94.87%	94.13%	94.56%	-2.33%	4.56%	6.89%
C2	85.88%	85.82%	91.57%	89.58%	91.45%	-0.06%	5.70%	5.76%
C3	82.84%	83.36%	88.83%	85.65%	87.45%	0.52%	5.99%	5.47%
C4	81.30%	81.97%	87.17%	83.48%	85.35%	0.67%	5.87%	5.20%
C5	79.00%	81.02%	84.45%	82.53%	83.05%	2.02%	5.46%	3.43%
C6	77.67%	79.96%	82.55%	80.52%	81.49%	2.29%	4.88%	2.59%

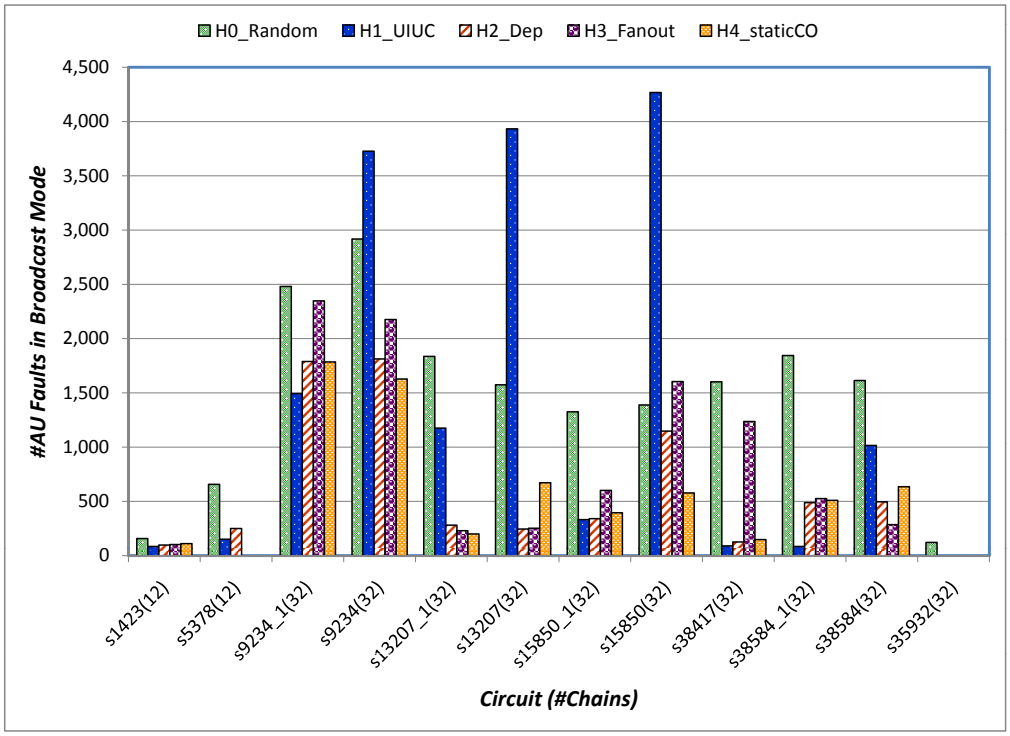


(a) #AU Faults in *C1*

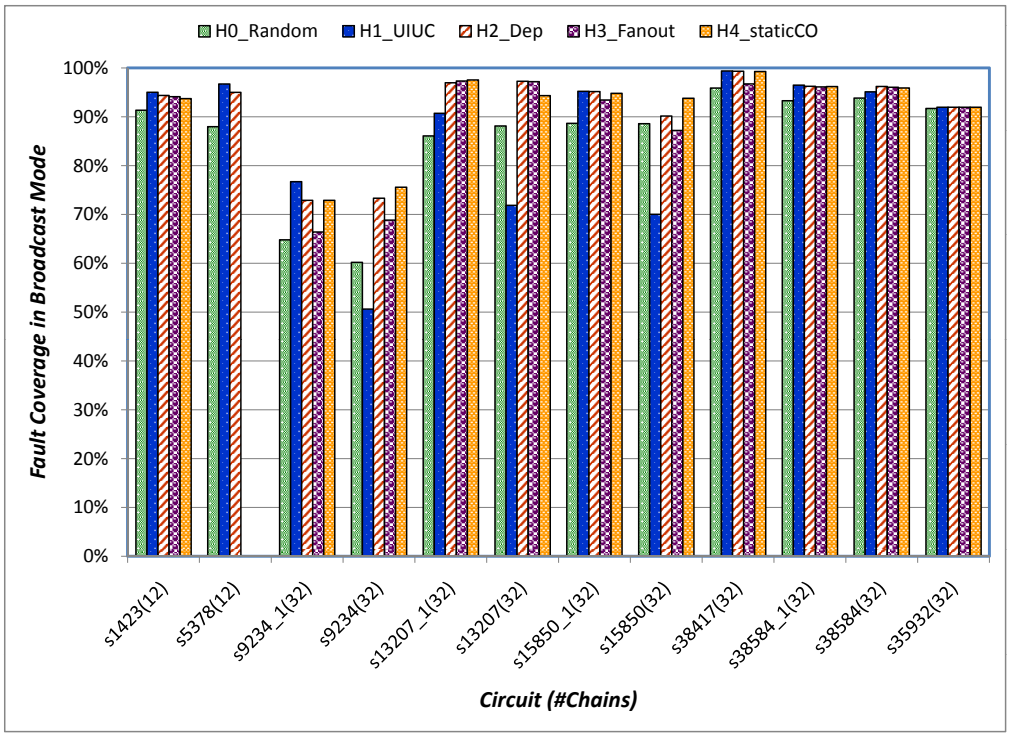


(b) Fault Coverage in *C1*

Figure 3.10: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration $C1$

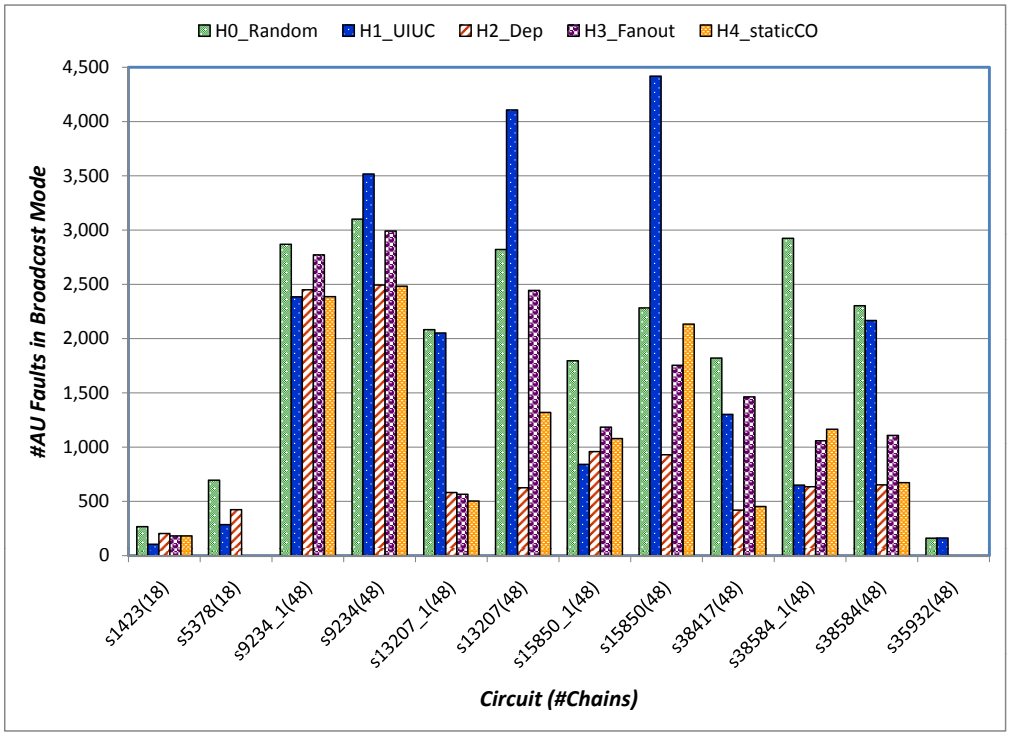


(a) #AU Faults in *C2*

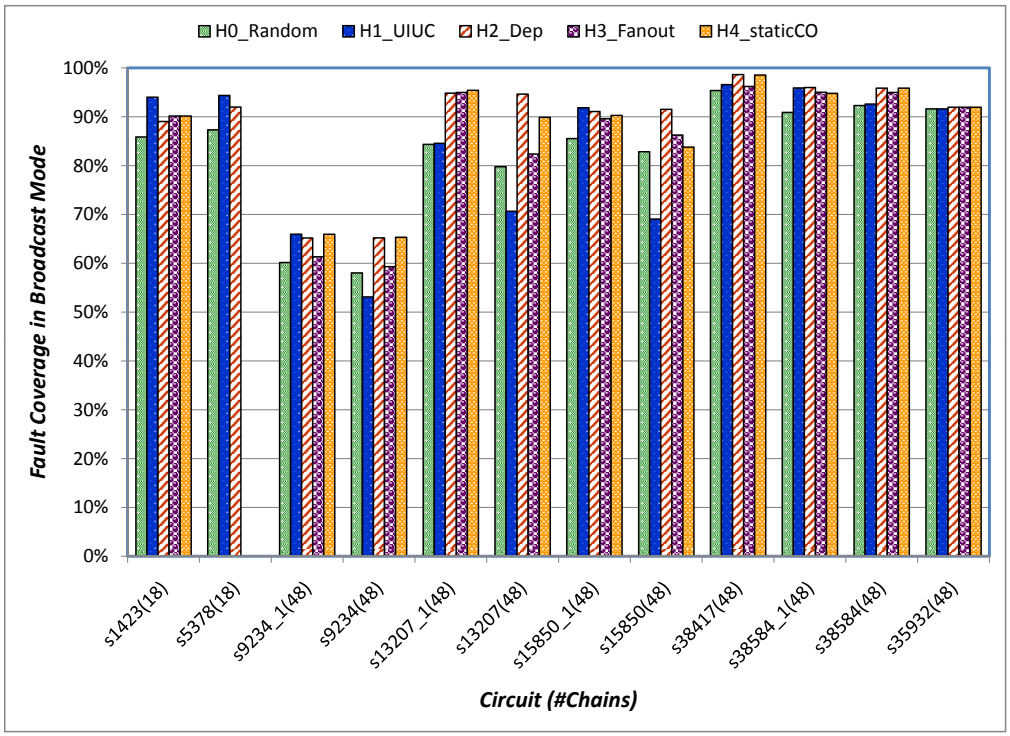


(b) Fault Coverage in *C2*

Figure 3.11: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration $C2$

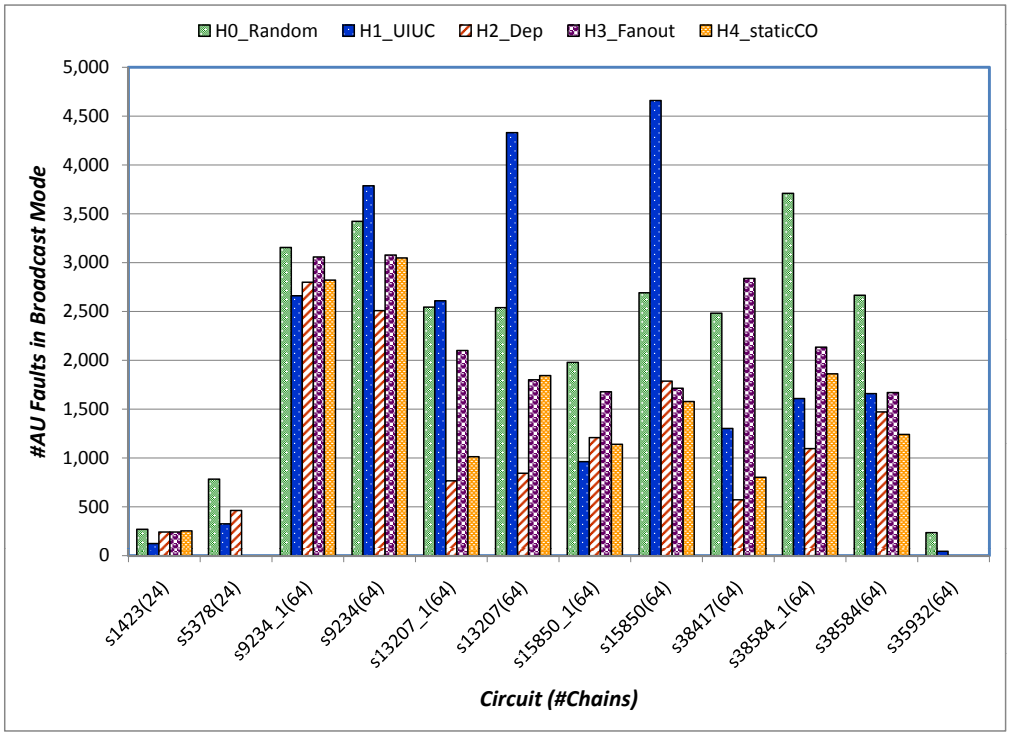


(a) #AU Faults in *C3*

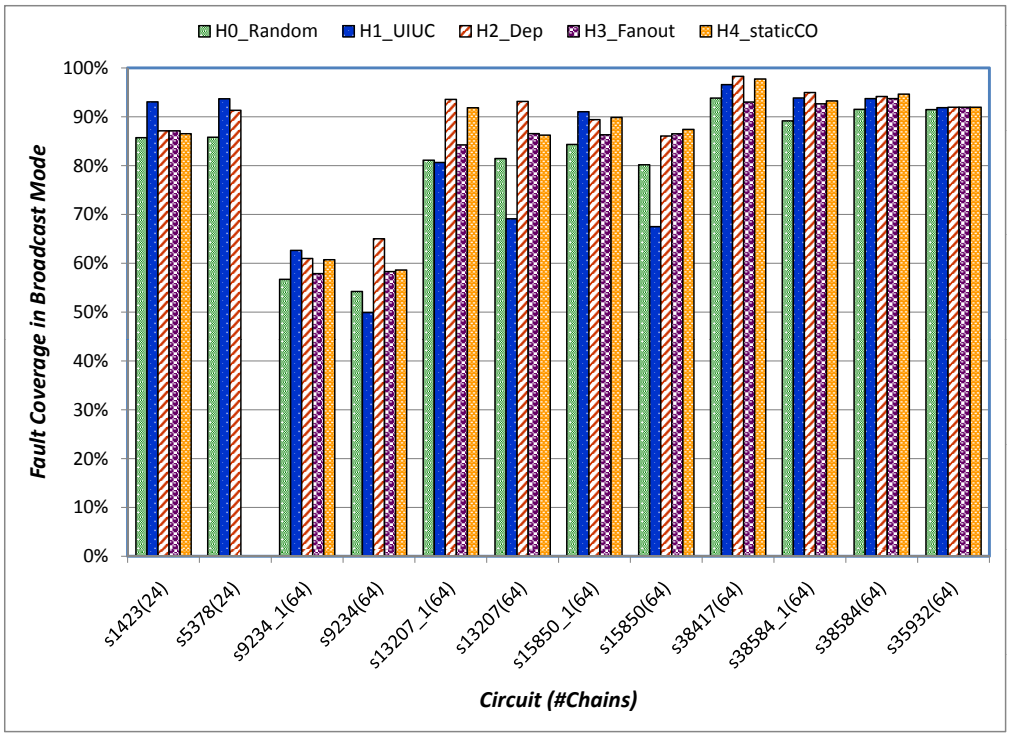


(b) Fault Coverage in *C3*

Figure 3.12: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration $C3$

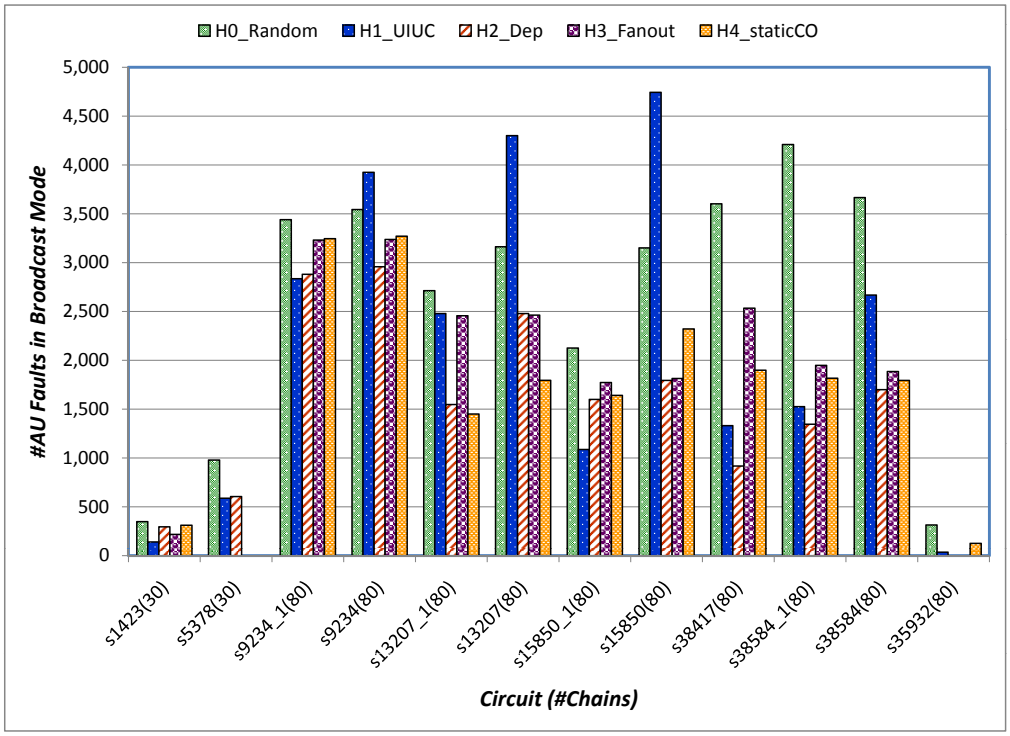


(a) #AU Faults in *C4*

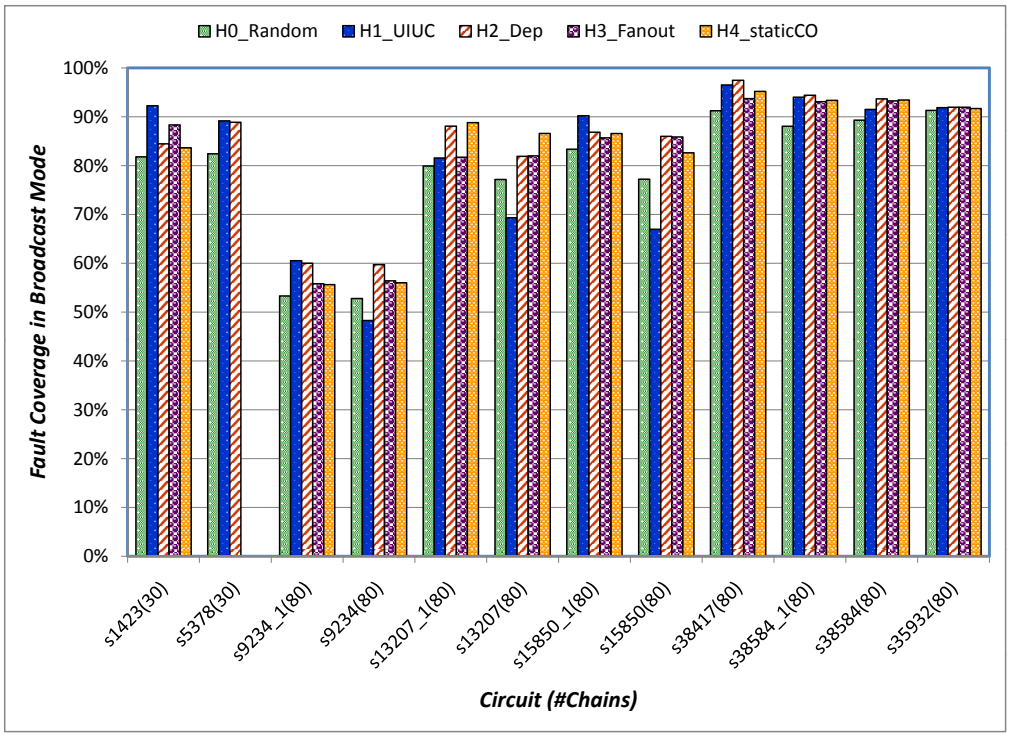


(b) Fault Coverage in *C4*

Figure 3.13: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration *C4*

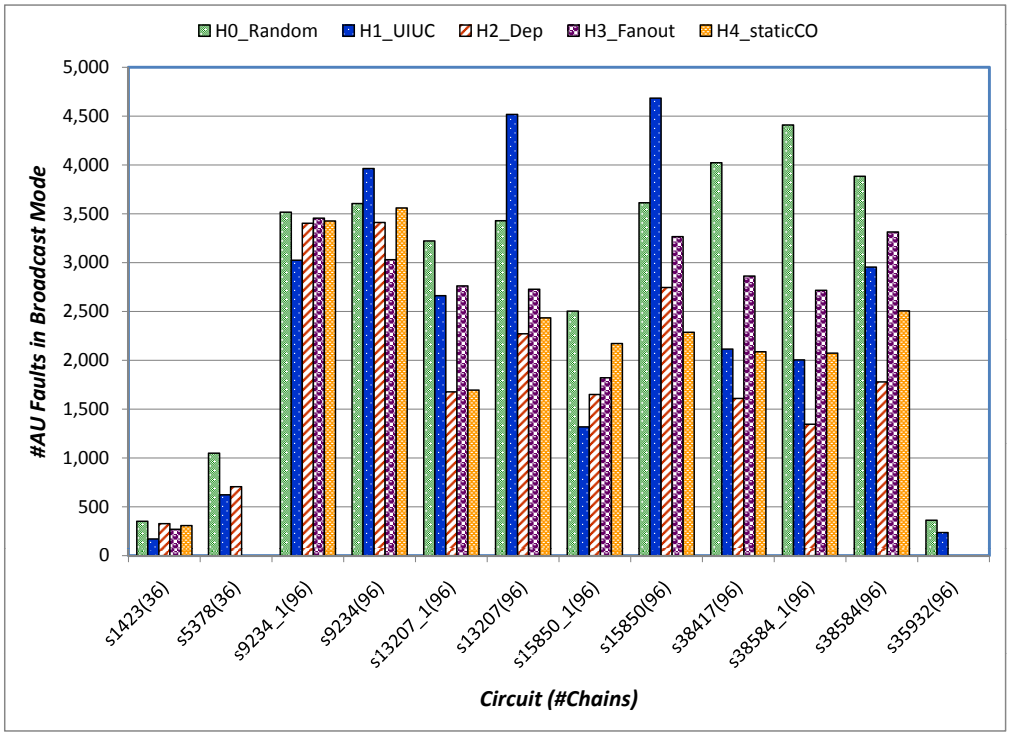


(a) #AU Faults in *C5*

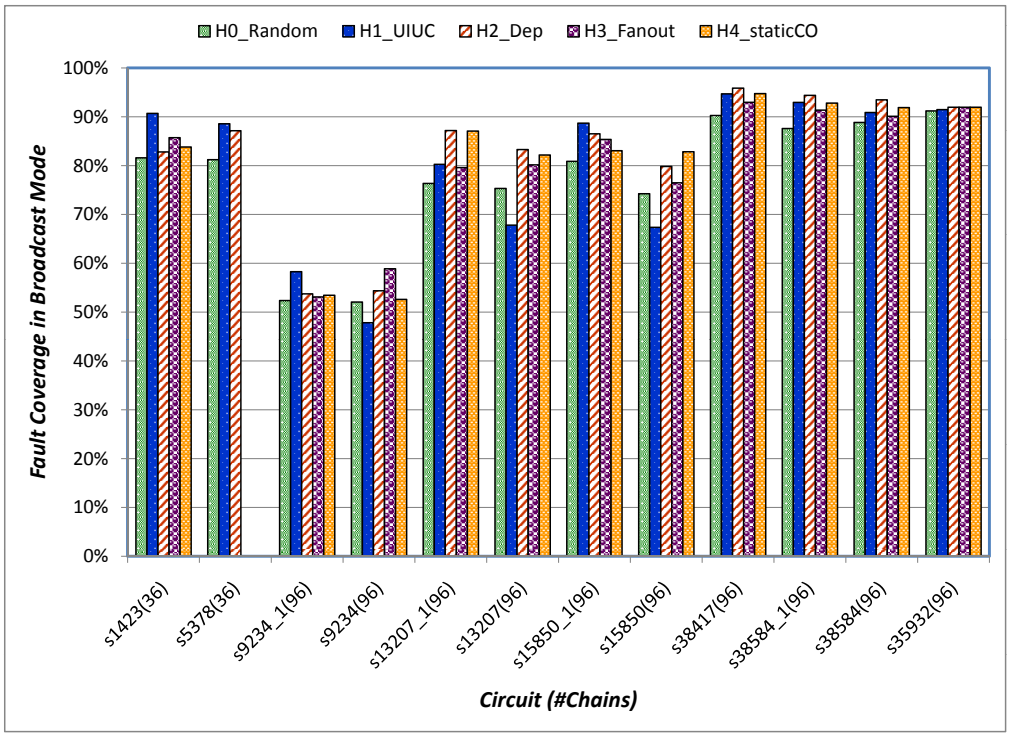


(b) Fault Coverage in *C5*

Figure 3.14: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration $C5$

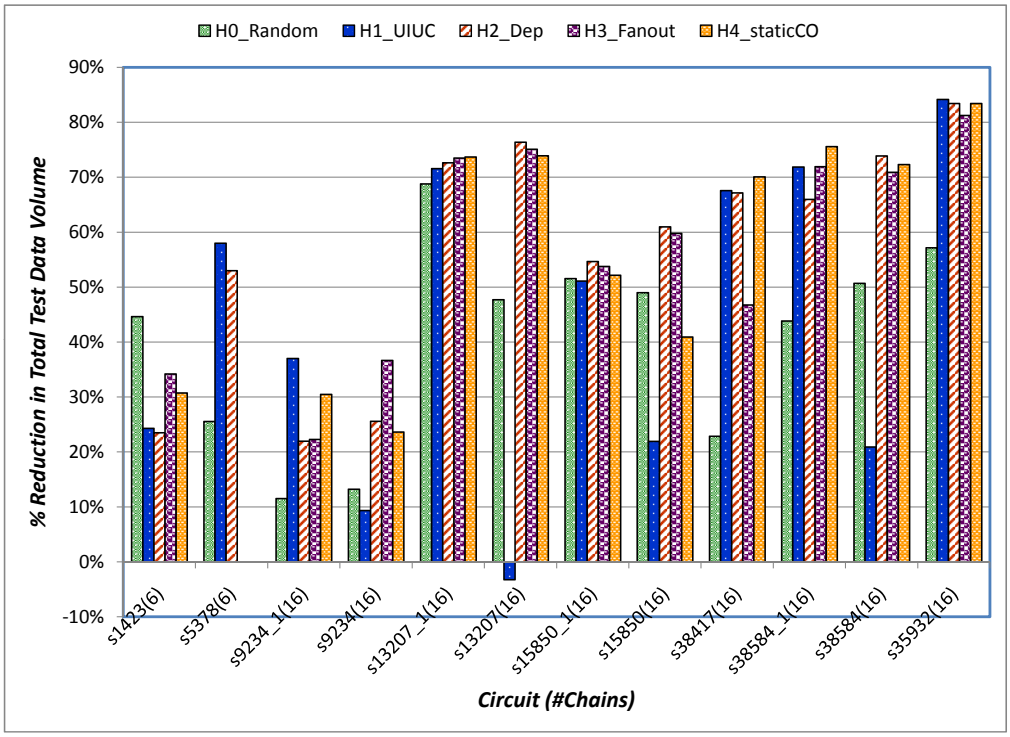


(a) #AU Faults in *C6*

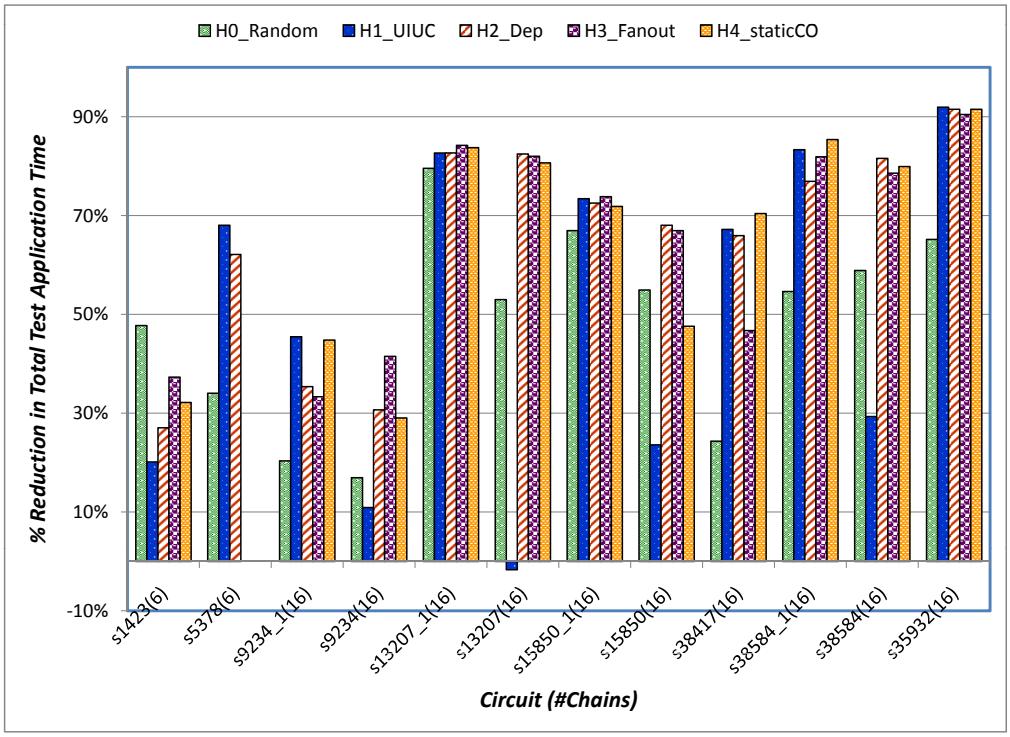


(b) Fault Coverage in *C6*

Figure 3.15: Comparison of #AU Faults and Fault Coverage in Broadcast Mode for configuration *C*6

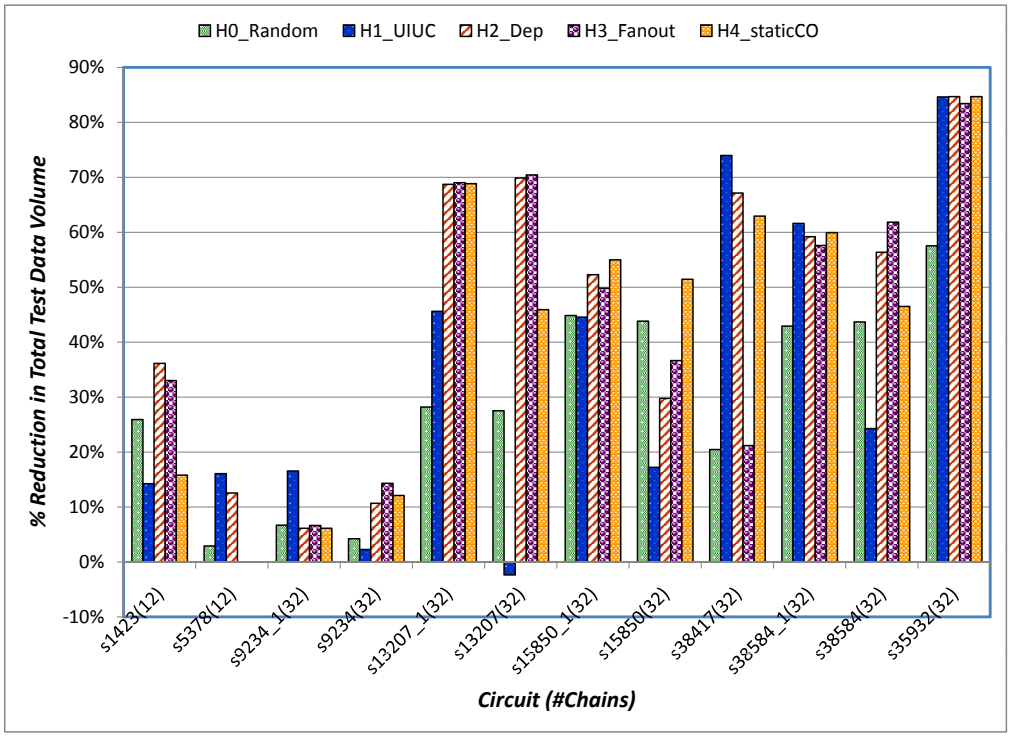


(a) TDV Reduction for *C1*

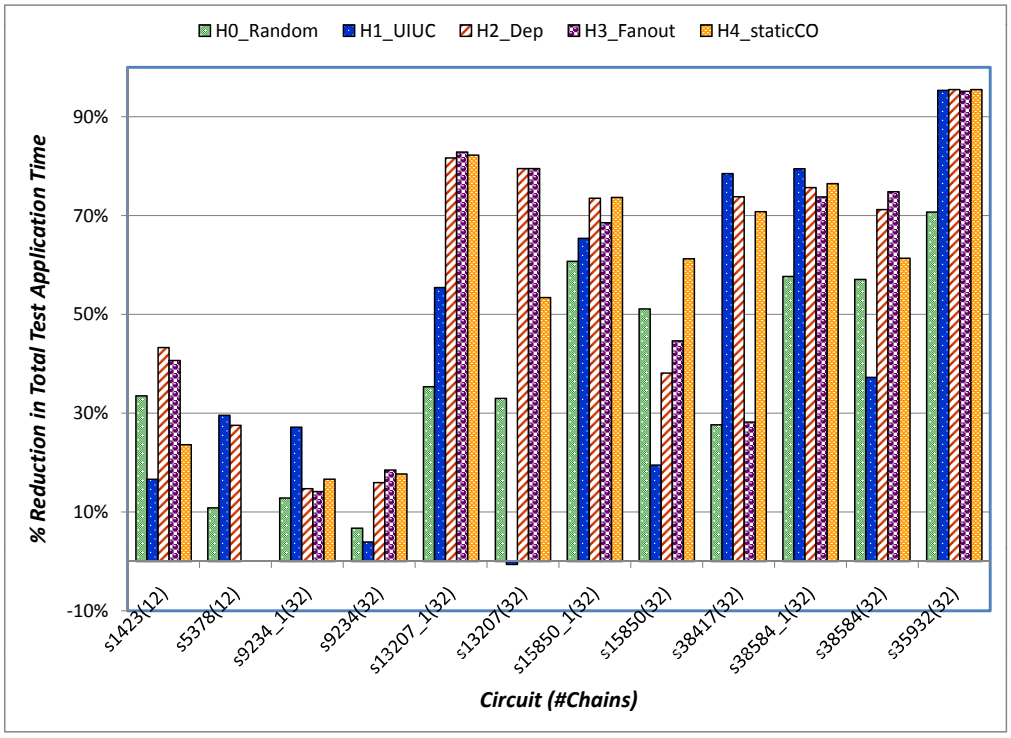


(b) TAT Reduction for *C1*

Figure 3.16: Comparison of reduction in TDV and TAT for configuration $C1$

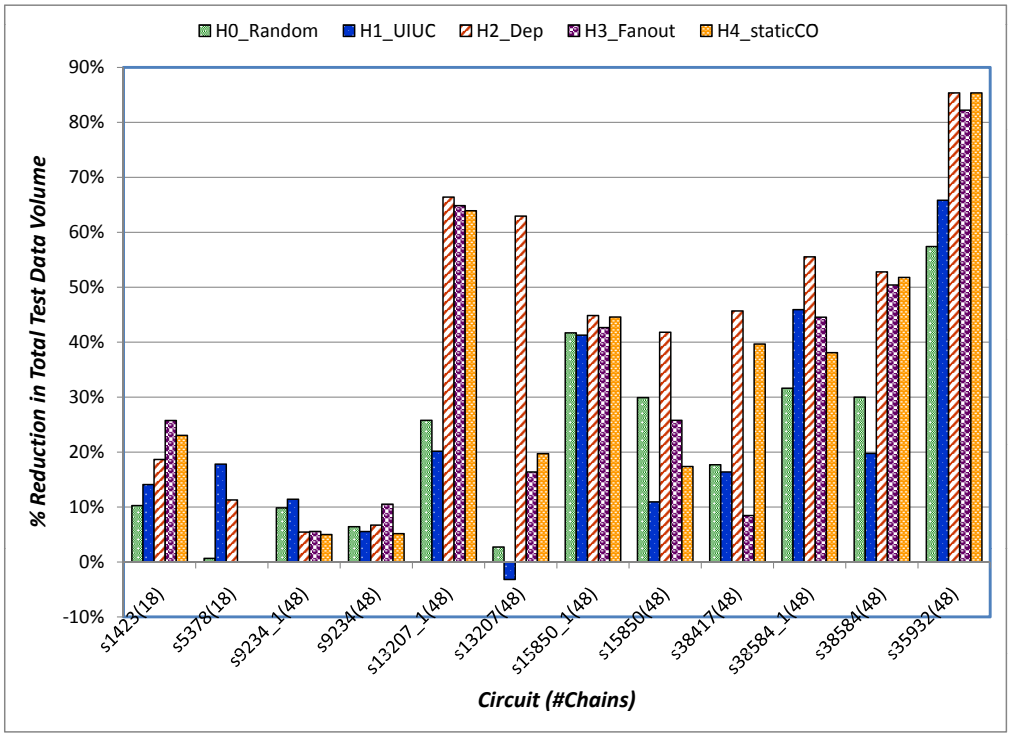


(a) TDV Reduction for *C2*

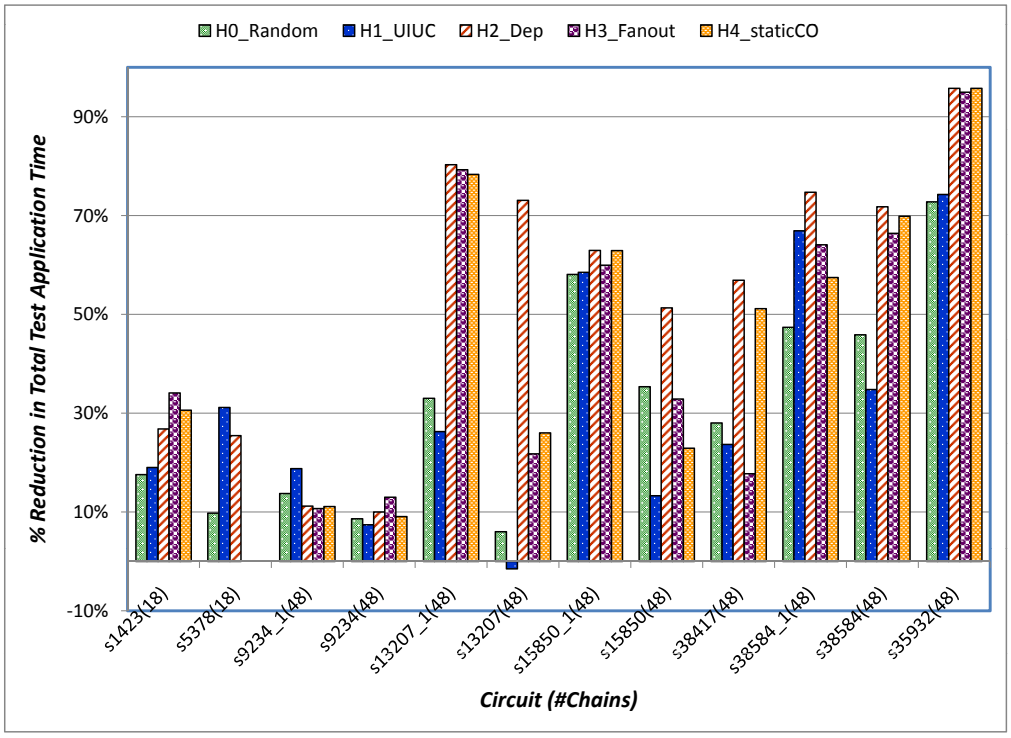


(b) TAT Reduction for *C2*

Figure 3.17: Comparison of reduction in TDV and TAT for configuration $C2$

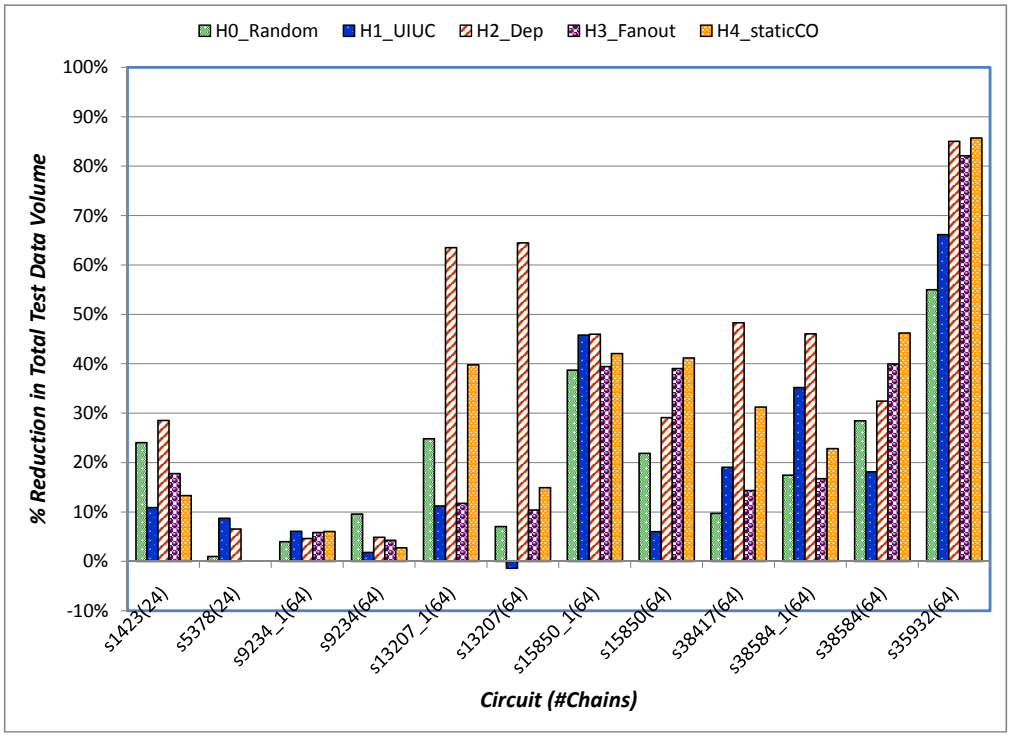


(a) TDV Reduction for *C3*

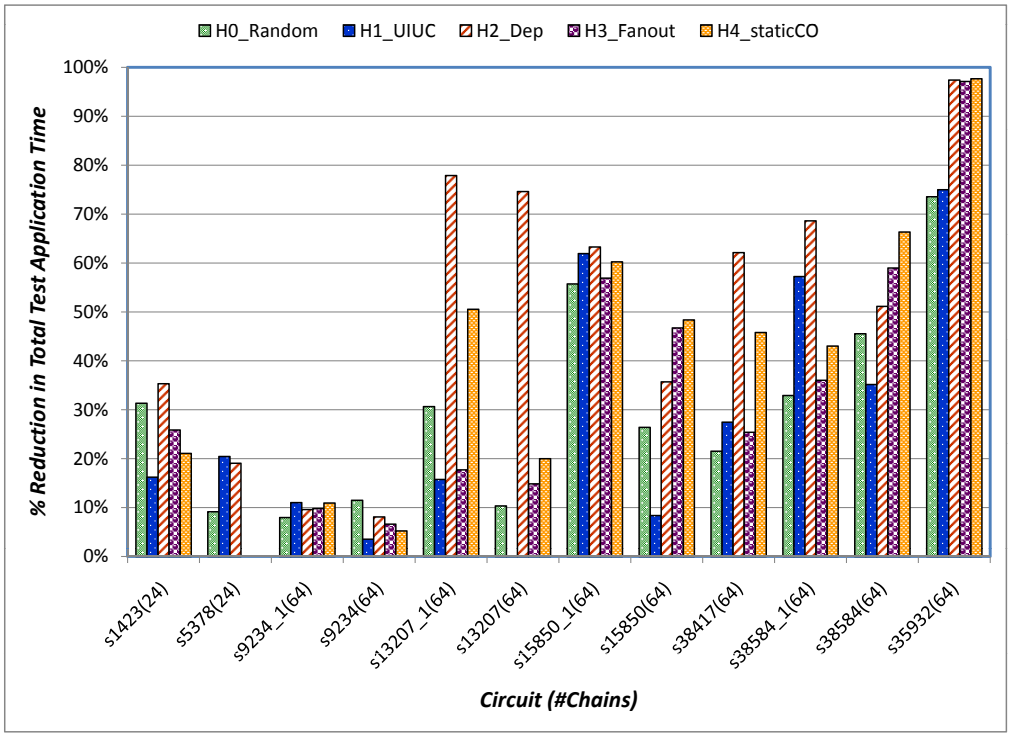


(b) TAT Reduction for *C3*

Figure 3.18: Comparison of reduction in TDV and TAT for configuration $C3$

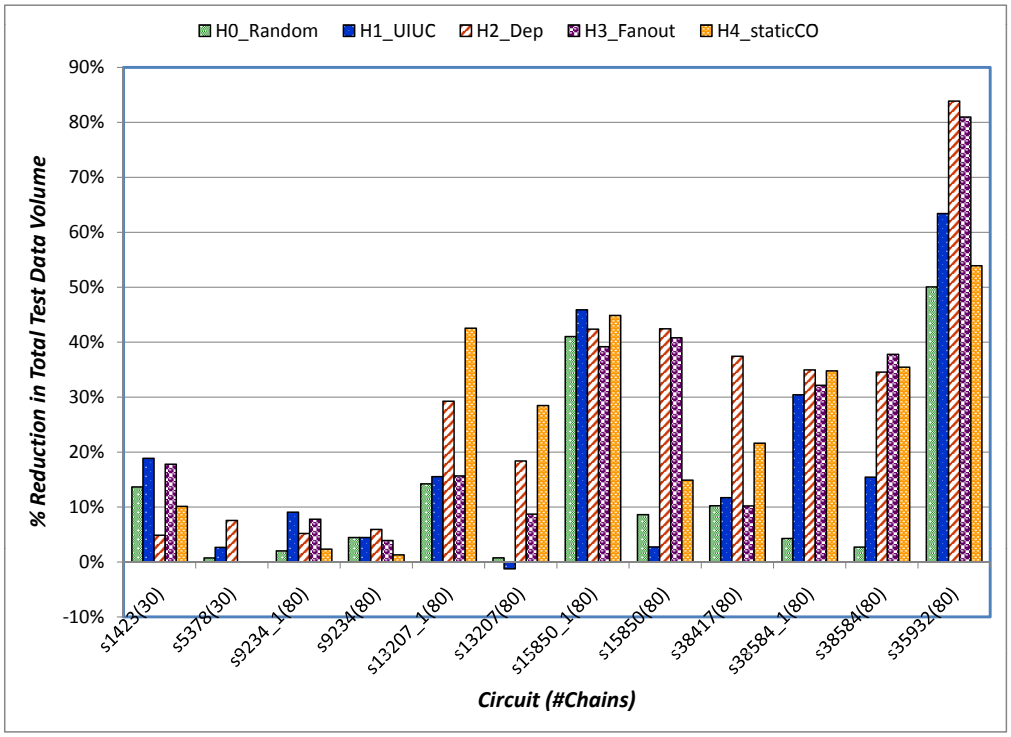


(a) TDV Reduction for *C4*

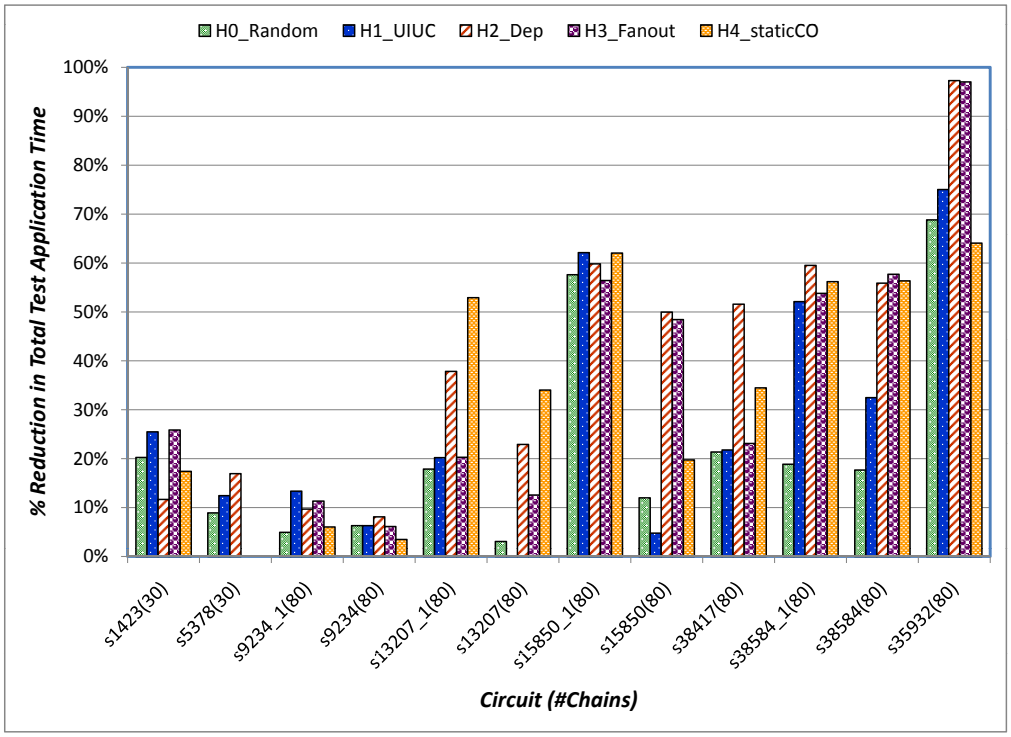


(b) TAT Reduction for *C4*

Figure 3.19: Comparison of reduction in TDV and TAT for configuration $C4$

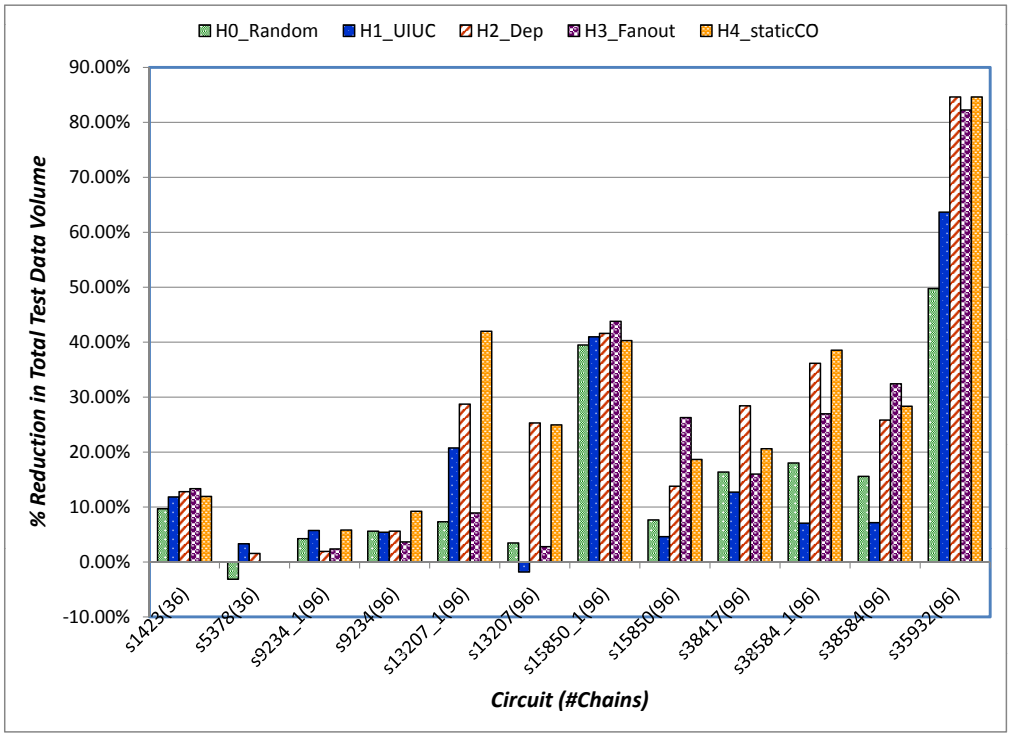


(a) TDV Reduction for *C5*

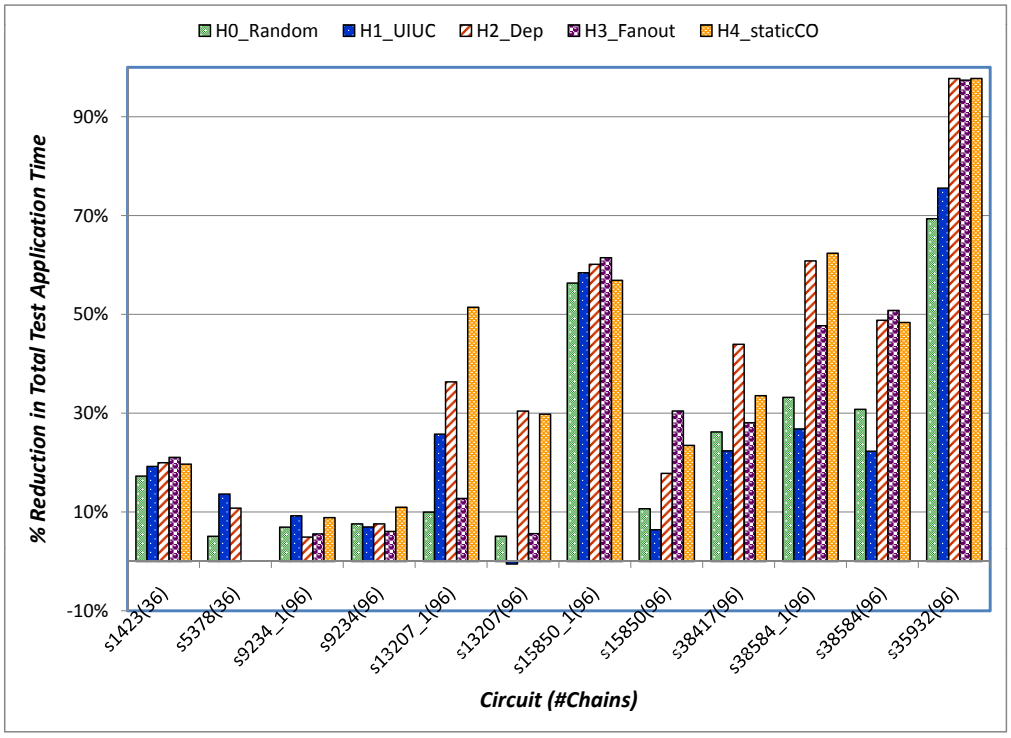


(b) TAT Reduction for *C5*

Figure 3.20: Comparison of reduction in TDV and TAT for configuration $C5$



(a) TDV Reduction for *C6*



(b) TAT Reduction for *C6*

Figure 3.21: Comparison of reduction in TDV and TAT for configurations $C6$

Table 3.5: Average % Reduction in TDV

Conf	H0_Rand	H1_UIUC	H2_Dep	H3_Fanout	H4_SCOAP	H1-H0	H2-H0	H2-H1
C1	40.55%	42.89%	56.60%	56.93%	57.00%	2.33%	16.04%	13.71%
C2	29.07%	33.23%	46.14%	45.82%	46.31%	4.16%	17.07%	12.90%
C3	22.00%	22.16%	41.46%	34.29%	35.80%	0.16%	19.46%	19.30%
C4	20.16%	18.99%	38.30%	25.62%	31.47%	-1.16%	18.14%	19.31%
C5	12.73%	18.25%	28.91%	26.82%	26.39%	5.52%	16.17%	10.66%
C6	14.51%	15.12%	25.53%	23.53%	29.55%	0.62%	11.03%	10.41%

Table 3.6: Average % Reduction in TAT

Conf	H0_Rand	H1_UIUC	H2_Dep	H3_Fanout	H4_SCOAP	H1-H0	H2-H0	H2-H1
C1	48.05%	49.53%	64.75%	65.17%	65.20%	1.48%	16.69%	15.21%
C2	38.10%	42.30%	57.55%	56.44%	57.52%	4.20%	19.44%	15.25%
C3	31.36%	31.06%	53.36%	44.99%	46.84%	-0.30%	22.00%	22.30%
C4	29.71%	27.69%	50.24%	36.00%	42.65%	-2.03%	20.52%	22.55%
C5	21.47%	27.19%	40.10%	37.51%	36.98%	5.71%	18.63%	12.91%
C6	23.21%	23.85%	36.61%	33.36%	40.29%	0.64%	13.40%	12.76%

3.7 Summary

In this chapter we presented a novel topology-based analysis for generating a low cost scan chain configuration in the ILS architecture that is effective in getting the high fault coverage in the broadcast mode. Our method obviates the need of an a-priori ATPG run to configure the scan chains *effectively* which saves the design time and makes our approach scalable and easy to adopt in the conventional scan chain insertion flow. We proposed three novel heuristics to assign the position of flip-flops in different scan chains so as to minimize the number of broadcast mode undetectable faults. Experimental results on a wide range of ISCAS'89 circuits show that this approach achieves a high fault coverage. Our method achieves an average gain of 5% over the existing methods across several scan chain configurations. Since the higher broadcast mode fault coverage requires less number of expensive *serial mode* patterns, we also achieve a substantial reduction - on average 15% more than existing methods - in the total test data volume and test application time. This work can be further enhanced by considering the routing and placement requirements for the proposed

configuration of the scan chains. Also, the Fanout Estimate based heuristic and SCOAP based heuristic can be improved by accounting for the reconvergent fanout cones. Another important factor for structural testing is limiting the power dissipation during the test mode. Current methods for this are minimizing the toggle activity and/or shutting-off some of the chains during the scan-chain shifting process. The proposed framework can be enhanced to consider these features also.

Chapter 4

Hybrid ILS-RAS Architecture

In this chapter, we propose a new scan architecture to further maximize the gains of the conventional Illinois Scan Architecture. This architecture combines the *broadcast mode* of the conventional ILS with the Broadcast-enabled Partial Random Access Scan to eliminate the need for the *serial mode* of the conventional ILS.[1] Since the *serial mode* of the conventional ILS is very inefficient from the test data volume and test application point of view, it follows that the proposed new architecture will enable further reduction in the test data volume and test application time. In this chapter, we describe the proposed architecture, test generation and test application procedure and show that indeed significant reduction in test data volume and test application time can be achieved for the large benchmark circuits with reasonable increase in the area overhead. Specifically, for the bigger benchmark circuits [2], for the best case area overhead of 1.69%, the proposed Hybrid Architecture achieves 27% more reduction in the test data volume and 37% reduction in the test application time over the *effectively configured* conventional ILS. With the additional area overhead, we do get additional reduction in the test data volume and test application time but with diminishing scale.

[1] Here *Broadcast-enabled* means a given address can select multiple flip-flops and *Partial* means only portion of the flip-flops are addressable.

[2] circuits with at-least 1000 flip-flops

4.1 Motivation

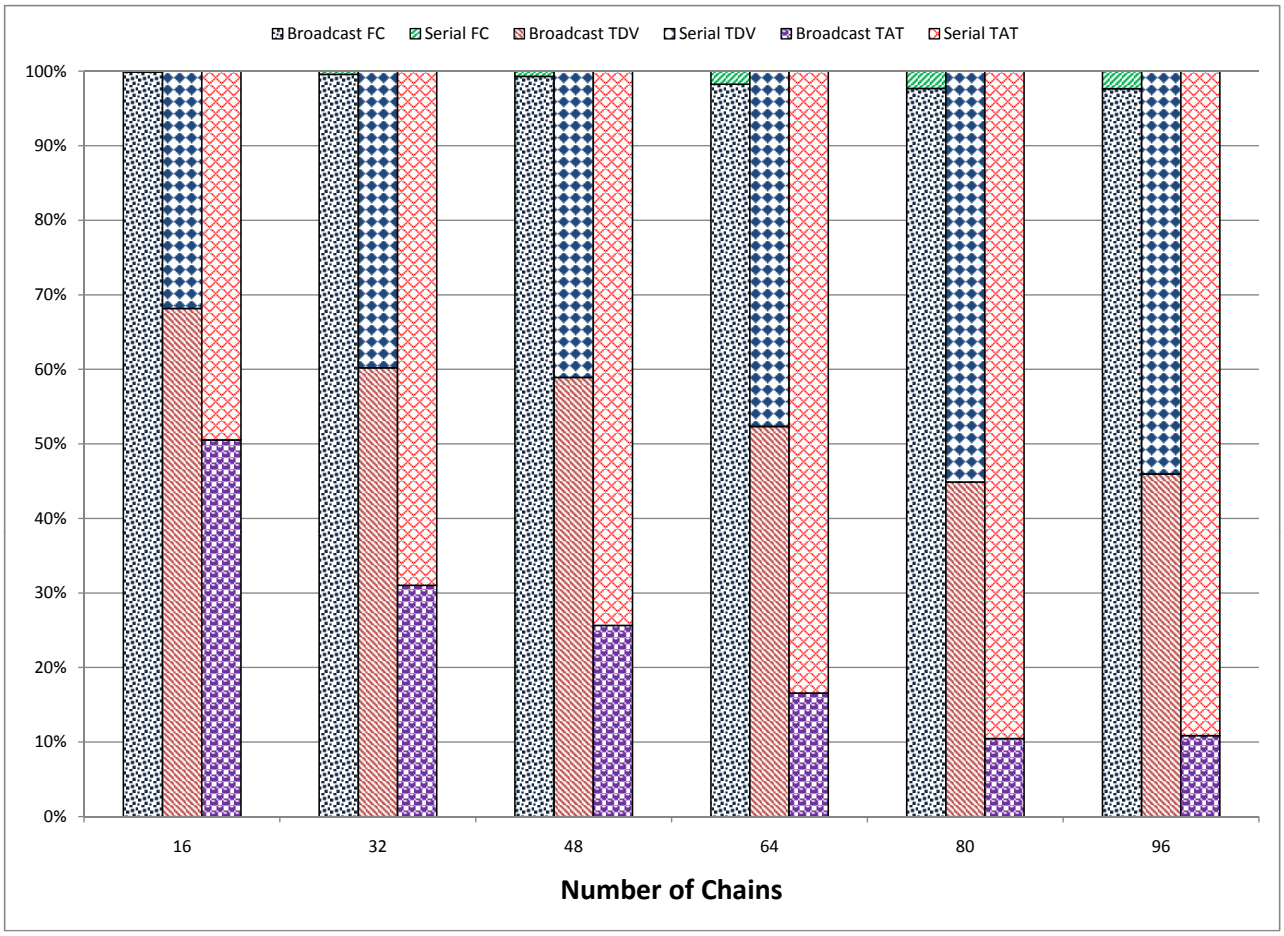


Figure 4.1: Normalized Fault Coverage, TDV and TAT for various number of chains

Figure 4.1 compares the percentage contribution of the broadcast mode and the serial mode of the conventional Illinois Scan Architecture with respect to the parameters fault coverage, test data volume and test application time for the circuit *s38584.1* for six different number of chains. All these configurations are generated using our *Dependency-based heuristic* described in Section 3.4.1. From the chart it follows that even though the *serial mode* is required to cover a very small percentage of the total testable faults, the test data volume and test application time requirements for it are too high when compared to the broadcast mode. This is due to the fact that for each serial mode pattern, we need to shift through

all the flip-flops in the design. Thus, it corroborates the intuition that maximizing the fault coverage in the broadcast mode is the key to maximize the ILS gains. Now since the heuristic used to generate this chart has been shown to give the best results (refer to Section 3.6), the chart shows the best case of test data volume and test application time reduction achievable for the conventional ILS. Thus, it follows that if we want to extract even more test data volume and test application time reductions, we need to minimize the test data volume and test application time requirements to cover those broadcast mode ATPG untestable faults. We propose a variation of the Random Access Scan (RAS) architecture for this purpose which obviates the need for the *serial mode* altogether and thus, enables further significant reductions in the test data volume and test application time requirements.

As described in Section 2.2.2, RAS provides the ability to individually control and observe each flip-flop directly from the top-level. While such fine granularity may be helpful to reduce the test data volume and test application time for those patterns that require few bit flips, it can lead to very high area and routing overhead due to the size of address decoder needed. Also, most of the traditional RAS architectures depend on the flip-flop capture values from the previous pattern to determine the set of flip-flops that need to be loaded for the current pattern [33, 38, 35, 32]. If a large number of flip-flops need to be loaded for each pattern, then this may reduce the savings in the test data volume and test application time due to the need to address many flip-flops. Note that irrespective of the method of test application viz. serial scan or random access scan, the underlying test patterns contain very few *care bits*[3] that are actually needed to detect the faults. Hence, it follows that a Broadcast method for RAS that is analogous to Broadcast in serial scan may be helpful. In [38], the authors implement this exact idea and use the same graph coloring method, described in Section 2.3.3.3 to configure the ILS, to build the RAS Broadcast groups. Note that here all the flip-flops are addressable.

In our proposed architecture, we also use the RAS broadcast groups but just for the *ATPG untestable faults* in the ILS broadcast mode. This approach differs from [38] in

[3]logic 1 or 0 value assignments to primary inputs and pseudo primary inputs.

the sense that we employ RAS as the supplementary architecture rather than the primary architecture. Hence, we require far less number of flip-flops to be addressable which in turn leads to a less number of broadcast groups. This reduces the address decoder area overhead. Although only a portion of flip-flops is addressable in the RAS mode and used to apply the pattern, all the flip-flops are used to capture the circuit response to the applied pattern. This ensures that we don't compromise on the circuit observability front. Also, we reuse the existing chains and compactor logic of the ILS to shift and compress the captured response. This will further reduce the compactor and routing overhead as individual flip-flop outputs are no longer required to be routed to the compactor logic. Note that shifting in the next pattern together with the shift-out operation will not be helpful, since we are targeting the broadcast mode untestable faults. This explains the motivation and salient features of our proposed architecture. Subsequent sections describes the actual architecture in detail and also elaborates on the test generation and application procedures.

4.2 Architecture Overview

Here we give high level conceptual view of the proposed new hybrid architecture. Actual configuration details are given in the next section. The proposed hybrid scan architecture has two distinct test modes and both of these modes broadcast the scan values to the scan flip-flops. The first mode is *Serial Scan Broadcast Mode* (*SSB*) and the second is *Random Access Scan Broadcast Mode* (*RASB*). These modes are defined by a dedicated top-level test pin called *BM_SELECT* which stands for broadcast mode select. When *BM_SELECT* is 0, *SSB* is in effect and when *BM_SELECT* is 1, *RASB* is in effect. The *SSB* mode is the primary testing mode and the *RASB* mode is the supplementary testing mode. The *SSB* Mode can be any broadcast based serial scan architecture (see Section 2.3). In our work, we consider the broadcast mode of Illinois Scan Architecture as the *SSB* mode. We test as many faults as possible in the *SSB* mode and test the remaining undetected faults in the *RASB* mode. We reuse the scan chains and the compactor logic of the *SSB* mode in the

RASB mode to observe and shift out the circuit response to the applied *RASB* pattern.

In our experimental setup, we assume a *full-scan* setup, i.e., all the flip-flops in the design are scan flip-flops and are part of the scan chains in the *SSB* mode. We compute the subset S_{RAS} of these flip-flops such that when the flip-flops of this set are made addressable, they can detect all the *SSB* mode untestable faults in the circuit. By addressability we mean the selected flip-flops can be loaded with the required values directly from the top-level without any scan-chain shifting. To compute this subset of flip-flops, we use a test-set based approach (described in Section 4.3). The test-set based approach used guarantees that all the detectable faults will be detected. The set S_{RAS} is then subdivided into multiple (say N_{RASB}) smaller groups such that all member flip-flops of a given subgroup are fully compatible for the underlying test set. Each subgroup, henceforth called as a *RAS group*, is then assigned a unique address and thus, all the flip-flops belonging to this subgroup are loaded simultaneously in the *RASB* mode. Thus, we need an address decoder which can decode at-least N_{RASB} unique addresses.

In our architecture, we modify the *scan-input* connection for each member flip-flop of the set S_{RAS} so that it can be augmented with the addressing capability. Figure 4.2 shows the *si selector module* that is used to provide this addressability. This module contains two multiplexers - the first multiplexer selects between the *SSB* mode scan-input and *RASB* mode scan-input and the second multiplexer selects between the feedback from the current state of the flip-flop and output of the first multiplexer. Here, *SSB SI* is either top-level *SSB* mode scan-input signal (if it is the first flip-flop in the scan chain) or the output of previous flip-flop in the scan chain and *RASB SI* is the scan-input signal for the *RASB* mode. If in the *RASB* mode and the flip-flop is addressed (i.e., ADDR = 1), the first multiplexer selects the *RASB SI* which then gets passed through the second multiplexer also and flip-flop is loaded with the required value. For the flip-flops in the other groups, since the *ADDR* is not asserted and *BM_SELECT* is asserted, the flip-flop retains it's current value (which may be don't care or set by *RASB SI* earlier) through the feedback path in the second multiplexer. When we are in the *SSB Mode, BM_SELECT* is deasserted and the *SSB SI* has a direct

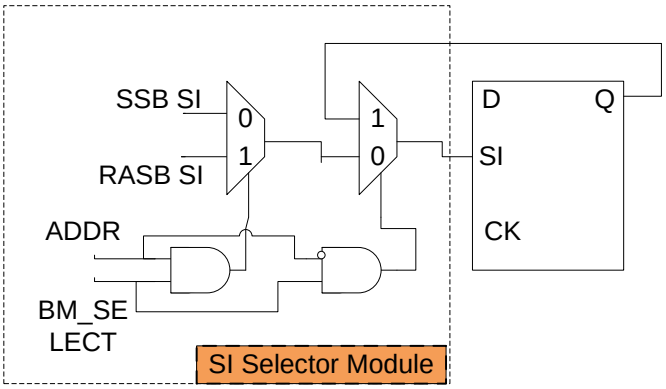


Figure 4.2: Augmenting regular scan flip-flop with addressing capability

path to the flip-flop's scan-input pin through both the multiplexers.

In our architecture, we assume that the addressing capability for the *RASB* mode is provided directly from the top-level, i.e., the input address pins for the address decoder can be controlled directly from the top-level. These pins can be dedicated or multiplexed with the functional input pins. The size of the address decoder is decided by the number of RAS groups[4] - we need a unique address for each group. The number of input address lines required is given by the equation

$$Number\,of\,Input\,Address\,Lines = \lceil \log_2(\#RAS\,Groups + 1) \rceil \tag{4.1}$$

Here, we assume a *one-hot* address decoder i.e., only one of the n output lines of the address decoder will be at logic high value at any given instance of time. If an input address of i is applied, $(i-1)^{th}$ output line will be asserted.

4.3 Architecture Configuration

In our work, we implement the broadcast mode of the conventional ILS as the *SSB* mode. Chapter 3 describes the various topology based heuristics to configure it *effectively* so that maximum savings in the test data volume and test application time can be achieved. In this

[4]it is actually number of RAS groups + 1 as the address 0 is reserved and is not considered to be a valid RAS address

work, we have used our *Dependency-based heuristic* for it (refer to Section 3.4.1).

To configure the *RASB* mode, we first need to populate the S_{RAS} set. For this purpose, we first perform the ATPG for the *SSB* mode with random filling of the don't care bits. Then, for the *SSB* mode ATPG untestable faults, we perform second run of ATPG with single scan chain and no filling of the don't care bits to produce the test set $T_{Serial} = \{t_1, t_2, \ldots, t_n\}$. All the flip-flops which are specified at-least once in the T_{serial} form the set S_{RAS}. Next, we populate the *Broadcast* groups for *RASB* mode. Towards this, we populate the set of *test-set wise compatible* flip-flops $T_{compatible} = \{T_{C1}, T_{C2}, \ldots, T_{Ck}\}$ based on the T_{serial} using the simple greedy algorithm described in [3]. Here, two flip-flops are said to be *test-set wise compatible* if for all the patterns t_i in the given test-set T, the values of two flip-flops are either the same or at-least one of them is don't care X. While populating the set $T_{compatible}$, a given flip-flop is added into the set of first encountered *compatible* flip-flop. Then, a unique address (starting from 1[5]) is assigned to each of the *compatible set* T_{Ci}. Next, for each flip-flop in the set S_{RAS}, the *si selector module* is added into it's scan-input path with the *ADDR* input connected to the $(i-1)^{th}$ output line of the address decoder where i is the address assigned to the *compatible set* T_{Ci} to which the flip-flop belongs.

4.4 Test Generation and Application Methods

The test set for the proposed architecture T_{Hybrid} is the union of the test set for the *SSB* mode T_{SSB} and the test set for the *RASB* mode T_{RASB}. Thus,

$$T_{Hybrid} = T_{SSB} \bigcup T_{RASB} \tag{4.2}$$

Note that T_{SSB} is already available from the configuration steps described in Section 4.3. T_{RASB} is generated by translating the serial mode T_{Serial} test set (used during the configuration steps) into the *RASB* mode. The translation procedure is quite straight forward - we just need to Figure out which RAS broadcast groups need to be loaded and with what value

[5]address 0 is reserved and not used as a valid *RAS* address

Algorithm 4.1 Translate Single Scan Chain Patterns into RASB Mode Patterns (T_{Serial}, groups, N, M)

1: $PatternCount = N$
2: $GroupCount = M$
3: **for** $i = 0\,to\,N$ **do**
4: $PiValues \Leftarrow GetPiValues(T_{Serial}[i])$
5: $SpecifiedGroups \Leftarrow GetSpecGroups(T_{Serial}[i])$
6: $SpecifiedValues \Leftarrow GetSpecVals(T_{Serial}[i],\ SpecifiedGroups)$
7: $SpecifiedAddresses \Leftarrow GetAddresses(SpecifiedGroups)$
8: $T_{RASB}[i] \Leftarrow GenPat(PiValues,\ SpecifiedAddresses,\ SpecifiedValues)$
9: **end for**

Table 4.1: Functions of Algorithm 4.1

Function	**Description**
GetPiValues(*pattern*)	Extract and return primary inputs assignments in the *pattern*
GetSpecGroups(*pattern*)	Return the RASB groups specified in the *pattern*
GetSpecVals(*pattern, groups*)	Return the specified scan load values for *groups* in the *pattern*
GetAddresses(*groups*)	Return the RAS addresses for the *groups*
GenPat(*pis, addrs, vals*)	Generate pattern by concatenating given primary input values, RAS addresses and associated scan load values

and then use this information to write the pattern in the required format. The pseudo-code for this procedure is shown in the Algorithm 4.1

The test application procedure is also very simple - it consists of simply applying the *SSB* mode patterns followed by the *RASB* mode patterns. The *SSB* mode pattern application is exactly similar to the conventional serial scan and is shown in the pseudo code shown in Algorithm 4.2. In the pseudo code, *scan-enable* is the top-level control pin which selects between the scan-input and functional input of the scan flip-flop and enables the serial scan-chains operation. Also, it is possible to *shift-out* the MISR contents for comparison after each pattern rather than after all the patterns are applied.

The *RASB* mode pattern application makes use of the flip-flop addressability to set them to the required values and then uses the serial-scan shifting to observe the circuit response. The pseudo code for it is shown in the Algorithm 4.3

Algorithm 4.2 Test Application in SSB Mode

1: *PatternCount* = N
2: *set BM_SELECT = 0 to enable SSB mode*
3: *reset the MISR*
4: **for** $i = 0$ *to* N **do**
5: *assert scan enable*
6: *apply test clocks to load the scan chains (shift-in)*
7: *apply required primary input values*
8: *deassert scan enable*
9: *strobe primary outputs*
10: *apply test clock to capture the circuit response*
11: *assert scan enable*
12: *apply test clocks to unload the scan chains (shift-out)*
13: **end for**
14: *shift-out the MISR contents and compare with expected response*

Algorithm 4.3 Test Application in RASB Mode

1: *PatternCount* = N
2: *reset the MISR*
3: **for** $i = 0$ *to* N **do**
4: *assert BM_SELECT = 1 to enable RASB mode*
5: *assert scan_enable*
6: **for all** specified RAS groups **do**
7: *apply its address*
8: *apply its load value*
9: **end for**
10: *deassert BM_SELECT, scan_enable*
11: *apply required primary input values*
12: *strobe primary outputs*
13: *apply test clock to capture the circuit response*
14: *assert scan enable*
15: *apply test clocks to unload the scan chains (shift-out)*
16: **end for**
17: *shift-out the MISR contents and compare with expected response*

4.5 Experimental Setup

As described earlier, in our experiments, we use the broadcast mode of ILS as the *SSB* mode and the proposed *Dependency*-based heuristic to configure it. Similar to Section 3.5, we use six configurations $C1-C6$ based on the number of chains in the *SSB* mode and compare test data volume (TDV) and test application time (TAT) savings achieved using the proposed architecture against the conventional ILS architecture. We use a commercial ATPG tool to generate the patterns and all the experiments are performed on a 3GHz Linux machine with 8GB RAM. All the procedures to configure the architecture and to translate the patterns for *RASB* mode are written in C++. Similar to Section 3.5, we do not implement any output compactor circuit.

The test data volume and test application time for the *SSB* mode are calculated using the equations 3.5 and 3.6. For *RASB* mode, these are calculated using the following equations:

$$RASB\,Mode\,TDV\,(bits) = \sum G \times (ADDR + PI + PO + 2) \quad (4.3)$$

$$RASB\,Mode\,TAT\,(cycles) = \sum (G + 1 + LSCL) \quad (4.4)$$

In the above equations, the summation is over all the *RASB* mode patterns, G is the number of RAS broadcast groups that need to be loaded for a given pattern, $ADDR$ is the number of input address lines required to decode the addresses, PI is the number of primary inputs, PO is the number of primary outputs and $LSCL$ is the longest scan chain length in the *SSB* mode. Note that, in general, $LSCL$ is much small compared to the total numnber of flip-flops in the design. Hence, it follows that the test application time requirement for a single *RASB* mode pattern in the hybrid architecture will always be much lower compared to that of a single *Serial* mode pattern in the conventional ILS. For example, consider a circuit with 100 flip-flops. Assume that there are 10 chains in the *SSB* mode with each chain having 10 flip-flops. Also, assume that there are 20 broadcast groups in the *RASB* mode. Then the worst case test application time requirement for a *RASB* mode pattern will be

31 while that of a *Serial* mode pattern in the conventional ILS will be 101! In equation 4.3, the number 2 is to account for the top-level *BM_SELECT* and *RASB scan-input* pins and in equation 4.4, the number 1 is to account for the scan-clock pulsing during the circuit response capture procedure. Again, similar to Section 3.5, we ignore the test data volume and test application time for the *MISR*.

4.6 Results

We will evaluate the proposed Hybrid Architecture based on the following parameters:

- Area overhead incurred
- Additional reduction achieved in test data volume
- Additional reduction achieved in test application time

Plots in Figures 4.3 - 4.8 show the % reduction achieved for all the benchmark circuits considered for all six configurations for the conventional ILS and proposed Hybrid Architecture with respect to the single scan chain. Plots in Figures 4.9 - 4.11 compare the performance of the Hybrid Architecture relative to the conventional ILS.

Table 4.2 compares the average performance of the proposed hybrid architecture for six different configurations $C1 - C6$ similar to Section 3.5. In the table, column

- *conf* stands for the configuration
- *CITDVR* and *CITATR* stands for % Reduction in Test Data Volume and Test Application Time using Conventional ILS
- *HATDVR* and *HATATR* stands for % Reduction in Test Data Volume and Test Application Time using Hybrid Architecture
- *AR* for Additional Reduction achieved
- *HAAO* for Area Overhead of Hybrid Architecture

Columns 2 to 7 compare the average performance of the regular Illinois Scan Architecture and Hybrid Architecture relative to the single serial scan chain. We can observe that the proposed Hybrid Architecture helps to achieve on average 9% more reduction in test data volume and 33.67% more reduction in test application than the conventional ILS Architecture. These observed results are in accordance with the expected results. Columns 8 to 10 compare the area overhead, test data volume reduction and test application time reduction of the Hybrid Architecture relative to the conventional Illinois Scan Architecture. These columns are indicator of reduction achievable in test data volume and test application time required for *serial mode* in conventional ILS and the area overhead required for that. From these columns we can observe that as the number of chains increases in $C1\,to\,C6$, the area overhead increases and the achievable test data volume reduction decreases but the achievable test application time reduction increases! This can be explained in the following way: as the number of chains in the *SSB* mode increases, the fault coverage achievable in it decreases (see Section 3.6). This drop in the *SSB* mode fault coverage requires the *RASB* mode to cover increased number of faults which in turn translates to more number of flip-flops that need to be RAS enabled. This increases the area overhead due to the *si selector module*. Also, these more number of RAS enabled flip-flops may result into more number of *RASB* groups which will call for bigger address decoder and thus, more area overhead. Also, the number of patterns required in the *RASB mode* may be more and in addition, each of these patterns may require more number of *RASB* groups to be specified. This coupled with the fact that the bigger address decoder will require more number of input address bits will result into more test data volume for the *RASB* mode. All these factors combined will lead to more test data volume overall and hence, lesser reduction in test data volume. Next, for each *RASB* mode pattern, we need to specify all the primary input and output values in addition to the addresses for the specified groups and their load values. But to apply all of this test data, the number of test cycles required is just the summation of the number of RAS groups that need to be loaded and the longest scan chain length. Thus, the test application time requirement for a *RASB* pattern will always be much smaller compared to

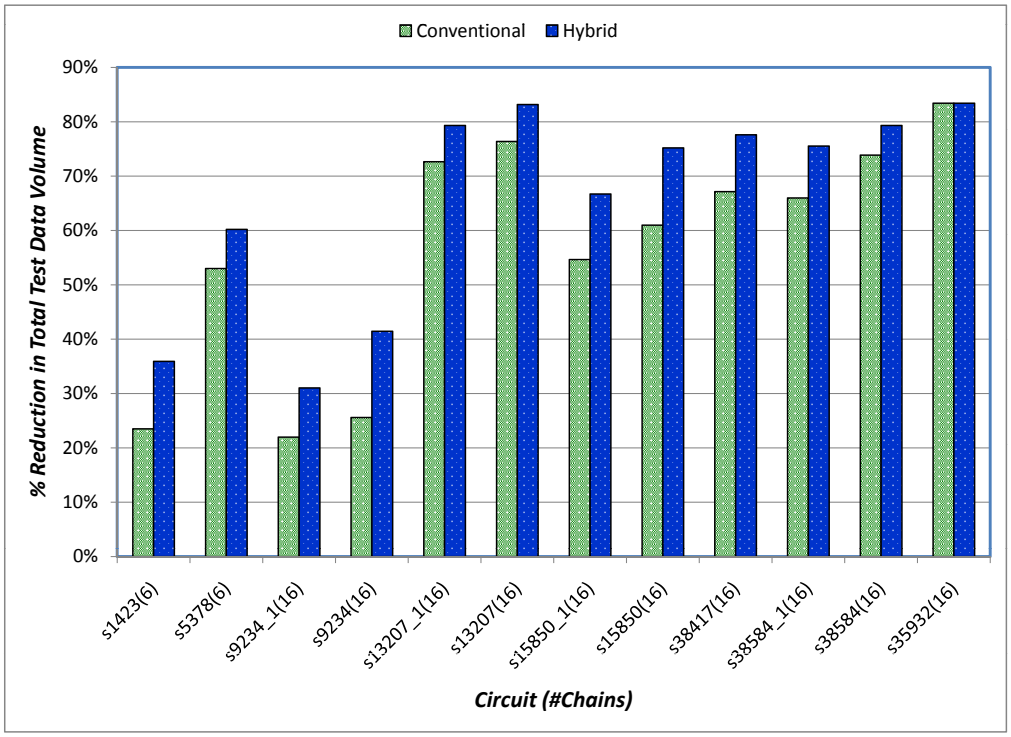
the test data volume requirement. Hence, the factors like increased number of *RASB* mode patterns, increased number of specified *RASB* groups per pattern or increased number of input address lines for the address decoder will result in relatively very little increase in the *RASB* mode test application time and hence, we will have more reduction in the test application time overall. This explains, why, in the Hybrid Architecture, the reduction achieved in test application time is always higher than the reduction achieved in test data volume. This same reasoning also explains why in some cases Hybrid Architecture has more test data volume than the conventional ILS.

From the plots shown in Figures 4.3 - 4.8 and 4.9 - 4.11, we can also observe that the proposed Hybrid Architecture achieves the best test data volume and test application time reduction with the least area overhead for the bigger circuits. For example, results for the bigger circuits *s38417, s38584 and s38584_1* for configuration *C1* indicate that, with just 1.69% more area, we can achieve 27% and 37% additional reduction in the test data volume and test application time relative to the *effectively configured* conventional ILS. The corresponding numbers for the remaining circuits are 16% more area, 23% and 48% additional reduction in test data volume and test application time respectively. Thus, for the smaller circuits, the area overhead incurred is quite significant and the additional reduction in test data volume is less (sometimes even negative) as the number of chains increase from configuration $C1\,to\,C6$. This is so, because, for these smaller circuits, fault coverage in the *SSB* mode drops quite significantly (due to the high number of ATPG untestable faults) when compared with the bigger circuits. Test patterns required to cover these faults have significantly higher number of unique flip-flops specified (across the test-set). This leads to more number of RAS enabled flops and more number of *RASB* groups as fewer flip-flops are test-wise compatible. Thus, we may end up with the *RASB* mode patterns in which almost all of the RAS groups are specified in each pattern. This leads to more area overhead and higher *RASB* mode test data volume as explained in the preceding paragraph.

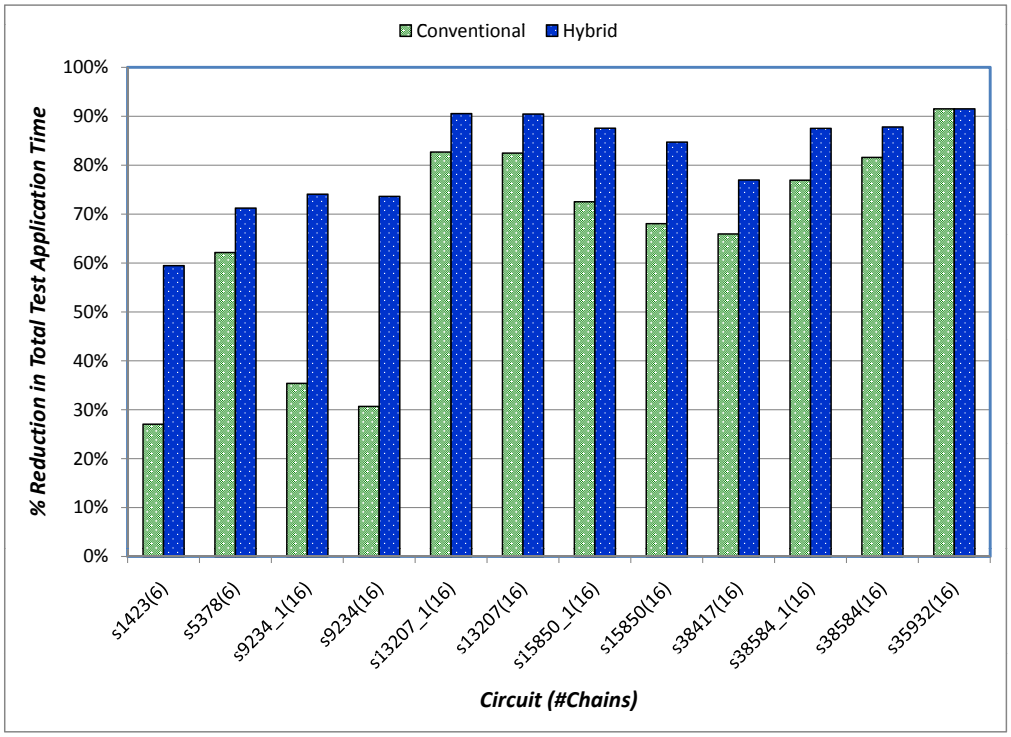
Table 4.2: Relative Performance Comparison

	% Reduction Relative to Single Scan Chain						Relative to Conventional ILS		
Conf	CITDVR	HATDVR	AR	CITATR	HATATR	AR	HAAO	HATDVR	HATATR
C1	56.60%	65.74%	9.15%	64.75%	81.29%	16.55%	12.26%	23.77%	45.49%
C2	46.14%	56.98%	10.84%	57.55%	84.49%	26.95%	19.09%	24.32%	63.05%
C3	41.46%	48.18%	6.72%	53.36%	84.63%	31.27%	22.86%	17.99%	69.52%
C4	38.30%	44.80%	6.50%	50.24%	84.80%	34.57%	24.83%	15.25%	71.71%
C5	28.91%	41.55%	12.65%	40.10%	85.07%	44.97%	26.07%	20.91%	76.65%
C6	25.53%	35.37%	9.84%	36.61%	84.33%	47.71%	27.30%	16.59%	76.81%

conf = configuration; CITDVR = Conventional ILS %TDV Reduction; HATDVR = Hybrid Architecture % TDV Reduction;
CITATR = Conventional ILS % TAT Reduction; HAAO = Hybrid Architecture Area Overhead; AR = Additional Reduction

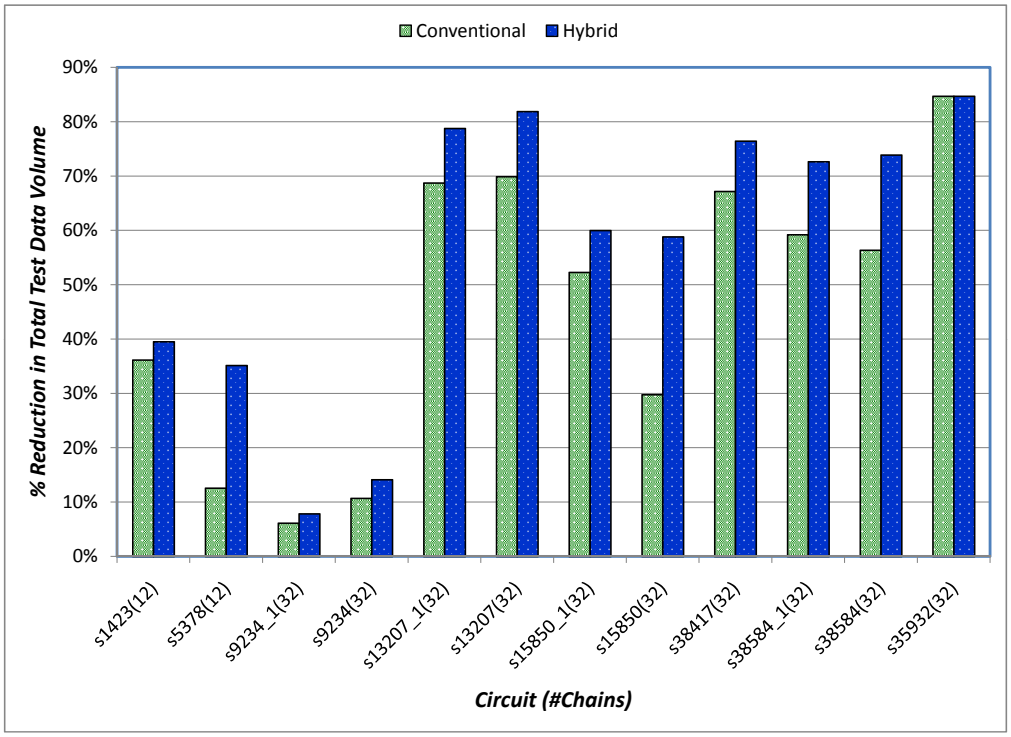


(a) TDV Reduction for *C1*

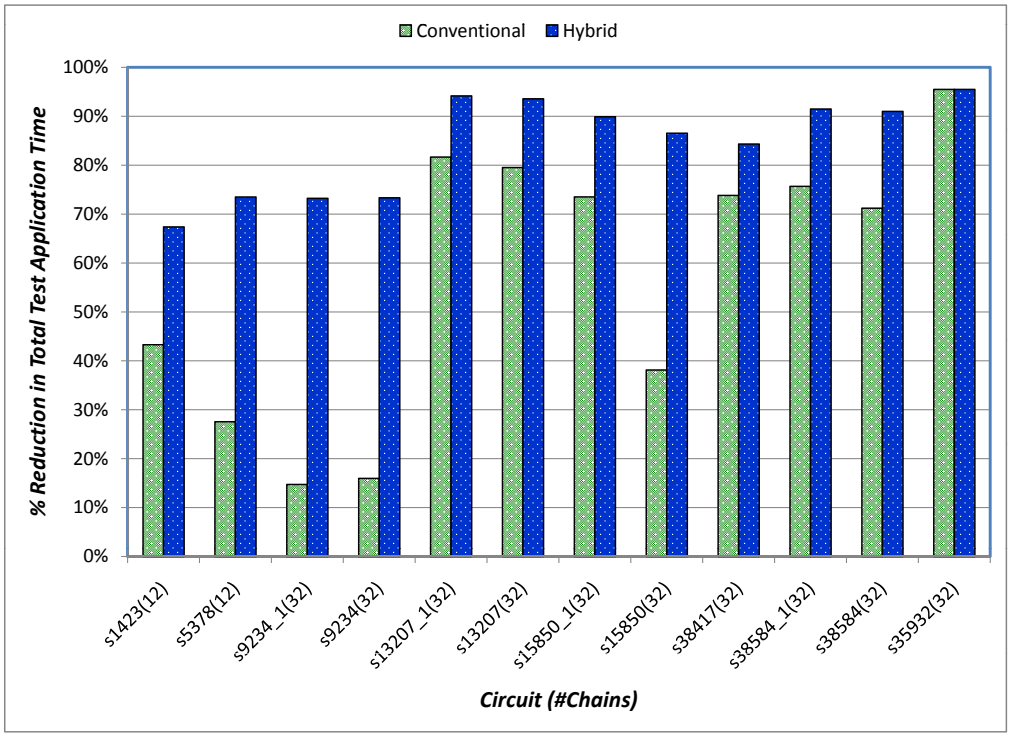


(b) TAT Reduction for *C1*

Figure 4.3: Comparison of reduction in TDV and TAT for configuration $C1$

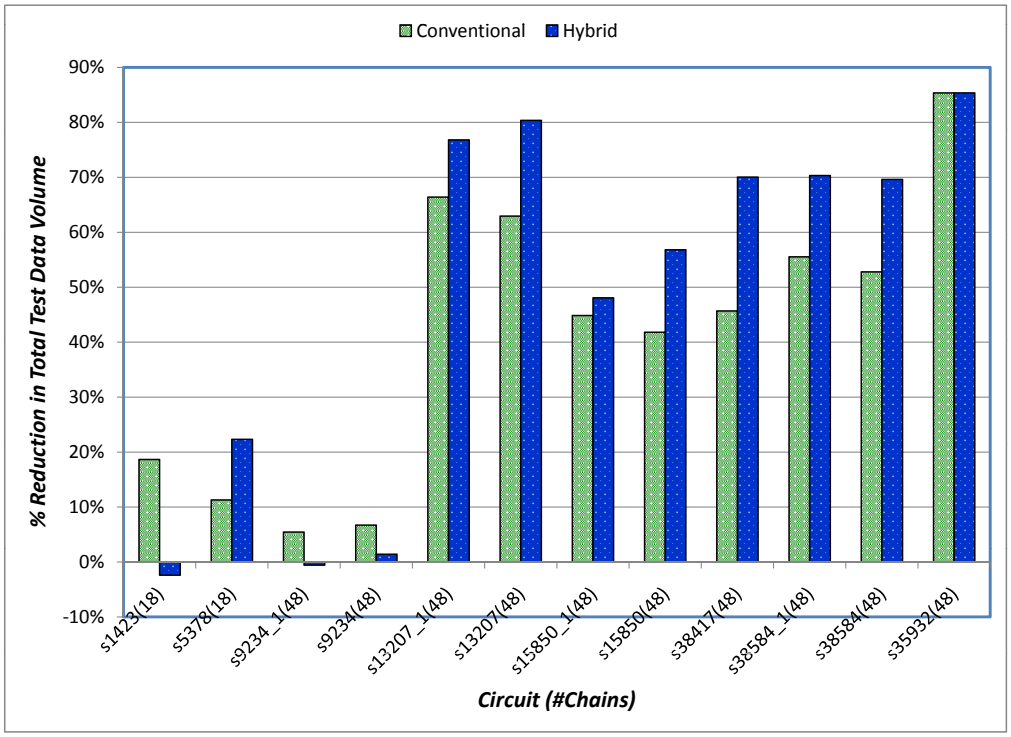


(a) TDV Reduction for *C2*

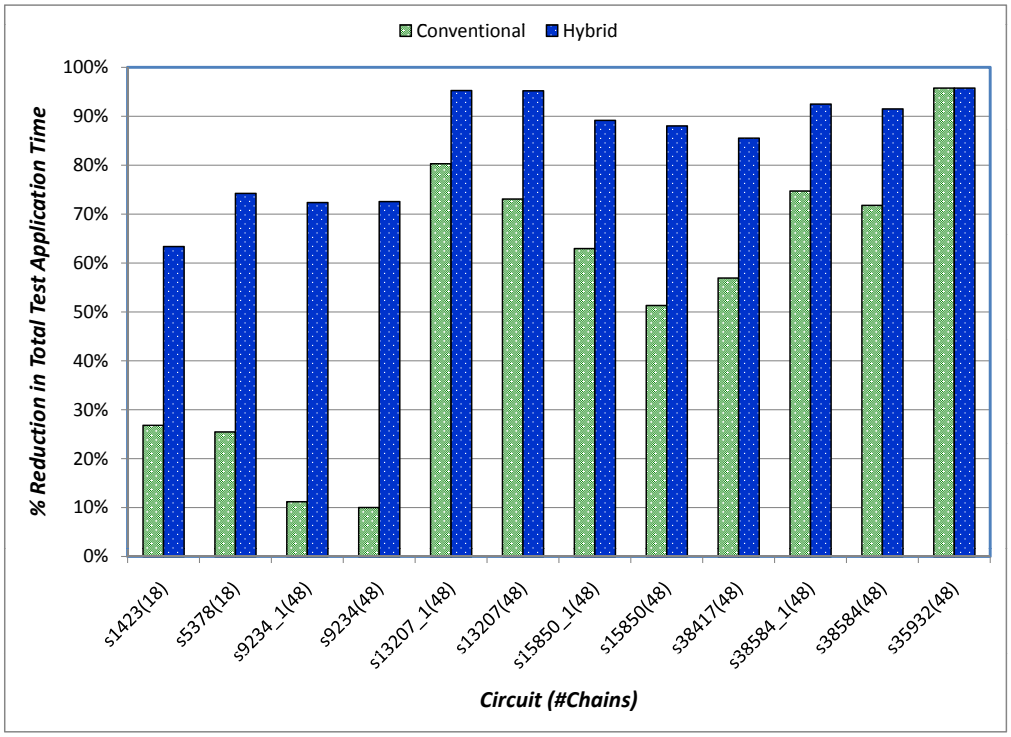


(b) TAT Reduction for *C2*

Figure 4.4: Comparison of reduction in TDV and TAT for configuration $C2$

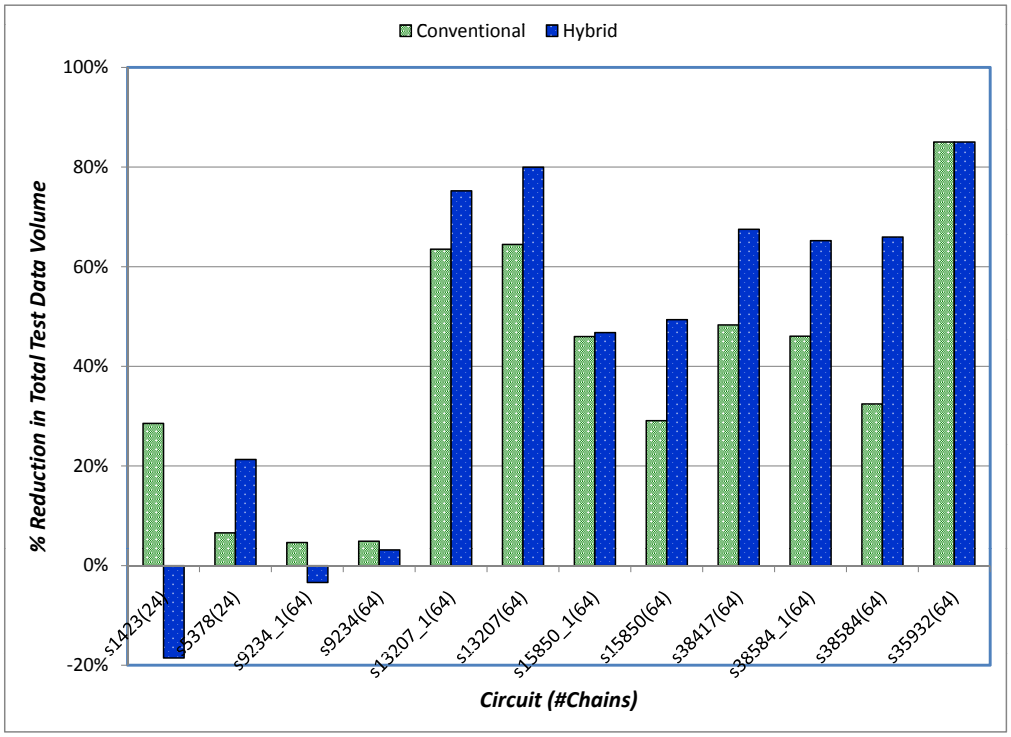


(a) TDV Reduction for *C3*

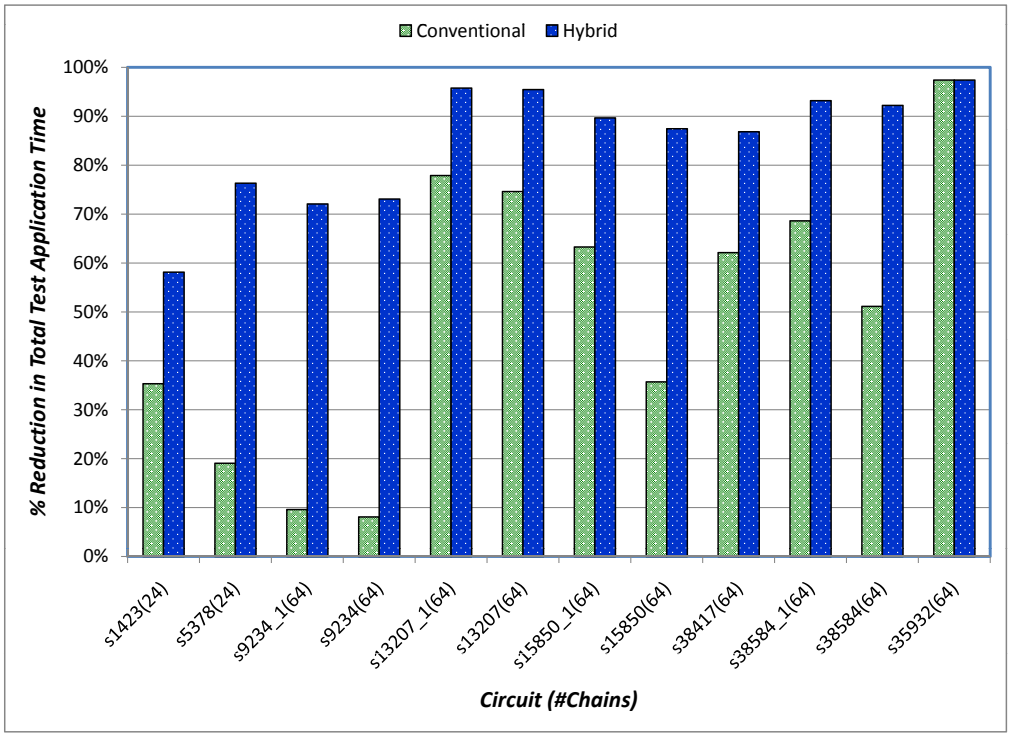


(b) TAT Reduction for *C3*

Figure 4.5: Comparison of reduction in TDV and TAT for configuration $C3$

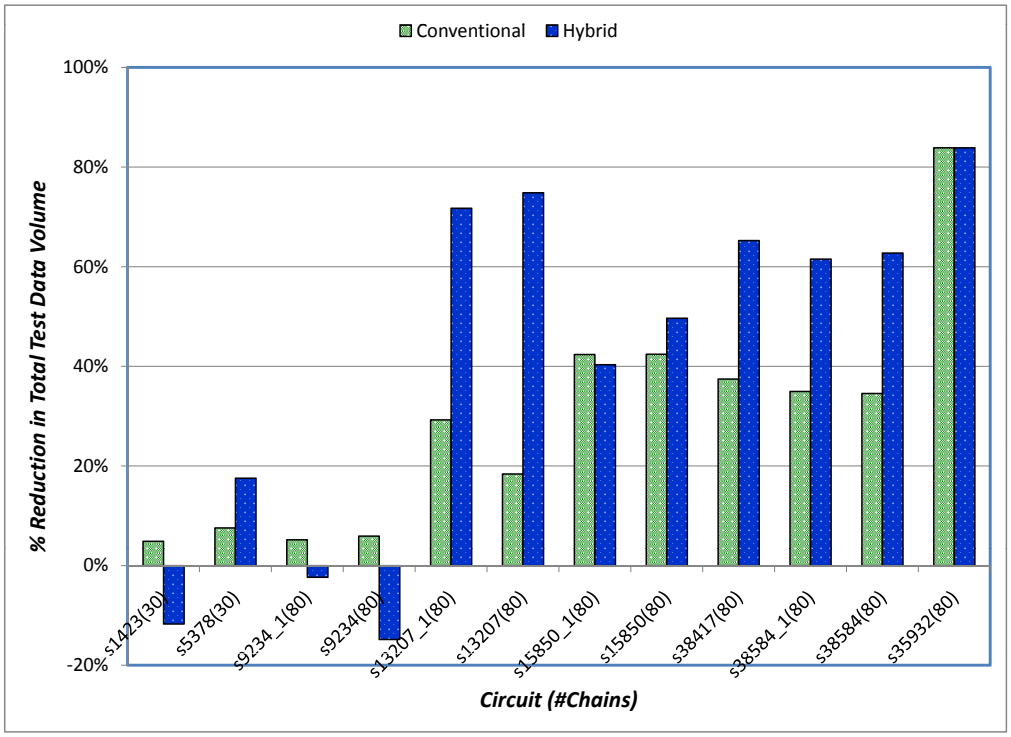


(a) TDV Reduction for *C4*

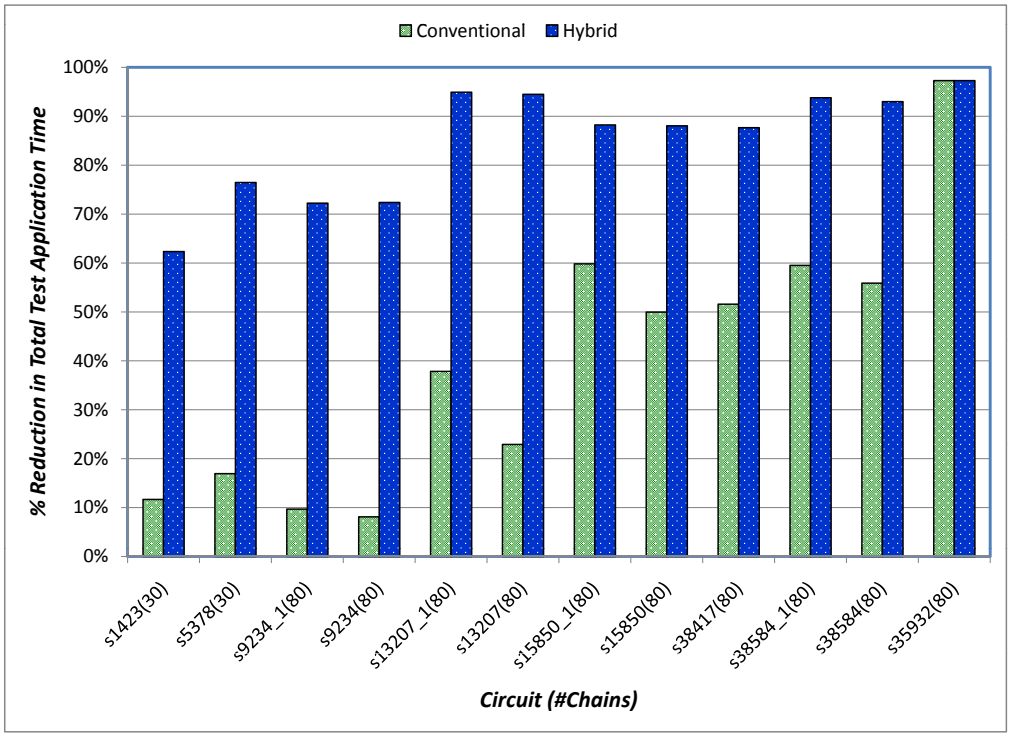


(b) TAT Reduction for *C4*

Figure 4.6: Comparison of reduction in TDV and TAT for configuration $C4$

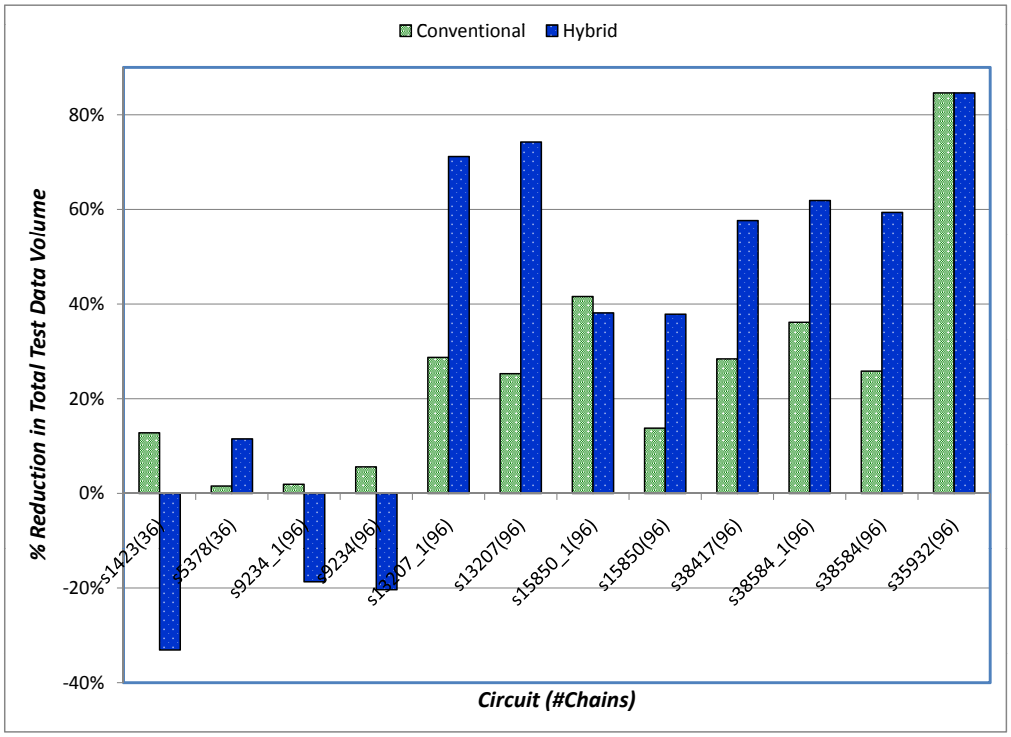


(a) TDV Reduction for *C5*

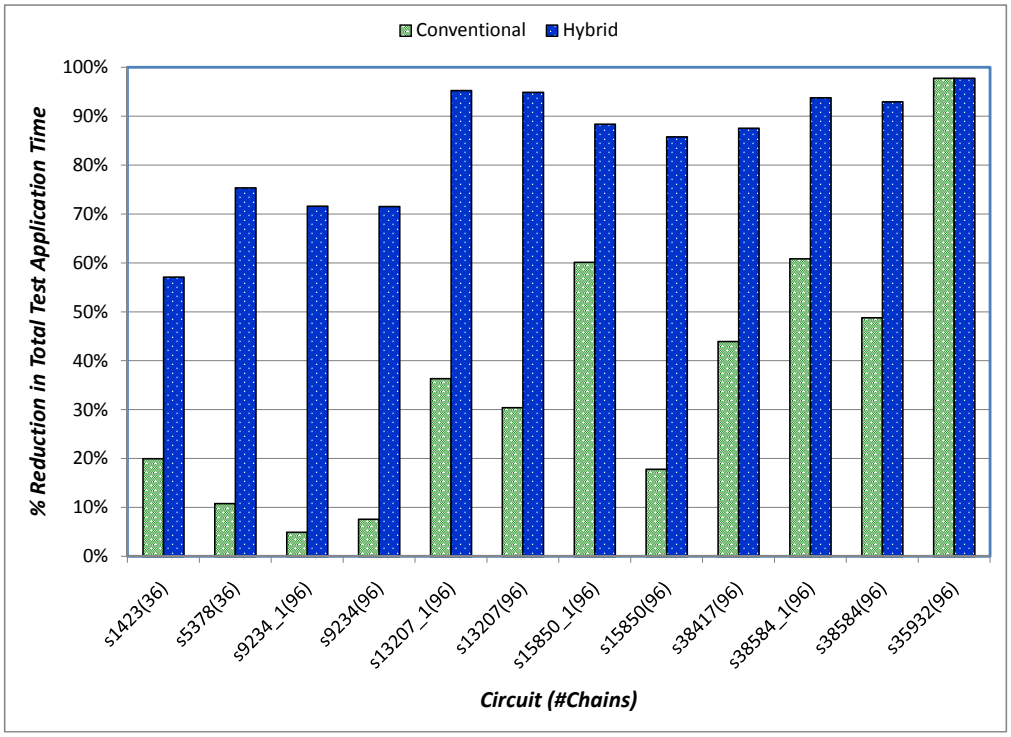


(b) TAT Reduction for *C5*

Figure 4.7: Comparison of reduction in TDV and TAT for configurations $C5$

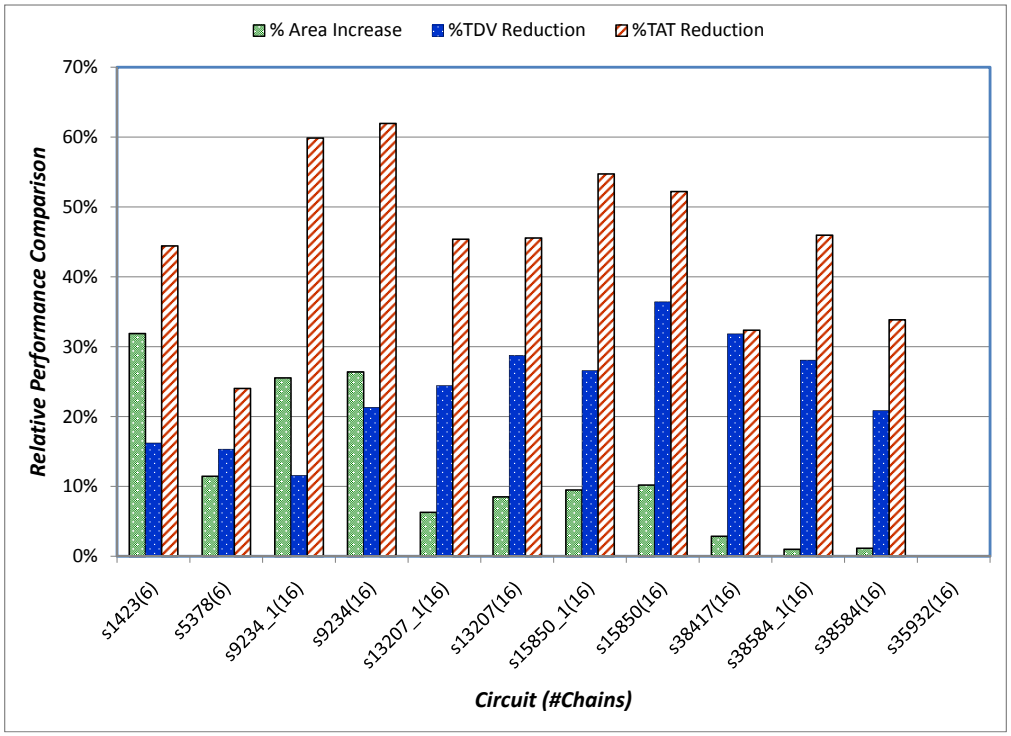


(a) TDV Reduction for *C6*

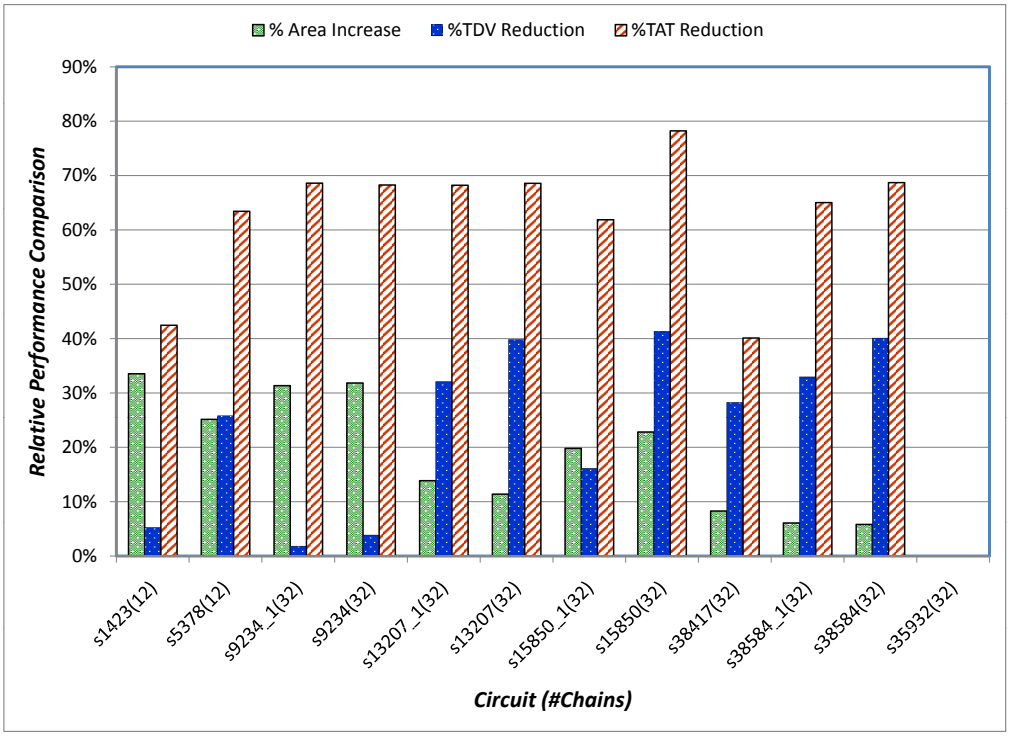


(b) TAT Reduction for *C6*

Figure 4.8: Comparison of reduction in TDV and TAT for configuration $C6$

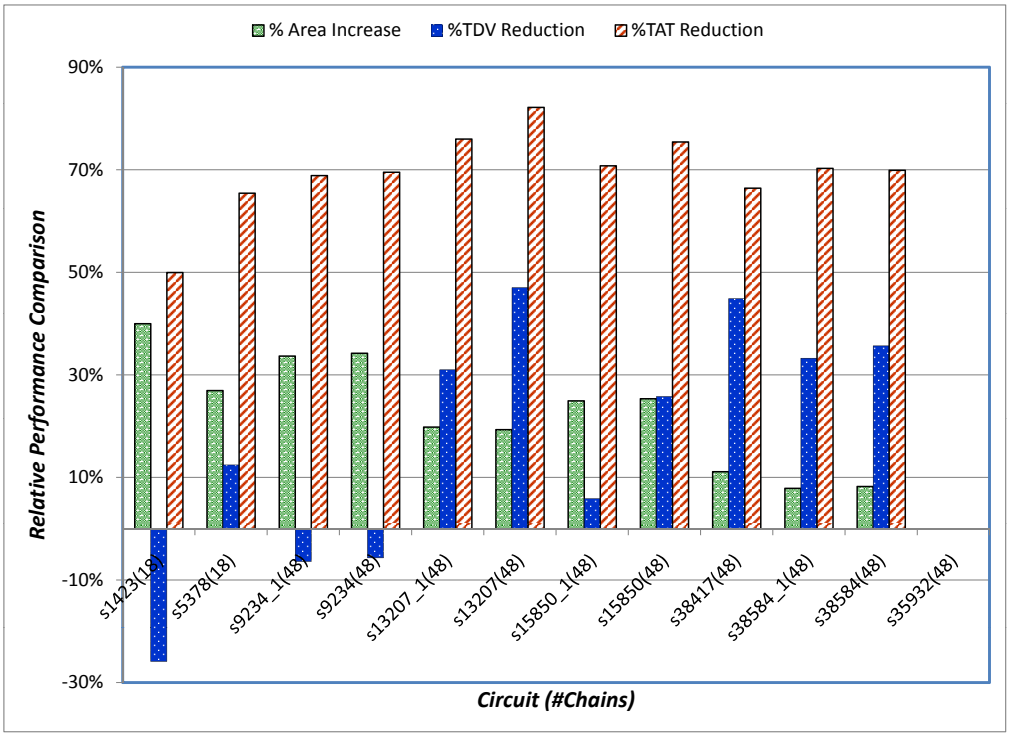


(a) Configuration *C1*

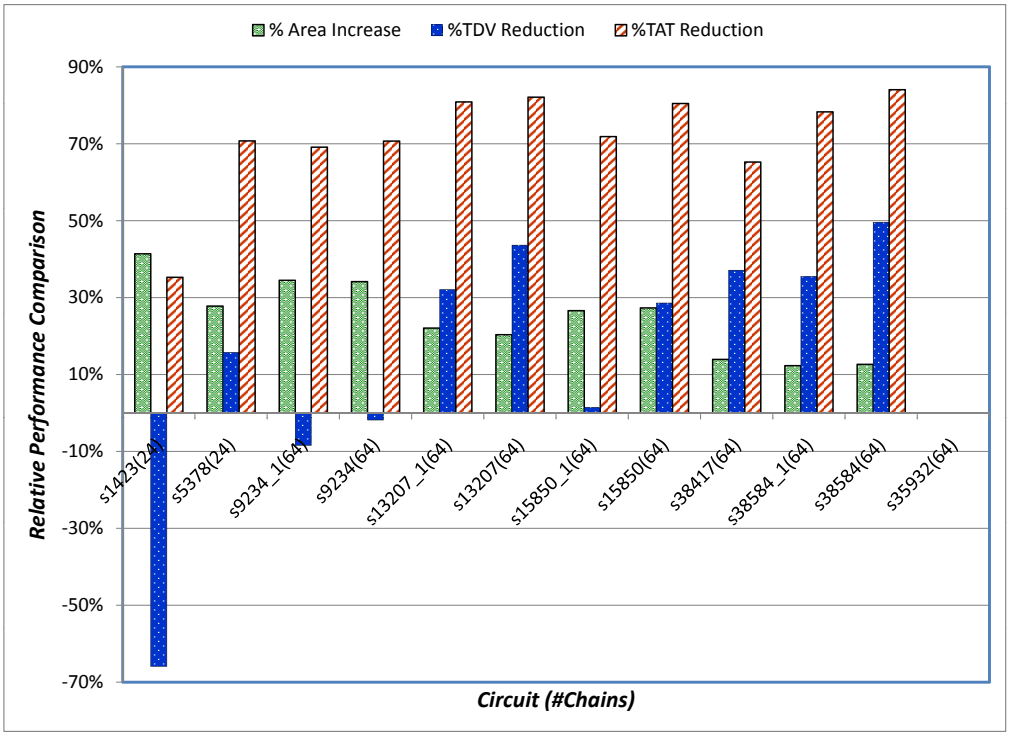


(b) Configuration *C2*

Figure 4.9: Comparison of Hybrid Arch. relative to Conventional ILS Arch for conf. $C1, C2$

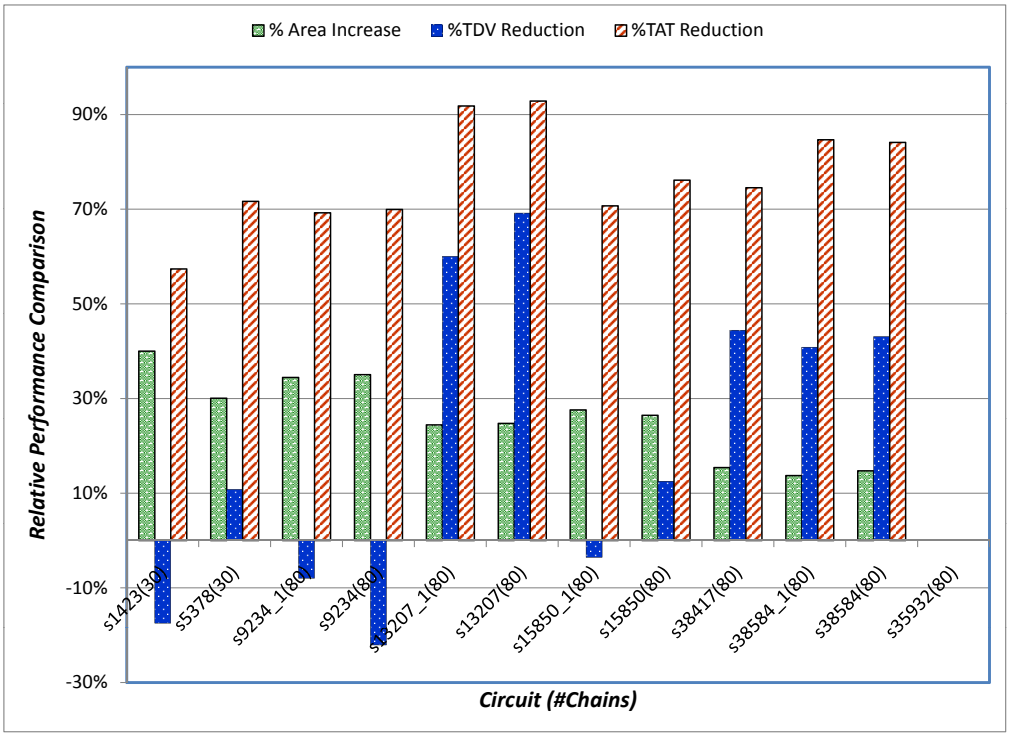


(a) Configuration *C3*

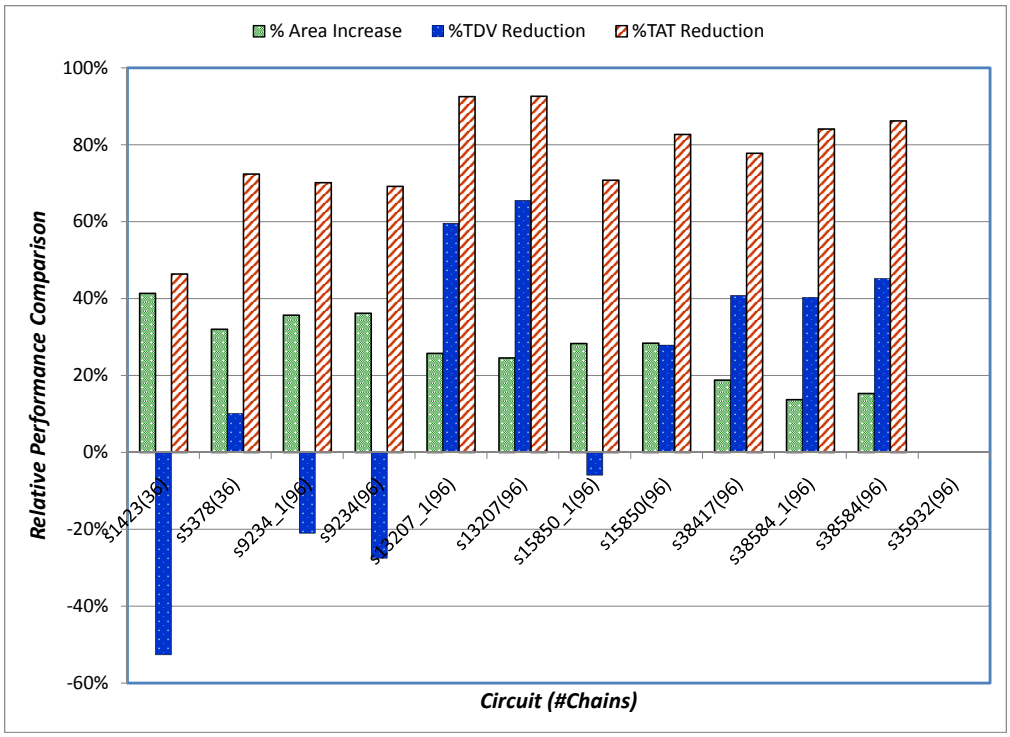


(b) Configuration *C4*

Figure 4.10: Comparison of Hybrid Arch. relative to Conventional ILS Arch. for conf. *C3*, *C4*



(a) Configuration *C5*



(b) Configuration *C6*

Figure 4.11: Comparison of Hybrid Arch. relative to Conventional ILS Arch. for conf. *C*5, *C*6

4.7 Summary

In this chapter we proposed a new scan architecture based on the dual broadcast mode strategy. It uses the serial scan broadcast mode as the primary test mode and RAS broadcast mode as the supplementary mode. The RAS broadcast mode reuses the scan chains and the compaction logic of the *SSB* mode for test response observation. The experimental results show that the proposed architecture is best suited for the bigger circuits which is important from the real designs point of view. Currently we use simple techniques to generate the *RASB* mode patterns from the conventional single scan chain based patterns. Since the *RASB* mode configuration is test set based, it follows that utilizing a specialized, tailor-made ATPG engine to generate patterns for the *RASB* mode will further improve the performance of the *RASB* mode. This, in turn, will boost the overall test data volume and test application time reduction achievable with the proposed Hybrid Architecture with reduced area overhead.

Chapter 5

Conclusions and Future Work

In this thesis, we have tackled one of the most pressing needs of the semiconductor industry today - reduction of the test data volume and test application time to enable cost-effective quality structural testing. The current industry standard serial scan based DFT technique suffers from high test data volume and longer test application time requirement as the circuit size increases. The need to test the chips using advanced fault models aggravates the problem. We introduced this problem of VLSI testing in *Chapter 1.* In *Chapter 2,* we outlined various DFT techniques used in the industry for structural testing, explained the need for the *compression schemes* and elaborated on one such scheme - the Illinois Scan Architecture (ILS). We described the need for *effective configuration* of ILS and gave overview of the existing methods.

In *Chapter 3*, we proposed a novel circuit topology based framework to *effectively configure* the ILS and presented 3 *low cost heuristics* based on it. The framework uses a fan-in/out cone relation matrix (FCM) for flip-flops and generates groups of flip-flops which can be placed in the same column in the ILS. The experimental results on the *ISCAS'89* benchmark circuits show that the proposed approach improves the achievable broadcast mode fault coverage by an average *5%* and achieves *15%* more reduction in test data volume and test application time when compared to the existing test-set based *effective* method.

In *Chapter 4*, we proposed a new hybrid scan architecture that combines the broadcast

mode of ILS and Random Access Scan architecture to eliminate the *serial mode* of the ILS. We described the motivation, configuration method and test generation and application scheme for the proposed architecture. The experimental results on the *ISCAS'89* benchmark circuits show that the proposed architecture is especially effective for the bigger circuits. When compared to the *effectively configured ILS* using our proposed heuristic, for the best case results, for circuits with more than 1000 flip-flops, we get 27% and 37% more reduction in the test data volume and test application time respectively for an area overhead of just 1.69%.

The proposed scan chain partitioning framework can be enhanced by considering some kind of routing measures so that the routing overhead can be minimized. Also, the Fanout Estimate based heuristic and SCOAP based heuristic can be improved by accounting for the reconvergent fanout cones. Another important factor for structural testing is limiting the power dissipation during the test mode. Current methods for this are minimizing the toggle activity and/or shutting-off some of the chains during the scan-chain shifting process. The proposed framework can be enhanced to consider these features also.

The proposed Hybrid ILS-RAS architecture has a lot of avenues for improvement, particularly in the *RASB* mode configuration method. As discussed in the preceding *Chapter 4*, to maximize the gains of the hybrid architecture, minimizing the number of RAS-enabled flip-flops (N_{RAS}) is crucial. Currently, we use a test-set generated using a single scan chain to compute this set of RAS-enabled flip-flops. The conventional single scan-chain ATPG treats all the primary inputs (PIs) and pseudo primary inputs (PPIs) equally. But to reduce N_{RAS}, we need an ATPG engine that gives priority to PIs over PPIs and to an already specified PPI over an unspecified PPI. Another way can be to replace the test-set based approach by a circuit topology based approach that can consider the existing ILS groups and can come up with a set of flip-flops which when made RAS-enabled can guarantee no degradation in the achievable fault coverage.

Bibliography

[1] G. Moore, "Cramming more components onto integrated circuits", in *Electronics*, 38(8), 1965, pp. 114-117.

[2] F. Hsu, K. M. Butler and J. H. Patel, "Case study on the implementation of the Illinois Scan Architecture",in *Proc. IEEE International Test Conference*, Nov. 2001, pp. 538-547.

[3] I. Hamzaoglu and J. H. Patel, "Reducing test application time for full scan embedded cores", in *Digest of papers, 29th International Symposium on Fault-Tolerant Computing*, June 1999, pp. 260-267.

[4] K.-J. Lee, J. J. Chen, and C. H. Huang, "Broadcasting test patterns to multiple circuits", in *IEEE Transactions on Computer-Aided Design*, 18(12), December 1999, pp. 1793–1802.

[5] K. Lee , J-J. Chen and C-H. Huang, "Using a single input to support multiple scan chains", *Digest of Technical Papers, 1998 IEEE/ACM International Conference on Computer-Aided Design,* Nov. 1998, pp. 74-78.

[6] C. Barnhart et.al., "OPMISR:Foundation for Compressed ATPG Vectors", *Proc. IEEE International Test C*onference , Nov. 2001, pp. 748-757.

[7] C. Barnhart et.al., "Extending OPMISR beyond 10X scan test efficiency", *Design and Test of Computers,* Oct. 2002, pp. 65-73.

[8] M. Ashouei, A. Chatterjee, A. Singh, "Test volume reduction via flip-flop compatibility analysis for balanced parallel scan", *Proc. 2004 Current and Defect Based Testing*, April 2004, pp.105-109.

[9] A. R. Pandey and J. H. Patel, "Reconfiguration Technique for Reducing Test Time and Test Data Volume in Illinois Scan Architecture Based Designs", *Proc. VLSI Test Symposium,* May 2002, pp. 9-15.

[10] N. Oh, R. Kapur, T. Williams and J. Sproch, "Test pattern compression using prelude vectors in fan-out scan chain with feedback architecture", *Proc. Design, Automation and Test in Europe,* May 2003, pp. 110-115.

[11] M. A. Shah and J. H. Patel, "Enhancement of the Illinois scan architecture for use with multiple scan inputs", in *Proc. VLSI Test Symposium* 2004, pp. 167-172.

[12] D. Pradhan and J. Saxena, "A design for testability scheme to reduce test application time in full scan", in *Proc. VLSI Test Symposium,* April 1992, pp. 55-60.

[13] S. Lee and K. G. Shin, "Design for test using partial parallel scan", in *IEEE Trans. on Computer Aided- D*esign, vol. 9, February 1990, pp. 203-211.

[14] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", in *Proc. International Symposium on Circuits and Systems,* May 1989, pp. 1929-1934.

[15] F. Corno, M. Sonza Reorda and G. Squillero, "RT-Level ITC 99 Benchmarks and First ATPG Results", in *IEEE Design and Test of Computers*, July-August 2000, pp. 44-53.

[16] S. Mitra and K. S. Kim, "X-Compact: an efficient response compaction technique for test cost reduction", in *Proc. International Test Conference,* Oct. 2002, pp. 311-320.

[17] L. Wang, C. W. Wu and X. Wen, *VLSI Test Principles and Architectures*, San Fransisco: Morgan Kaufmann Publishers, 2006.

[18] B. Arslan, O. Sinanoglu, A. Orailoglu, "Extending the applicability of parallel-serial scan designs", in *Proc. International Conference on Computer D*esign, Oct. 2004, pp. 200-203.

[19] S. Narayanan and M. A. Breuer, "Reconfiguration techniques for a single scan chain",in *IEEE Trans. Computer-Aided D*esign, vol. 14, no. 6, June 1995, pp. 750-765.

[20] L. R. Harriott, "A new role for e-beam: Electron projection", in *IEEE Spectrum*, vol. 36, no. 7, July 1999, pp. 41-45.

[21] A. Jas, B. Pouya, and N. A. Touba, "Virtual scan chain: Means for reducing scan length in cores", in *Proc. VLSI Test S*ymposium, April 2000, pp. 73-78.

[22] M. Sharma, J. H. Patel and J. Rearick, "Test Data Compression and Test Time Reduction of Longest-Path-Per-Gate Tests based on Illinois Scan Architecture", in *Proc. VLSI Test Symposium,* May 2003, pp. 15-21.

[23] A. Pandey and J.H. Patel, "Incremental algorithm for test generation in Illinois Scan Architecture based designs", in *Proc. Design, Automation and Test in Europe,* March 2002, pp. 369-375.

[24] K. Cho, S. Mitra, E. J. McCluskey, "California scan architecture for high quality and low power test", in *Proc. International Test C*onference, Nov. 2007, pp. 1-10.

[25] A. Chandra, F. Ng and R. Kapur, "Low Power Illinois Scan Architecture for Simultaneous Power and Test Data Volume Reduction",in Proc. *Design, Automation and Test in Europe,* March 2008, pp. 462-467.

[26] N. A. Touba, "Survey of test vector compression techniques", in *IEEE Design and Test of Computers*, vol. 23, July-Aug. 2006, pp. 294-303.

[27] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur and T. W. Williams, "A reconfigurable shared scan-in architecture", in *Proc. VLSI Test Symposium,* May 2003, pp. 9-14.

[28] A. Chandra, Y. Haihua and R. Kapur, "Multimode Illinois Scan Architecture for Test Application Time and Test Data Volume Reduction", in *Proc. VLSI Test S*ymposium, May 2007, pp. 84-92.

[29] L. Wang, W. Xiaoqing, H. Furukawa, F. S. Hsu, S. H. Lin, S. W. Tsai, K. S. Abdel-Hafez, W. Shianling Wu, "VirtualScan: A new compressed scan technology for test cost reduction", in *Proc. International Test Conference,* Nov. 2004, pp. 916-925.

[30] L. H. Goldstein, "Controllability/observability analysis of digital circuits", in *IEEE Trans. Circuits Systems, CAS-26(9),* 1979, pp. 685-693.

[31] L. H. Goldstein and E. L. Thigpen. "SCOAP: Sandia Controllability/Observability Analysis Program", in *Proc. Design Automation Conference*, June 1980, pp. 190–196.

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

[33] D. Baik, S. Kajihara and K. K. Saluja, "Random access scan: A solution to test power, test data volume and test time", in *Proc. International Conference on VLSI Design*, January 2004, pp. 883-888.

[34] D. Baik and K. K. Saluja, "Progressive Random Access Scan: A simultaneous solution to test power, test data volume and test time", in *Proc. International Test Conference*, November 2005, Paper 15.2 (10 pp.).

[35] A. S. Mudlapur, V. D. Agarwal and A. D. Singh, "A random access scan architecture to reduce hardware overhead", in *Proc. International Test Conference*, November 2005, Paper 15.1 (9 pp.).

[36] D. Baik and K. K. Saluja, "Test cost reduction using partitioned grid random access scan", in *Proc. International Conference on VLSI Design*, January 2006.

[37] S. P. Lin, C. L. Lee and J. E. Chen, "A cocktail approach on random access scan toward low power and high efficiency test", in *Proc. International Conference on Computer Aided Design*, November 2005, pp. 94-99.

[38] Y. Hu, C. Li, J. Li, Y-H. Han, X.-W. Li, W. Wang, H.-W. Li, L.-T. Wang, X.-Q. Wen, "Test Data Compression Based on Clustered Random Access Scan", in *Proc. Asian Test Symposium*, November 2006, pp. 231-236.

[39] Y. Hu, X. Fu, X. Fan, H. Fujiwara, "Localized random access scan: Towards low area and routing overhead", in *Proc. Asia and South Pacific Design Automation Conference*, March 2008, pp. 565-570.

[40] Y. Hu, Y.-H. Han, X.-W. Li, X.-Q. Wen, "Compression/scan co-design for reducing test data volume, scan-in power dissipation and test application time", in *Proc. Pacific Rim International Symposium on Dependable Computing*, December 2005.

[41] S.-J. Wang, K.S.-M. Li, S.-C. Chen, H.-Y. Shiu, "Scan-Chain Partition for High Test-Data Compressibility and Low Shift Power Under Routing Constraint", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 28(5), May 2009, pp. 716-727.

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

[43] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded Deterministic Test," in *IEEE Transactions on Computer Aided Design*, vol. 23, May 2004, pp. 776–792 .

[44] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, Revised Printing, Piscataway, NJ, 1994.

[45] S. M. Reddy, K. Miyase, S. Kajihara and I. Pomeranz, "On test data volume reduction for multiple scan chain designs", in *Proc. VLSI Test Symposium,* April 2002, pp. 103-108.

[46] A. Jas, J. Ghosh-Dastidar, M. Ng and N. A. Touba, "An efficient test vector compression scheme using selective Huffman coding", in *Proc. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 22(6), 797-806, 2003.

[47] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based design," in *Proc. International Test Conference*, Nov. 1998, pp. 458-464.

[48] A. Chandra and K. Chakrabarty, "System-on-a-Chip Test Data Compression and Decompression Architectures Based on Golomb Codes", in I*EEE Transactions Computer-Aided Design*, vol. 20, no. 3, Mar. 2001, pp. 355-368.

[49] I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction through Scan Chain Concealment", *in Proc. Design Automation Conference*, ACM Press, New York, 2001, pp. 151-161.

[50] G. Mrugalski, J. Rajski, and J. Tyszer, "Ring generators: New devices for embedded test applications", *in IEEE Transactions on Computer-Aided Design*, 23(9), September 2004, pp. 1306–1320.

[51] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding", in *Proc. Asian Test Symposium*, November 2001, pp. 325–330.

[52] C. V. Krishna, A. Jas, and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression", in *Proc. Int. Test Conference*, October 2002, pp. 321–330.

[53] N. Sitchinava, S. Samaranayake, R. Kapur, E. Gizdarski, F. Neuveux, and T. W. Williams, "Changing the scan enable during shift", in *Proc. VLSI Test Symposium*, April 2004, pp. 73–78.