Effective Techniques for Post-silicon Validation and Debug

A THESIS SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

by

Binod Kumar

(Roll No. 143079023)

Under the guidance of

Prof. Virendra Singh

Department of Electrical Engineering

Indian Institute of Technology Bombay

Mumbai 400076, India

January 2020

Thesis Approval for Ph.D.

The thesis entitled **Effective Techniques for Post-silicon Validation and Debug** is approved for the degree of **Doctor of Philosophy**.

External Examiner

Prof. Vineet Sahula

- - - - - - - - - - -

Internal Examiner

Prof. Sachin Patkar

- - - - - - - - - - -

Supervisor

Prof. Virendra Singh

- - - - - - - - - - -

Chairman

Prof. Vinish Kathuria

- - - - - - - - - - -

Date:

Place:

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

Binod Kumar

(Name of the Student)

143079023

(Roll No.)

Date: _____

Acknowledgements

I express my heartfelt indebtedness towards my supervisor, Prof. Virendra Singh. He has provided ample freedom to me for exploring a wide range of topics. He has also been generous enough to provide me with stipends and arrange my travel for one international conference. Additionally, I wish to express my gratitude towards Prof. Masahiro Fujita (University of Tokyo) for guiding me throughout these years and providing me valuable suggestions. I am also grateful to Prof. Kanad Basu (University of Texas at Dallas) for assisting me through few implementation tasks and assisting in improving the write-up of some of the collaborative papers. This thesis would not have been possible without active and sincere co-operation from many members of the CADS laboratory at IIT Bombay. I am particularly grateful to Jay Adhaduk, Atul Bhosale, Ankit Jindal and Prachi Sahu for collaborating with me. I am also indebted to Mr. Kentaro Iwata (former Masters student at University of Tokyo) for assisting me in one of the implementation tasks. I have also enjoyed collaborating with Vineesh V S (CADSL, IITB) on different aspects of formal verification of designs. I must express my indebtedness to Prof. Ashwin Gumaste (CSE, IITB) for funding me (for one-and-a-half years) through the collaborative project between him and Prof. Singh. I am deeply grateful to professors at IIT Bombay (particularly, Prof. M P Desai, Prof. D K Sharma, and Prof. S Patkar) for teaching me many courses and serving on my research progress committee. I am thankful to Vaishali madam (staff, CADSL) for assisting me in office-related tasks. Throughout these years, my family members have been the strongest pillar of my strength. No matter how hard I attempt, I can not express their contribution and sacrifice in words.

Publications

Included in Thesis

Journal publications

1. **Binod Kumar**, Jay Adhaduk, Kanad Basu, Masahiro Fujita, and Virendra Singh,“ A Methodology to Capture Fine-grained Internal Visibility during Multi-session Silicon Debug”, IEEE Transaction on on Very Large Scale Integration Systems (TVLSI), January 2020

2. **Binod Kumar**, Masahiro Fujita, and Virendra Singh, “SAT-based Silicon Debug of Electrical Errors under Restricted Observability Enhancement”, Journal on Electronic Test Theory and Applications (JETTA), October 2019

3. **Binod Kumar**, Kanad Basu, Masahiro Fujita, and Virendra Singh, “Post-silicon gate-level error localization with effective & combined trace signal selection”, IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems (TCAD), November 2018

Peer reviewed conferences

1. **Binod Kumar**, Atul Kumar Bhosale, Masahiro Fujita and Virendra Singh, “Validating Multi-processor Cache Coherence Mechanisms Under Diminished Observability,” IEEE 28th Asian Test Symposium (ATS), Kolkata, India, Dec 2019

2. **Binod Kumar**, Masahiro Fujita and Virendra Singh, “A Methodology for SAT-based Electrical Error Debugging during Post-silicon Validation,” 32nd International Conference on VLSI Design (VLSID) 2019, New Delhi, Jan 2019

3. Ankit Jindal, **Binod Kumar**, Kanad Basu, and Masahiro Fujita, "ELURA: a methodology for post-silicon gate-level Error Localization Using Regression Analysis," 31st International conference on VLSI Design (VLSID) 2018, Pune, Jan 2018

4. Ankit Jindal, **Binod Kumar**, Masahiro Fujita and Virendra Singh " Silicon debug with maximally expanded internal observability using nearest neighbour algorithm," IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2018, Hongkong, SAR, China, July 2018

5. **Binod Kumar**, Ankit Jindal, Masahiro Fujita, and Virendra Singh, "Combining Restorability and Error Detection Ability for Effective Trace Signal Selection," 27th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2017, Alberta, Canada, May 2017

6. **Binod Kumar**, Ankit Jindal, Masahiro Fujita and Virendra Singh, "Post-silicon Observability Enhancement with Topology Based Trace Signal Selection," 18th IEEE Latin American Test Symposium (LATS) 2017, Bogota, Colombia, March 2017

7. **Binod Kumar**, Kanad Basu, Ankit Jindal, Masahiro Fujita, and Virendra Singh, "Improving post-silicon error detection with topological selection of trace signals," 25th IEEE/IFIP International Conference on on Very Large Scale Integratiion (VLSI-SoC), Abu Dhabi, UAE, Oct 2017

8. **Binod Kumar**, Ankit Jindal, Jaynarayan Tudu, Brajesh Pandey and Virendra Singh, "Revisiting Random Access Scan for Effective Enhancement of Post-silicon Observability," 23rd IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS) 2017, Thessaloniki, Greece, July 2017

9. **Binod Kumar**, Ankit Jindal, Jaynarayan Tudu, and Virendra Singh, "A methodology for post silicon debug utilizing progressive random access scan architecture," 17th IEEE Workshop on RTL and High Level Testing (WRTLT) 2016, Hiroshima, Japan, Nov 2016

10. **Binod Kumar**, Ankit Jindal, Virendra Singh, and Masahiro Fujita, "A methodology for trace signal selection to improve error detection in post silicon validation," 30th International conference on VLSI Design (VLSID) 2017, Hyderabad, Jan 2017

11. **Binod Kumar**, Ankit Jindal and Virendra Singh, "A trace signal selection algorithm for improved post silicon debug," 14th IEEE East-West Design and Test Symposium (EWDTS) 2016, Yerevan, Armenia, Oct 2016

Other Contributions

1. **Binod Kumar**, Swapniel Thakur, Kanad Basu, Masahiro Fujita and Virendra Singh, "A Low Overhead Methodology for Validating Memory Consistency Models in Chip Multiprocessors," VLSID 2020, India

2. **Binod Kumar**, Akshay Kumar Jaiswal, Vineesh V S and Rushikesh Shinde, "Analyzing Hardware Security Properties of Processors through Model Checking," VLSID 2020, India

3. Vineesh V S, **Binod Kumar**, Rushikesh Shinde, Akshay Kumar Jaiswal, Harsh Bhargava and Virendra Singh, "Orion: A Technique to Prune State Space Search Directions for Guidance-Based Formal Verification," IEEE 28th Asian Test Symposium (ATS), Kolkata, India, Dec 2019

4. Vineesh VS, **Binod Kumar** and Jay Adhaduk, "Identification of Effective Guidance Hints for Better Design Debugging by Formal methods," 23rd International Symposium on VLSI Design and Test (VDAT) 2019, Indore, India, July 2019

5. Saurabh Gangurde and **Binod Kumar**, "A Unified Methodology For Hardware Obfuscation and IP Watermarking," 23rd International Symposium on VLSI Design and Test (VDAT) 2019, Indore, India, July 2019

6. **Binod Kumar**, Kanad Basu, and Virendra Singh, "A Technique for Electrical Error Localization with Learning Methods During Post-silicon Debugging," 10th

International Green and Sustainable Computing Conference (IGSC) 2018, Pittsburgh, USA, Oct 2018

7. **Binod Kumar**, Kanad Basu, Virendra Singh and Masahiro Fujita, "RTL level trace signal selection and coverage estimation during post-silicon validation," 19th IEEE International High-Level Design Validation and Test Workshop, Santa Cruz, California, U.S.A.,Oct. 5th-6th, 2017

8. **Binod Kumar**, Kanad Basu, Ankit Jindal, Brajesh Pandey and Masahiro Fujita, "A Formal Perspective on Effective Post-silicon Debug and Trace Signal Selection," 21st International Symposium on VLSI Design and Test (VDAT) 2017, Roorkee, India, July 2017

9. Toral Shah, Anzhela Matrosova, **Binod Kumar**, Masahiro Fujita and Virendra Singh, "Testing Multiple Stuck-at Faults of ROBDD Based Combinational Circuit Design," 18th IEEE Latin American Test Symposium (LATS) 2017, Bogota, Colombia, March 2017

10. **Binod Kumar**, Boda Nehru, Brajesh Pandey, Jaynarayan T Tudu, and Virendra Singh, "A technique for low power, stuck-at fault diagnosable and reconfigurable scan architecture," 14th IEEE East-West Design and Test Symposium (EWDTS) 2016, Yerevan, Armenia, Oct 2016

11. **Binod Kumar**, Boda Nehru, Brajesh Pandey and Jaynarayan Tudu, "Skip-Scan: A Methodology for Test Time Reduction," 20th International Symposium on VLSI Design and Test (VDAT) 2016, Guwahati, India, May 2016

Abstract

Due to tremendous growth in the complexity of modern designs, bugs inevitably escape the pre-silicon verification stage because of incomplete functional verification. Furthermore, given the menace of process variations and the inaccuracy in simulation models, a sufficiently verified design still stands probable of failing in first silicon. This necessitates verification of the design functionalities at the first silicon stage and is commonly referred to as post-silicon validation. Various industrial case studies have reported that this step requires 30-40% of the total time spent in the design development cycle. Analysis of the bug behaviors and localizing them to smaller portions of the design can assist in reducing this time. The step of post-silicon validation and debugging is challenging because of a variety of reasons such as lack of golden responses, highly restricted internal visibility, irreproducible nature of bugs and large error detection latencies. In this thesis, some of these problems are addressed and effective solutions to them are presented. Some of the major processor errata documents reveal the cause of many failures as linked to the operation of the cache coherence protocol (CCP) mechanism in multi-processors. We propose an on-chip signal logging method which helps in bug detection in the case of design errors and bit-flips (which are manifestations of various electrical failures) without requiring golden responses. The proposed methodology utilizes cache coherence protocol specifications to obtain the expected coherence transactions and the detector module flags an error once a mismatch is found between observed signal states and the expected signal states. The proposed logging mechanism decreases the error detection latency at minimal (less than 1%) area and power overheads.

With the availability of an electrical error trace, the localization of bit-flips is still

challenging because of the diminished observability of the internal signals. To solve this issue, the limited number of on-chip trace buffers are employed to store internal signals which can be used for analyzing and debugging. We propose a grouping-based signal tracing methodology for collecting useful internal signal values from chip execution. The proposed methodology analyzes logical connectivity in the design netlist for deriving different signal groups from which a limited number of signals are traced. With the traced data, different formulations as per satisfiability (SAT) are attempted. Since the structure of the netlist is correct, applying error trace as constraints assists in discovering the logical inconsistencies. During solving of SAT instances, the solver can provide clauses which are reasons behind the UNSAT behavior.

The success of post-silicon error localization with the help of trace buffer-based techniques largely relies on an appropriate selection of trace signals. The thesis proposes several heuristics-based selection methodologies to target error localization with traced and restored signal states. A selection methodology is also proposed which aims quick error detection by selecting from a topologically ordered signal. The selection is directed by a scoring mechanism that calculates the number of error propagation paths between different flip-flops. In order to further enhance the profitability of signal tracing, a combined signal selection methodology is proposed which enhances both properties of state restoration and error localization along with consideration of physical design choices of routing congestion and wire length. Although the application of state restoration technique enhances the limited debug data available through on-chip trace buffers, yet the number of restored signal states is not significant. A k-nearest neighbor (kNN) algorithm-based visibility expansion is proposed to maximally expand internal visibility. Based on the analysis of pre-silicon buggy signatures (states of flip-flops of the circuit), a set of neighbors is identified for every flip-flop of the design. By applying majority voting of the signals states of identified neighbors, the states of untraced signals can be predicted with an accuracy ranging from 70-99% on a wide variety of benchmark circuits. The expanded internal visibility is utilized for design error localization. To minimize the number of false positives during the error localization, a model-building methodology is

also proposed in the thesis. The model is built through error injection into very small portions of the design (netlist) for training. The model provides a ranking of suspect regions of the netlist given a particular circuit response for debugging.

The last portion of the thesis deals with compression issues during post-silicon debug. Typically, the debug exercise is carried out in multiple sessions which are characterized by run-and-halt steps. One of the important criteria for the success of this method is that the debug infrastructure captures only the erroneous data which adds important insights to the debug process. We propose a debug architecture for enhancing the multi-session procedure using the technique of debug data compression with on-chip MISR and XOR gates. The first session assists in identifying those erroneous clock cycles and the useful debug data are collected in the second session. This assists in better utilization of the on-chip storage. For achieving reconfigurable internal visibility, we present a methodology for post-silicon debug utilizing the special features of progressive random access scan (PRAS). The PRAS architecture offers a read-out of non-destructive scan values which is the bottleneck in the process of debugging. The proposed methodology avoids the large overhead of additional resources for debugging as the DfT architecture is reused. This debug scheme offers reconfigurability which enables selective visibility of internal states of a certain portion of the design.

The thesis encompasses effective techniques for solving multiple problems in the broader realm of post-silicon validation and debug. Methodologies are presented that achieve significant improvement in the areas ranging from trace signal selection to self-checking error detection and automatic error localization over the other methods proposed in the literature. The thesis also presents several important directions for meaningful extensions of the proposed techniques.

Keywords

Post-silicon validation, Design bugs, Post-silicon error localization, State restoration, Trace signal selection, On-chip compression, Randm access scan, Machine learning, Topology-based selection, Satisfiability-based localization, Cache coherence errors, Electrical errors, Model-building, Processor bugs, Gate-level error localization

Notations and Abbreviations

N_{clu}	:	number of clusters
T	:	length of error trace
B	:	Boolean formula(for SAT solving)
F_i	:	flip-flop (where bit-flip is injected)
C_i	:	clock cycle (where bit-flip is injected)
net_i	:	net (where stuck-at is injected)
TBw	:	trace buffer width
S_{Tr}	:	set of trace signals
k	:	number of partitions
Su_{fc}	:	suspect flip candidates (flip-flops)
Su_{cc}	:	suspect flip candidates (clock cycles)
Su_{sa}	:	suspect stuck-at candidates (nets)
L_{sc}	:	length of smaller scan chains
N_E	:	no. of error injection experiments
F_{tot}	:	total flip-flops in the netlist
UD_i	:	unrolled design (netlist) in i^{th} cycle
STB_i	:	state of traced signals in i^{th} cycle
PI_i	:	state of primary inputs in i^{th} cycle
PO_i	:	state of primary outputs in i^{th} cycle
CUD	:	circuit (or, core)-under-debug
L	:	number of clock cycles of CUD execution
W	:	data-word (signature) width
D_{tb}	:	depth of trace buffer
S_i	:	i^{th} debug session
$Bits$	:	CUD data to be stored in TB
G	:	S-graph
Gr	:	grid for layout of design
G_{lc}	:	logic cone connectivity graph
$TBits$	:	tag bits (for clock cycles) to be stored
Cy	:	set of debug data cycles to be stored in TB
$\|Cy\|$	:	number of debug data cycles to be stored in TB
ER	:	error rate in CUD execution
M	:	training design response model
ffZ_k	:	any *flip-flop zone* of the circuit
K	:	no. of iterations of training
Sle_i	:	signature used for training
Cy_{iter}	:	no. of cycles in each iteration of training
FF_j	:	error signature of j^{th} flip-flop for training
Ste	:	signature used for testing
$rank_j$	:	*rank* of j^{th} flip-flop
ff_j	:	error signature of j^{th} flip-flop for testing
M_{nbr}	:	k-NN learning model
L_i	:	i^{th} Nearest Neighbor technique
ϕ_j	:	numbers of nearest neighbors in M_{nbr}
FF_e	:	error signature of design
ξ_i	:	features of the design(netlist)
FF_t	:	flip-flop(s) signature used for testing
GFF_t	:	reference/golden flip-flop signature(s)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Post-silicon Validation and Debug

Demand for higher levels of integration coupled with the shrinking time-to-market window has drastically increased the chances of errors escaping to the first released silicon [15–20]. Pre-silicon verification techniques are not capable to establish full correctness of the design. This is primarily due to the reason that simulation of the design is at least million orders of magnitude slower than the real execution of the design post-fabrication [21],[22–26]. Due to exploding search space, formal methods fail in case of verification of complete designs. Furthermore, given the menace of process variations and the inaccuracy in simulation models, a sufficiently verified design still stands probable of failing in first silicon [5, 27–29]. Other factors responsible for silicon failures range from inaccurate modeling of physical effects in simulation to incomplete understanding of some of the design specifications [30, 31]. Figure 1.1 indicates the number of respins required for a successful product cycle. The fast speed of chip execution can be leveraged for debugging purpose as large number of test cases are applied to exercise the design functionality and is capable of exposing subtle bugs. This step commonly referred to as *post-silicon validation* aims to match the chip (first silicon) behavior with the end-user user expectations by running applications from many hours to days. Many studies suggest that about 70% of the overall time, efforts, and resources are spent during System-On-Chip

(SoC) validation (and verification) [15]. Therefore, the debug step needs great deal of attention for achieving time-to-market (TTM) targets.

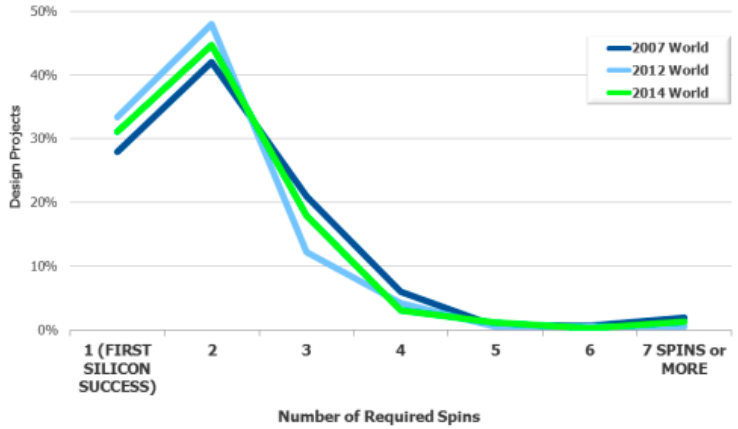

Figure 1.1: Design respins required, Mentor Graphics Study [4]

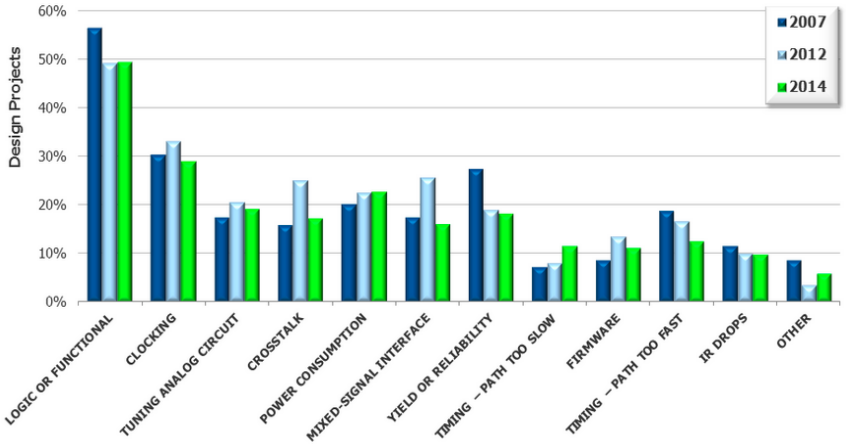

Figure 1.2: Causes behind design respins, Mentor Graphics Study [4]

As is shown in Figure 1.1, functional error are major source of the design respins. Thus, the random/constrained-random tests applied during pre-silicon verification stage are not always suitable to exercise those cases, resulting into their escaping into first silicon. If a large number of tests are applied, these can be caught and then localized. Diagnosing and debugging electrical failures in modern digital designs is relatively more challenging than functional errors. Unlike functional errors, electrical failures result in circuit failures only under certain conditions. For example, operating conditions such as voltage, temperature and frequency may change the behavior of the failure. Root causes

of electrical errors for a typical industrial microprocessor during post- silicon validation have been analyzed in [5], shown in Figure 1.3. For experimental evaluation, we assume their manifestation as flipping of bit in the flip-flop(s) of the design [24, 32].

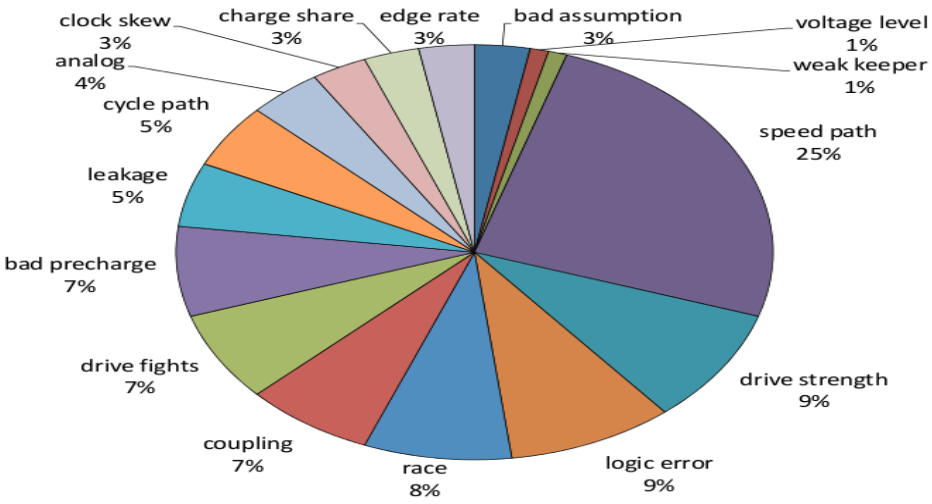

Figure 1.3: Sources of electrical errors [5, 6]

As stated earlier, during post-silicon validation step, we target the detection (and subsequent localization) of both design and electrical errors because of the fast execution speed of the manufactured chip. However, merely applying a higher number of test vectors for validation purposes does not completely solve the debug problem because of several other hindrances. Restricted observability and controllability of the internal design states along with the lack of failure repeatability and scarcely available golden responses are some of the major obstacles during the post-silicon validation and debugging [33, 34]. Scan chains can be utilized for a periodic monitoring scheme wherein their contents are taken out after some pre-defined intervals [35]. However, for subtle electrical bugs, this scheme becomes insufficient as the scan dumps can not provide necessary debug information. This necessitates DfD (Design-for-Debug) features for the continuous tracing of internal signals of the design for a fixed number of clock cycles. Additionally, due to faster execution and limited on-chip storage available (for debug purpose), compression mechanisms are inevitably required so that useful debugging can be achieved in quick manner. In this thesis, we aim to solve some of these issues by proposing effective techniques that are significantly promising compared to similar techniques of the

literature. We also make a case for reuse of the design-for-testability (DfT) feature for the purpose of silicon debug. In particular, we explore a variant of the random access scan (RAS) architecture for achieving internal debug data acquisition in a reconfigurable manner. Definitely, because of subtle issues, RAS has not yet been able to replace the serial scan chains in the designs. However, we believe incorporation of RAS may yield profitable results for both the manufacturing test and debug purposes.

1.1.1 Examples from Errata Documents of Processors

Sarangi et al.[36] identified the floating-point division bug in the Pentium processor as the most publicized design error, which led to a $475-million chip recall [37]. In 1999, another design error in the Pentium III temporarily halted shipment of Intel servers. Problems in the Pentium IV cache and prefetch engine temporarily led to disabling prefetching in multiprocessor systems. The authors in [36] stated that design errors led to a recall of Itanium II processors, incorrect results in the AMD Athlon 64, and circuit errors prevented the IBM PPC 750GX from running at 1 GHz. They have also claimed that almost every modern processor has tens or even hundreds of design errors, which manufacturers discover after shipment and publish in errata sheets. Typical reasons behind such design bug escapes are [38]: (a) a result of not fully reading the specification or starting the design implementation before the specification was complete, (b) situations in which the design was changed, usually to fix bugs or timing problems, and the designer did not take into account all the places that would be impacted by the design change, (c) architects not communicating their expectations clearly to the designers or some misunderstandings between microcode and design as well as between different parts of design (for instance- misassumptions about what another unit in the design was doing), and (d) problems in the micro-architecture definition or algorithm/micro- instruction/protocol was not documented properly. Specific examples (from errata sheets) have been provided by the authors in [36].

Some examples of errata sheets for memory sub-systems in modern processors is shown in Figure 1.4 [7]. Memory sub-systems are a major region which are prone to bug

escapes. We get motivation for cache coherence protocol (CCP) mechanism validation from this study as the CCP-related errors are most difficult to debug [39, 40]. Note that constrained-random tests provide very poor coverage while formal methods are infeasible for verification of complex protocols [41].

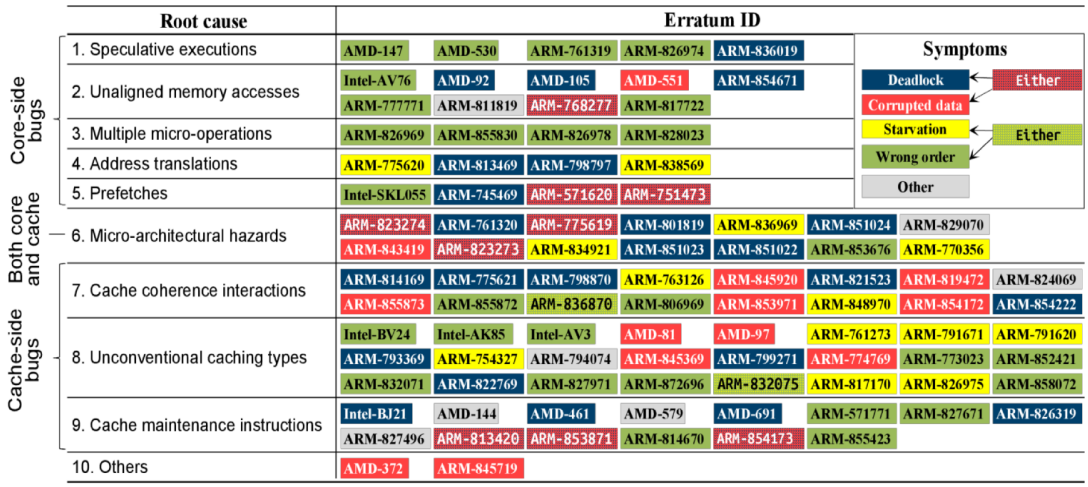

Figure 1.4: Bugs related to memory sub-systems in processor designs [7]

Constantinides et al.[8] have provided illustrations of design errors from 1) the Pentium 4 errata sheet, and 2) the Opteron errata sheet. Figure 1.5 shows a design bug in tlu_tcl module of OPENSPARC processor.

Illustration from tlu_tcl module of OPENSPARC processor

```
line 1089: assign intrpt_taken=
line 1090:             rstint_taken | hwint_taken | sirint_taken;
.................
line 1105: assign trap_to_redmode = trp_lvl_at_maxtlless1 & !(intrpt_taken);
line 1106: assign trap_to_redmode = trp_lvl_at_maxtlless1 &
line 1107:                                  !(rstint_taken | sirint_taken);
```

Figure 1.5: Design bug illustration at RTL [8]

The lines numbered as 1106 and 1107 are the actual RTL code and line 1105 (shown in red color) denote the buggy RTL code. From this illustration, it can be observed that gate-level bug models (like wire exchange between signals) can serve as representative of actual type of design bugs. This work focuses on the detection and localization of these kinds of errors at the gate-level abstraction of the design. Although at gate-level,

the high level functionality description is lost, we can analyze in terms of the gates and flip-flops present in the netlist that can provide better diagnosis and debug capabilities [42–44]. The netlist represents the design in post-silicon environment.

1.1.2 Observability Enhancement Issues

Mitra et al.[21] present various challenges in the process of post-silicon validation. The most important out of them is the limited observability of the internal states of the design due to the limited number of externally accessible pins. Abramovici et al.[45] proposed on-chip trace buffers as a solution to address this issue. A subset of internal signals of the design are selected and fed to the buffers. These signal values are used to observe the states of the chip without halting its execution like the scan chains.[1] Two industrial examples are: Intel TraceHub [46] and ARM CoreSight [47].

Depending on the occurrence of some event triggers, the selected trace signals can be dumped in the buffers. However, as selection of the trace signals must be performed at the design time, this becomes a complex problem. The difficulty further increases with the enhancement in design complexity. For instance- if 32 trace signals are to be selected out of total 1728 flip-flops of a design, there are $\binom{1728}{32} = 10^{69}$ such possible combinations. Clearly, an exhaustive evaluation of all such combinations is neither feasible nor profitable in terms of computational expenses. Hence, efficient trace signal selection is very crucial [22],[32, 48–51] and needs a systematic approach. Another major impediment in the success of validation of the first silicon is the absence of functional coverage models at the post-silicon stage. During pre-silicon verification, assertions help in detecting design misbehavior. A similar scheme can be envisaged for post-silicon stage too by synthesizing on-chip assertion checkers. However, this is very expensive from the viewpoint of area overhead. For instance- if checkers for all assertions for ISCAS'89 benchmark *s35932*

[1]The need of halting the chip execution is not fully eliminated with the trace buffer mechanism because when the buffers are full, to offload their contents, we need to stop the execution in order to avoid over-writing of traced contents. However, at least for the depth of the trace buffer (typically 1024 cycles), the dumping is continuous. In the next run, the contents of the scan chain from the last run is fed back to maintain state continuity in the chip execution.

circuit are synthesized on-chip, the area overhead is around 20 times of the original circuit [52]. Therefore, it is important to obtain functional coverage (i.e., occurrence of particular events during design execution in the post-silicon environment) without incurring large on-chip overhead.

1.1.3 Validation in a Self-checking Manner

Owing to the large error detection latency of subtle bugs and slow speed of pre-silicon design simulation, it is very difficult to obtain golden response (of internal signals, specifically, flip-flops) for a large range of clock cycles. This leads to the search of self-checking methods which are capable of validation without requiring correct responses and lead to an effective bug localization. Design components that constitute protocol-level specifications can be easily validated in this manner. Once the protocol is proven correct (say, through formal methods), it's implementation and interaction with the rest of the system can be checked for any design/electrical bugs with the help of high-level specifications of the protocol. Essentially, this type of validation can be understood as consistency-checking of the implementation. Addition of some on-chip hardware can assist in the collection of important signals/messages related to the execution which need to matched with the design specifications. However, the key challenges in this direction are achieving minimal error detection latency and ensuring tolerable overheads (because of on-chip realization). Implementations of cache coherence mechanisms, which are notorious for causing bug escapes, can be validated in this manner. For electrical bug scenarios, a smaller error trace can be enforced on the unrolled design netlist description to achieve consistency-based checking. When the resulting formula turns out to be UNSAT, the group of clauses behind its unsatisfiability can be obtained. This does not require the golden responses as the trace (because of electrical error) lead to inconsistencies when applied on a structurally correct design netlist.

1.2 Thesis Contributions and Organization

In the modern era of designs with enormous complexity, limited accessibility of internal signals is one of the main obstacles during post-silicon validation. Along with addressing this problem, this thesis addresses some other related problems such as validation without requiring the golden responses, on-chip debug data compression and automatic bug localization (at gate-level). The thesis overview is shown in Figure 1.6 where the individual chapters are marked against main contributions.

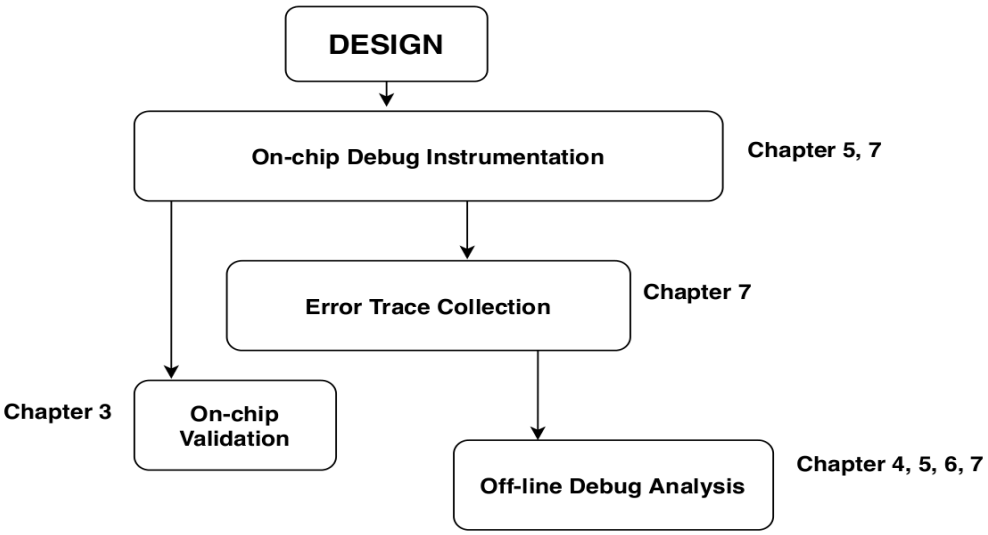

Figure 1.6: Overview of Thesis

The contributions[2] of the thesis are outlined as below:

- Chapter 2 presents the literature survey related to post-silicon validation and debugging. The limitations of the prior art are also briefly discussed in relevant sections of this chapter.[3] It also presents our motivation behind improved techniques for trace signal selection and automatic gate-level error localization.

- Chapter 3 elaborates on the proposed self-checking (i.e., without requiring golden responses) methodology for validation of cache coherence mechanism in chip multiprocessors. The self-checking methodology is aimed at the minimization of error

[2]The contributions are listed in the way the chapters are written in the thesis.
[3]The limitations of previous work are discussed in detail in the individual chapters.

detection latency and overheads.

- Chapter 4 explains the proposed self-checking methodology of satisfiability (SAT)-based error localization of bit-flips (which can model electrical errors).

- Chapter 5 presents the proposed methodology for topology-based trace signal selection. It also presents our combined trace signal selection and error localization methodology using traced and restored signal states.

- Chapter 6 describes the proposed methodology of complete internal visibility expansion and the subsequent error localization with this approach. It also explains our method of low-level error localization using design response model building approach. Both of these techniques utilize some learning from the traced (and restored) signal states for the purpose of effective error localization.

- Chapter 7 explains the proposed method of on-chip compression of debug data in a multi-session silicon debug procedure. This chapter also presents a reuse scheme for random access scan (RAS), that poses a viable alternative to scan chains as a design-for-test (DfT) solution, in the silicon debug step.

Finally, the thesis is concluded in Chapter 8 with a listing of contributions and future scope arising out of the work done in this thesis.

$- * - * -$

Chapter 2

Previous Work

It has been widely reported that pre-silicon verification techniques are not capable to establish full correctness of the design [4, 15–20, 27, 53, 54]. This is primarily due to the reason that simulation of the design is at least six to seven orders of magnitude slower than real execution of the design post-production [21, 55]. Only a few sub-modules can be completely verified by formal methods as they suffer from state space explosion problem as the number of state elements increases in the design [56]. Furthermore, owing to the other issues like process variations and inaccurate simulation models, post-silicon validation has attained an important position in the modern design implementation cycle [45]. Scan chains available in the design for DfT (Design for Test) purposes also offer insights into the internal states of the chip, however, this information is not real-time as the execution of the chip has to be stopped for dumping the scan values [57, 58]. Hence, on-chip trace buffers need to be utilized for debug data acquisition [59]. BackSpace [56] is a formal method proposed by De Paula et al. for post-silicon validation using reachability analysis and pre-image computation from the trace buffer contents. An approach proposed recently by Taatizadeh et al. [52] implements the idea of on-chip assertion checking for catching bugs (of bit-flip types). On-chip checkers can also play important role in functional validation [60, 61]. Placing on-chip assertion checkers, however, raises serious concerns of area overhead [62, 63]. In the next sections, we discuss some of the most important and relevant work from the literature targeting error

detection and localization techniques and debug methodologies.

2.1 Architecture-level Validation Methods

Foutris et al. [64] utilized diversity in Instruction Set Architecture (ISA) to expose design and electrical bugs in processor systems by the execution of random instruction tests and corresponding equivalent random instruction tests. Wagner et al. [65] proposed *Reversi* which involves inverse state computation for instruction blocks for detecting and subsequent localization of bugs. A major benefit of this self-checking approach is that time-consuming pre-silicon simulation results are not needed. To reduce the error detection latency in the validation of processor designs, duplicated programs can be run with instruction for checking inserted in the test programs [66–68]. Clearly, these kinds of approaches are not applicable for general digital blocks within complex SoCs. Park et al. [69] proposed a technique named as *Instruction Footprint Recording and Analysis (IFRA)* for error detection in processor systems using low cost on-chip recorders as an observability mechanism. They perform program analysis after constructing bug localization graphs [70] for error localization to a block-level granularity. However, typically an architectural block may contain a complete SoC. This makes it very difficult to localize/debug at the netlist level with techniques like *IFRA* [69] or that of *Reversi* [65]. Friedler et al. [71] proposed a method for automatic architectural localization of post-silicon test-case failures by mismatch of states from chip execution and the states obtained from executing the similar test on an Instruction Set Simulator (ISS). Leveraging this information helps in identification of a set of instructions that could lead to the faulty final state in a buggy microprocessor design.

Modern processors employ complex cache coherence operations for deployment of multi-core configurations. Verifying the implementation of a cache coherence protocol (CCP) in processor designs has been an active area of research for many years [41, 72–74]. However, despite significant success in the formal verification of CCP, errors related to operation of the cache coherence mechanisms of modern processors escape into the silicon

[7, 39]. Such bugs are a major contributor to the complex errors in processor errata documents and thus need measures for correction through hardware patching. Alternative to this, on-chip verification techniques can assist in the detection of bugs [74] at the cost of performance and area overheads. Cache coherence mechanisms are also susceptible to soft-errors and thus require effective fault-tolerance mechanisms [75, 76] to mitigate them. For snoopy-based CCP implementation, checkers have been proposed in [75] to mitigate soft errors. The technique in [75] employs distributed checkers which has a local store with tag information for all cache lines. Based on the incoming messages, checkers maintain correct state of each cache line in the separate (checked) cache. Compared to technique of watchdog processors in [74], the overheads are lower in [75]. The approach in [74] uses checker circuits on each cache line and implements a simplified version of CCP. Whenever state of cache line changes, related information is sent to the checker. The checker circuit recomputes coherence transactions and verifies the states provided by the caches. Apart from errors related to coherence mechanism operation, bugs in the operation of memory sub-systems have held a prominent place in various documented processor errata [7, 18, 77]. Such bugs are very hard to localize because of longer error detection latency and manifestation in different forms owing to their non-deterministic behavior [18, 78]. Errors in the ordering of memory operations are very hard to detect during simulation-based verification because of unclear specifications and a wide range of legal and valid execution results. The problem stems from the fact that architects deviate from the program order (which is provided by the programmer) and allow ordering of memory operations during execution. Although, the rules and semantics of such permissible orderings is formalized by *Memory Consistency Models (MCM)* [79], verifying the hardware implementation of a MCM is not straightforward [80]. The fundamental reason behind this is the fact that any sort of relaxation from the program order leads to a wide range of legal and valid execution results further increasing the non-determinism in MCM behavior. Moreover, the cache coherence mechanism which is inevitably needed for the correct operation of multiprocessors fails to ensure that MCM rules are followed always. This is because the cache coherence rules are concerned only

with a single memory location access while consistency caters to a global view of the memory access ordering. Meixner et al. [81] proposed a method of dynamic (on-chip) validation of memory consistency through invariant checking. Chen et al. [80] proposed run-time validation of the memory consistency models by performing on-line analysis of the constraint graph-based analysis. DeOrio et al. [77] proposed a hardware-based logging mechanism that records the ordering of memory events. Mammo et al. [82] proposed a hardware-based memory access tagging methodology for tracking the memory access orderings for post-silicon consistency validation. One of the earliest work in software-based memory consistency verification (primarily at the pre-silicon level) was done by Hangal et al.[78]. They developed a tool for validating Total Store Order (TSO) specifications in the processor designs through specially crafted test programs. On similar lines, [18] is a method of software-based post-silicon validation of memory consistency models. They instrument instructions in test program to capture loaded values. The bug detection in [18, 77, 78, 80, 82, 83] is facilitated by checking for cyclic property in the constraint graph, an idea utilized initially in [84].

Post-silicon validation approach for validation of CCP implementations is proposed in [85] and [39] at low performance and area overheads. Wagner et al.[85] proposed usage of a specific pattern-based error detection along with bypassing and programmable finite state machine for correction of coherence errors. DeOrio et al.[39] proposed a technique to match the coherence transactions between the cache hierarchies (L1 and L2 caches) through string matching of the logged contents (i.e., CCP state transitions) at both levels. These techniques have error detection latency in the range of millions of clock cycles based on their reported results on a high-level architectural simulator. It is worth to note that an architectural simulator fails to capture the finer intricacies of the complex designs [86, 87] and hence subtle bug manifestations may not be captured in an architectural simulator with fidelity. It is because of this reason, we attempt validation with the RTL description of a multi-core design. The RTL design description is relatively more closer to the post-silicon environment.

2.2 Gate-level Post-silicon Error Localization

Park et al. [69] proposed IFRA (Instruction Footprint Recording and Analysis) for debugging processor systems for errors with a block-level spatial localization by the use of low-cost hardware recorders. However, such an architectural block in a processor typically contain thousands of gates which makes it very difficult to debug at the gate-level. Furthermore, IFRA assists in localizing the bugs only when errors are detected on-chip within about a thousand cycles. In the recent years, machine learning techniques [88] have been utilized for the purpose of bug triaging or error localization at both the pre-silicon and post-silicon stage [89–92]. Poulos et al.[89] proposed grouping of different pre-silicon error traces obtained by SAT-based debugging into various groups through regression analysis. DeOrio et al. [90] proposed a post-silicon bug diagnosis methodology based on data collection from failing tests and then applying clustering technique to form signal groups. Since during post-silicon validation, a large number of tests are applied, the amount of logs collected is also very large. Therefore, machine learning techniques like clustering, classification methodologies can be deployed to extract hints for bug/error localization from the obtained test response logs [90, 91]. Mammo et al. [93] have proposed an automatic mismatch diagnosis approach for exactly identifying the buggy unit(i.e., processor block) with the assistance of hints from instruction set simulation in addition to the normal test execution on DUV (design-under-verification). In testing phase, the developed model assists in predicting the error location with the help of carefully engineered features (with the help of design functionality behavior) from the test execution log. Such feature engineering can not be easily adopted for uncore components/non-regular designs. Similar to [90], Bertacco et al. [91] proposed a mechanism to detect root-cause signals by identifying "outliers" signals from post-silicon tests and iteratively selecting signals to be monitored. Khudia et al. [92] have analyzed bugs that manifest inconsistently over repeated executions of the same test. They classified the internal signals into *passing groups* and *failing groups* for bug diagnosis upon test executions. A methodology to bridge pre-silicon and post-silicon verification by on-chip

detection of protocols (which have been extracted during pre-silicon) and then performing an off-line software diagnosis for localization at architectural block level and the corresponding buggy signal along with the detection cycle has been proposed in [94]. For general design blocks of SoC's, utilizing state restoration [1, 95] technique[1] assists in enhancing the restricted visibility which in turn would ease the localization principles. However, increment in state restoration does not translate to the ease in error detection/localization in the same proportions [96]. Thus, complete discovery of internal signal states can overcome this bottleneck.

Satisfiablity solving (SAT) is one approach that holds the promise of assisting us to localize to gate-level in a self-checking manner. Given that the structure of the design is correct, however, it is providing wrong functional responses because of electrical error(s) and we have obtain a smaller size error trace (in range of hundreds of clock cycles). Under this scenario, the error trace can be applied as constraints to the unrolled design netlist. The unrolling of the netlist is illustrated in Figure 2.1 where the unrolling is done for two cycles only. The CNF representation over 2 clock cycles is given by:

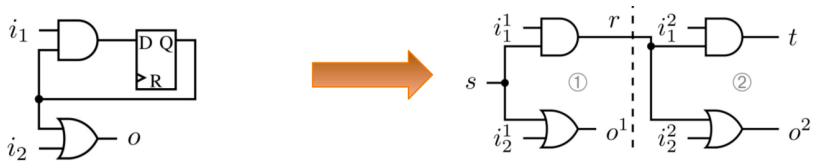

Figure 2.1: Unrolling a design into time-frames

$(\overline{r} + i_1^1) \cdot (\overline{r} + s) \cdot (\overline{i_1^1} + \overline{s} + r) \cdot (\overline{i_2^1} + o^1) \cdot (\overline{s} + o^1) \cdot (\overline{o^1} + i_2^1 + s) \cdot$
$(\overline{t} + i_1^2) \cdot (\overline{t} + r) \cdot (\overline{i_1^2} + \overline{r} + t) \cdot (\overline{i_2^2} + o^2) \cdot (\overline{r} + o^2) \cdot (\overline{o^2} + i_2^2 + r)$

Here, the superscripts denote the clock cycle (or, time-frame) number. After the clauses corresponding to the error trace are applied to the above netlist expression in a conjunctive manner, the resulting formula is applied to the SAT solver. Since the error trace contains a manifestation of electrical error, the final formula would be unsatisfiable. A proof trace can be extracted which can lead us to the reason behind functional failure (i.e., the failing flip-flop and the corresponding clock cycle) [34]. SAT-based methodology

[1]This notion of signal state restoration is totally different from the widely used terminology of state restoration in the field of processor design/computer architecture.

has been successfully applied in a few approaches for localization under a post-silicon environment from the efforts developed for pre-silicon stage debugging [97–99]. Vali et al. [34] presented a bit-flip detection-driven trace signal selection methodology and evaluated its efficiency on ISCAS'89 type benchmark circuits. They have considered a trace buffer width of 128, 256 and achieved spatial localization of around 20 to 40 flip-flops. Their temporal localization ranges from 5 or 10 to 20 cycles of the actual flip-cycle. Compared to this approach, the presented methodology (Chapter 4) achieves temporal and spatial localization with lesser TBw. We evaluate the localization attempts on both bit-flip and syuck-at errors. Zhu et al.[48] proposed a sliding window (corresponding to a small number of clock cycles of chip execution) based technique to localize stuck-at constant faults in the netlist. Debugging is assisted by computation of *backbones* which are set of signals (in the unrolled netlist) that are immutable under the chip execution constraints such as the signal values from the trace buffer or primary input (PI) and primary output (PO) values. Leveraging *backbones* computation, a sliding-window based approach is presented in [100] also for localizing design errors like gate replacements in the post-silicon environment by considering a trace of 5% of the total signals. The SAT-based methodology involving UNSAT core(s) can be applied in conjunction with the technique involving backbones computation. Yang et. al.[101] proposed a satisfiability-based register-transfer level (RTL) error localization using hierarchical knowledge of the design and utilized a group of 16 registers as trace buffer width.

2.3 Effective Trace Signal Selection

Typically, error localization becomes a bottleneck during post-silicon validation and debug phase because of the large number of possibilities for complex designs even if an efficient methodology of error-detection [66] is adopted. Under such a scenario, a quick debugging of the error scenario can be performed if the on-chip traces collected are highly relevant and useful which can save us the time spent during off-line processing and analysis. Majority of the trace buffer-based signal selection techniques [1, 9, 102–104] have not

attempted to evaluate the quality of debug data collected from trace buffers. One of the measures of evaluating trace data merit is error detection latency (EDL). Additionally, the ease of bug localization (which in turn decides how quickly the bug fixing can be done) must be of pivotal importance in deciding the merit of a particular trace signal selection technique.

Due to the constraint of area overhead, effective trace signal selection is essential for a profitable implementation of the scheme of on-chip trace buffers. Under this scenario, maximization of restored states is one objective which ensures the observability of internal states is enhanced [1, 9, 102, 104–107]. Restoration is the process of deduction of unknown signal states with the help of known signal states through the process of logic implication (i.e., either forward propagation or backward justification or combined justification as illustrated in Fig. 2.2). From a broader perspective, the various restoration-based signal selection methods can be classified into 3 major groups: a) heuristics exploiting design structure [1], b) techniques based on design simulation [9, 105] and c) a mixture of design structure and simulation-based techniques [102, 104].

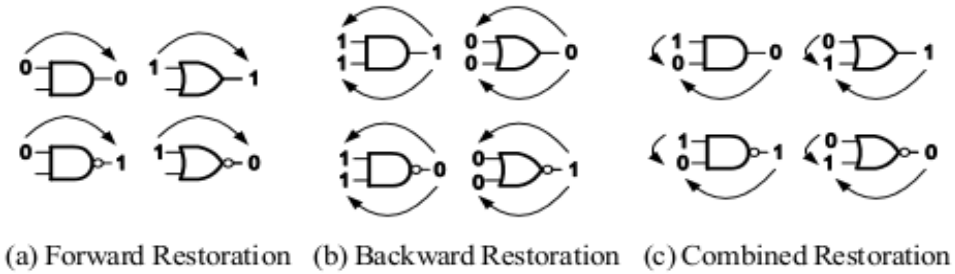

Figure 2.2: Restoration Illustration [9]

The quality of selection is characterized by a selection efficiency metric, called as "restoration ratio (RR), often also referred to as state restoration ratio (SRR)" which is given by the Equation 2.1 (Higher RR, higher is the internal observability) [1, 9, 22, 102, 104, 108–110]:

$$RR = \frac{Signals\ restored\ + Signals\ traced}{Signals\ traced} \tag{2.1}$$

For the example circuit shown in Figure 2.3, the traced and restored signal states are shown in Table 2.1 (X depicts an unknown signal value).

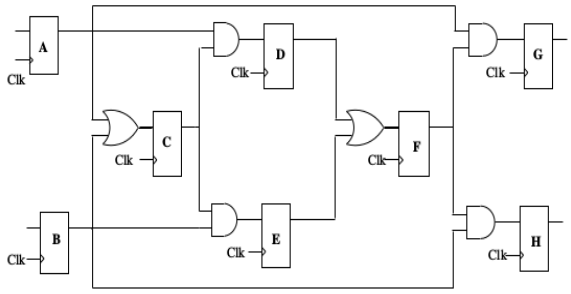

Figure 2.3: Example Circuit for illustrating illustration [1]

Table 2.1: Restored and traced signal states [1]

Signal	Cycle1	Cycle2	Cycle3	Cycle4
A	0	0	0	0
B	1	0	1	0
C	1	1	0	1
D	X	0	0	0
E	X	1	0	0
F	X	X	1	0
G	X	0	0	0
H	X	X	0	0

It has been observed that selective dumping of signals from a larger pool of candidate signals is superior to a fixed selection of trace signals [111]. However, dynamic trace signal selection is achieved at the cost of area overhead due to the multiplexers in addition to the overhead incurred due to the usage of trace buffers. To enhance restorability maximization, combination of select trace signals and a higher number of flip-flops connected in scan chains (illustrated in detail in Section 2.3.3) was suggested since this scheme covers a larger range of FFs as compared to tracing only few flip-flops (trace signals) [112].

In the literature, few attempts have been made to critically examine the utility of SRR. Hung et al.[113] proposed two different metrics for evaluation of the selected trace signals for ease in observing the incorrect behavior due to design bugs in post-silicon

environment. Ma et al.[96] have strongly argued against SRR stating various reasons for its inefficacy. One of them is the equal assignment of priority to all the signals without taking into account the particular nature of some bugs which have more chances to occur in certain regions of the design only. Secondly, the conditions under which different signals of the design need to be monitored are not identical.

2.3.1 Motivation behind Alternative Signal Selection

To illustrate the indifference of trace signal selection algorithms towards actual localization of errors, we simulated the RTL descriptions of the example circuit in Fig. 2.2 (utilized in previous works [1, 9]) for two different types of errors.[2] First, the random gate replacement ($B1$ type) and second, exchange of some of the wires of the circuit with one another ($B2$ type). Under some circumstances, these kinds of bugs in the design implementation may not be easily detectable at the pre-silicon stage. Suppose, the next state logic of a design contains a 2-input XOR gate which is replaced by a 2-input OR gate. The only input combination that can identify this gate replacement is "11". However, if the corresponding state is rarely activated in the state machine of the design, then pre-silicon verification fails to identify this change of gate. A similar analogy can be drawn for $B2$ type bugs too for justifying their relevance.

Tables 2.2 and 2.3 respectively report the error detection latency (i.e., the first clock cycle in which flip-flops capture an erroneous bit as compared to the golden response bits).[3] Observing an erroneous bit at the flip-flop can give a quick indication of the actual error/bug location. For each type of error, four iterations of error injection were done. An entry of "ND" in the tables indicate that the flip-flop could not observe the error or the error did not propagate to these flip-flops.

Apart from the combination of {C,F}[50, 114] some other suggestions for trace signal

[2]More details on these are discussed in Section 5.9.3.1 of Chapter 5.

[3]In strict sense, the usage of term "latency" is not appropriate. However, considering the source of error as cycle 0, we compute the first clock cycle in which flip-flops capture an erroneous bit as compared to the golden response bits. So, it can be termed as error detection latency.

Table 2.2: Latency for observing $B1$ type bugs

Error	A	B	C	D	E	F	G	H
e1	ND	ND	ND	2	ND	3	4	5
e2	ND	ND	2	4	3	4	5	5
e3	ND	ND	3	4	5	5	2	7
e4	ND	ND	ND	ND	ND	ND	ND	2

selection for the example circuit (Fig. 2.2) include {A,C}[1] and {A,B}[9] for achieving high restoration ratio. If state restoration techniques were to effectively assist in error localization/detection, these combination of signals should be able to capture the erroneous response bit(s) as early as possible for all kinds of errors.

Table 2.3: Latency for observing $B2$ type bugs

Error	A	B	C	D	E	F	G	H
e1	ND	ND	2	3	ND	3	4	5
e2	ND	ND	ND	ND	ND	2	2	3
e3	ND	ND	ND	ND	3	4	4	5
e4	ND	ND	2	ND	5	ND	3	ND

It is evident from the above tables that the flip-flops which are suitable for high SRR values do not always observe the erroneous response. Typically, when two signals are highly correlated if one is traced the other can be almost fully restored leading to high SRR values. However, tracing one of these correlated signals turns out to be ineffective for error localization as if the traced flip-flop fails to capture an erroneous response, the restored states follow the same trend.

The trace signal selection methods classified in the previous sub-section are applied on the design description in the netlist (gate-level synthesized) form. However, signal selection can be performed at RTL level too [1, 115]. The data flow graph (DFG) or control data flow graph (CDFG) is generated from the design description and register-variables (possible candidates of signal selection) are chosen based on the relationships between different variables of the design.

2.3.2 Relevance of Restoration Ratio as Signal Selection Metric

Ma et al.[96] have strongly argued with empirical evidence that RR (defined in Section 2.3) as a metric overlooks certain subtle aspects of the signal selection problem. The authors state that the principle of restorability maximization favors large arrays for the design. This is because tracing some elements of big arrays helps to restore a larger fraction of internal states of the design. However, tracing such elements do not provide useful hints for error localization. Another argument which authors in [96] put forward against using RR as a metric is the fact that it does not take into account natural design structure. The ignorance of the original design structure leads this metric to consider all the signals equally which is not true from the viewpoint of debugging.

We have shown results for some design errors on ISCAS'89 benchmark circuits which indicate that state restoration technique performs poorly on the account of error detection/localization [51, 55]. In [55], assignment of ranks to flip-flops through simulations based on pre-determined list of netlist-based bugs/errors. A simplified approach to the methodology of [111] is presented in [116]. The techniques described in [55, 116] apply erroneous traced state bits (tsb_{err}) for a fixed trace buffer configuration as a metric which is given by the following equation:

$$tsb_{err} = \frac{erroneous\ traced\ signal\ states}{total\ number\ of\ traced\ signal\ states} \tag{2.2}$$

Compared to RR [1, 9, 102, 104, 105], the above metric helps to select trace signals which can detect more errors as compared to signals selected by restorability maximization techniques. Among other important parameters which must be investigated include the trigger conditions for deciding the intervals of on-chip signal dumping [117] and generation of proper input stimuli. For in-system validation, based on assertions and property specifications, inputs can be derived and delivered to internal blocks of the design via some on-chip storage. However, facilitating such in-system validation tasks for complex designs is not trivial and requires proportionate on-chip infrastructure.

2.3.3 Different Types of Post-silicon Observability Enhancement Techniques

As stated previously, trace buffers are widely utilized as a design-for-debug feature for storing a selected group of signals for a specified number of clock cycles, which can then be dumped off-line for debugging [1, 15, 45, 59]. However, given the constraint of area overhead, number of signals to be traced is typically 1-2% of the total signals. Furthermore, the signals utilized for tracing in the post-silicon environment are decided at the design stage. When reconfigurability is intended in the tracing of internal signals, a multiplexed scheme needs to be employed [50, 118]. Table 2.4 shows a scenario where trace buffer width is 4 and buffer depth is 6 (denoted by the six clock cycles). Let us consider that FFA, FFB, FFC and FFD are four trace signals. Note that FFA_1 denotes the signal state of FFA in the first clock cycle. The contents of these trace signals(flip-flops) are continuously dumped into the trace buffers for six clock cycles.

Table 2.4: Illustration of signals for tracing

Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6
FFA_1	FFA_2	FFA_3	FFA_4	FFA_5	FFA_6
FFB_1	FFB_2	FFB_3	FFB_4	FFB_5	FFB_6
FFC_1	FFC_2	FFC_3	FFC_4	FFC_5	FFC_6
FFD_1	FFD_2	FFD_3	FFD_4	FFC_5	FFD_6

To aid in the process of internal signal tracing, some flip-flops can be combined into fixed length scan chains[4] which provide temporal variability in signal tracing [22, 103, 112, 119]. This variability in the temporal tracing of internal signals leads to a higher state restoration ratio [22] and also assists in error detection enhancement [119]. This mechanism needs the usage of shadow flip-flops and the dumping frequency of each signal is determined by the length of the chain. These mechanisms require a small controller

[4]This is slightly different from the notion of scan chains widely utilized in manufacturing testing. In this context, the input to the chain comes from the circuit components and the output is fed to the trace buffer. The dumping of contents as illustrated in Table 2.5 (where trace buffer width is reduced to 2 while the buffer depth is maintained at 6) shows the connection of the signals in the chain and the resulting dumping frequency.

Table 2.5: Combination of signals for temporal variability

Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6
FFA_1	FFB_1	FFA_3	FFB_3	FFA_5	FFB_5
FFC_1	FFD_1	FFC_3	FFD_3	FFC_5	FFD_5

required for managing the dumping of signal contents in trace buffers [22, 103, 120, 121]. In a static signal tracing scheme (as illustrated in Table 2.4), only 2 signals can be dumped for 6 clock cycles, whereas in this scheme, 4 signals (flip-flops) meaning that all the signals are dumped in alternate cycles. Since a larger number of internal signals can be covered for signal tracing with this observability enhancement scheme, error localization can be more effective. Note that the dumping of signal contents into the trace buffers is not continuous here and the dump frequency is decided by the length of these smaller chains.

2.4 Debug Data Compression Techniques

Due to the huge amount of data being generated during the step of post-silicon validation and limited on-chip storage, there is a need of effective compression mechanisms. Additionally, owing to the slow speed of offload dumping of internal contents (as chip execution is in range of Gigahertz while the offloading speed may be in range of tens of Kilohertz or Megahertz), if an on-chip compression scheme is adopted, the amount of debug data to be offloaded can be significantly reduced. This is because with compression, the chances of overwriting the data stored in on-chip trace buffers is minimized. Additionally, in a complex debug scenario, it is needed to halt the chip execution to facilitate the offloading of the contents of on-chip trace buffers. The debug procedure eventually requires mechanism to state continuity because of this halt of execution. With on-chip compression, since the amount of debug data to be stored in the on-chip storage gets reduced, the number of halts also decrease in number subsequently. For repeatable debugging experiments, scheduling of debug experiments in different sessions has been proposed by Anis et al. [122–125]. Compression of trace buffer data (through various

encoding schemes and their on-chip realizations) has been proposed in [126, 127] to reduce the amount of data to be stored after on-chip tracing. Since the debug exercise needs to be carried out in multiple sessions (which essentially mean multiple run-and-halt intervals), on-chip compression can play an important role in the effective utilization of on-chip storage and the reduction of overall debug time. However, we must ensure that the on-chip compression mechanisms have sufficiently low area and power overheads. For debug scenarios involving large number of experiments, significant improvement can be obtained with on-chip compression to get large expansion in the observation window. Typically, under these scenarios, the success of debug experiments is limited by the trace buffer capacity. Due to the efficient use of the on-chip trace buffers, a larger range of clock cycles can be observed at one time, leading to reduction in the total number of iterations required to analyze the debug data.

During manufacturing testing, serial scan chains offer observability and controllability at the expense of elongated test application time, inflated test data volume and excessive test power. This poses serious challenges to test strategies using serial scan chain architecture in the nanometer regime [128]. As a better alternative, random access scan (RAS) originally proposed by Ando [129] was re-investigated by Baik et al. [130] to address these issues and a slightly modified version known as progressive random access scan (PRAS). This scheme aims at a simultaneous minimization of test volume, power and time. Instead of stitching all flip-flops into a single chain, RAS (or, PRAS) felicitates selection of a single flip-flop for the purpose of writing the test vector bit or for the task of reading out the response bit. The mechanism uses a MISR (multiple input signature register) for obtaining time compaction of the responses generated by the application of test vectors. Hence, with the usage of MISR, the on-chip compression of test responses can be achieved. We view the PRAS architecture as an alternative debug scheme that offers on-chip compression. Apart from usage in structural testing, this mechanism can be utilized for the purpose of functional debug also with the exception that manufacturing test vectors need to be replaced by functional inputs. Essentially, we can achieve reconfigurable visibility of the internal signal states of the design with the usage of PRAS

architecture. This serves as efficient reuse of the infrastructure available on-chip (for the purpose of manufacturing test) in the silicon debug stage.

$- * - * -$

Chapter 3

Validating Multi-processor Cache Coherence Mechanisms

3.1 Introduction

Chip multi-processors need local caches for effective performance enhancement [2]. This is facilitated by complex cache coherence protocols (CCP) which maintain the correctness of caching operations in the multi-core system. However, it is well known that the functional verification of cache coherence protocol implementation is a challenging problem [72–75, 85]. This leads to bugs releasing into the errors in memory sub-systems of the multi-core[1] systems in the first silicon [39]. Since pre-silicon simulation is generally slow in nature, a large number of test cases covering all possible cache coherence transactions can not be exercised. Due to the fast execution speed of the silicon, comparatively higher number of complex cache coherence transactions can be analyzed. However, subtle errors such as those related to cache coherence mechanisms have very large error detection latency and their manifestations can not be easily detected [39, 85]. Furthermore, there is highly restricted observability at this stage which makes debugging very difficult. The internal signals of the complex coherence mechanism such as those involved in the maintenance of states of cache lines can not be observed directly and it can not be

[1] core and processor are often used interchangeably in this chapter.

determined if the state (whether it is *shared* or not, etc.) of each cache line is correct or wrong. Therefore, for detection of cache coherence errors, on-chip logging/observability mechanisms prove useful at the cost of limited area and power overheads. Apart from the limited visibility of internal signal states, lack of golden response and irreproducible nature of errors are the major obstacles in post-silicon validation and debugging. Hence, this necessitates a self-checking methodology for debugging complex coherence errors. Typically, global or distributed checkers can be incorporated on-chip to catch CCP errors. However, these checkers can have significant impact on the network traffic which can significantly deteriorate system performance [75].

In this chapter, we propose a hardware-based on-chip logging scheme which captures important signals related to cache coherence mechanism and error detection is performed on-line based on their analysis. Since the high-level specifications of the particular CCP such as MESI [131] are known beforehand, corresponding to an implementation of the protocol, signals required to be observed on-line can be determined. These observed signals are to be analyzed by a detector block which provides the expected (correct) protocol states and other associated signal states. By comparing the correct signal states (based on the protocol specifications) and the signal states of CCP of design-under-validation (DUV), errors can be detected. With the help of an on-chip checkpoint-based rollback and recovery scheme [132], the correct operation of coherence mechanism can be ensured. The proposed hardware structure can assist in the dependable operation of CCP mechanism for both design errors and soft errors. For the case of soft-errors, minor performance degradation is expected to ensure dependable operation as the detector module is deployed in active condition to the customers. However, for design bugs, the logging and error detection module can be disabled after the validation is over causing zero performance degradation to the customers.[2] The only difference in the procedure to ensure correct operations of CCP in case of design bugs and soft errors is that in case of the former, a dedicated bypass path is needed while for the latter, it is not needed as the

[2]Since we utilize a RTL framework evaluation of the proposed technique, the estimation of performance overhead (or, possible performance degradation) is not a trivial exercise.

impact of soft error typically is a transient one. Due to this reason, during the rollback and recovery phase after the soft error is detected, the design can easily resume correct operation. The remainder of the chapter is organized as follows. Section 3.2 presents the preliminaries behind working of directory-based cache coherence protocols. Section 3.3 presents the proposed methodology of on-chip observability and the detection scheme in detail. This section also elaborates on the hardware module which is to be incorporated on-chip for the purpose of validation. Section 3.4 discusses the experimental setup, observations, the results for various type of errors and discussions on them. Section 3.5 presents related work briefly. Different aspects of the proposed methodology are discussed in Section 3.6. Section 3.7 finally concludes the chapter.

3.2 Cache Coherence Protocol (CCP) Preliminaries

In Chip multi-processors (CMP) architecture, memory organization can be either shared or distributed. For better performance, there is a separate cache for every core. There can multiple copies of the same data in the main memory and caches of several cores. If one of the core modifies the data, then other copies of the same data should be updated. Cache coherence mechanism ensures that each core will use valid data, not the old data in case the data is modified.

Simplest form of ensuring cache coherence is through the snooping-based protocol. In this protocol, each core monitors the signals on the shared bus. When a read operation is observed on a shared bus, all processor check whether they have the copy of data in their cache. If they have the data, they have to supply it else main memory will provide the data. When one of the core writes the data in their cache, a signal related to the corresponding operation is broadcast on the bus. The caches which have the same data will update/invalidate their copy. Therefore, the shared bus becomes a severe bottleneck in this protocol. The issue of scalability can be resolved by storing the status of all the cache line in the single storage space called directory. This approach is called directory-based cache coherence mechanism (as introduced in previous subsection). Whenever

access to any memory address arises, the status of that memory block (cache line) is checked in the directory and its gets modified as per the protocol specifications. Based on the available information, we get the data (for read operation) or update the data and directory (after write operation). Figure 3.1 represents a simple directory structure.[3] It can be seen that for every cache line, there is an entry in the directory. Each entry consists of a presence bit and a dirty bit. Presence bit denotes which processors have that cache line. Dirty bit denotes which processor modified the cache line. A different representation of the directory structure is shown in Table 3.1.

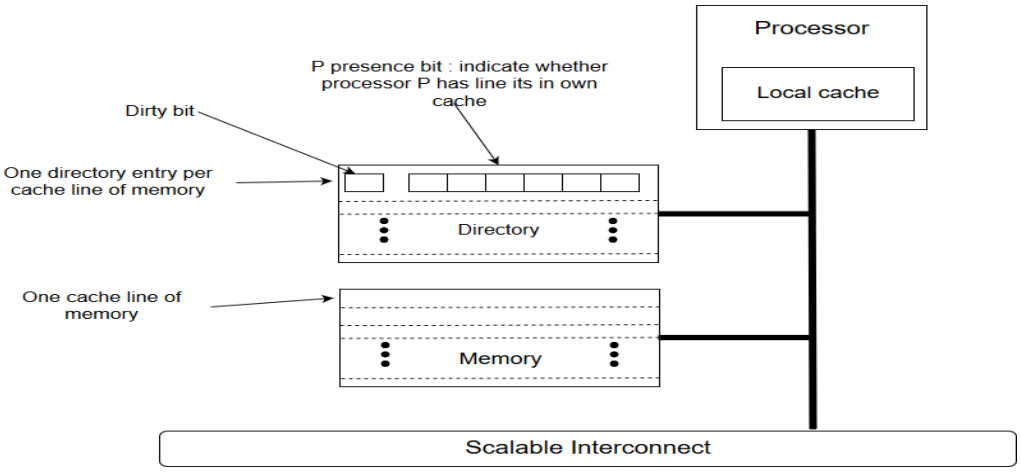

Figure 3.1: Simple Directory Structure [10],[11]

Table 3.1: Directory entry for a cache line in system with N cores [2]

b_{state} (2 bit)	*owner* ($\log_2 Nbit$)	*sharer list* (N bit as per one-hot encoding)

MESI protocol [131] is an invalidation-based cache coherence protocol which has been widely commercialized. It has four stable states. Modified (M) means that one processor core (owner) has data, but it is dirty (memory is out-of-date). Exclusive (E) means that one core (owner) has data, and it is clean (memory is up-to-date). Shared (S) means

[3]This is for the purpose of illustration only. The design we utilized in our experiments has slightly different structure than this.

that data is cached in more than one core and memory is up-to-date. Invalid (I) means that no core has data (i.e., data is not valid in any cache).

There are four types of operations in MESI protocol: *local read* and *local write* that mean core operations is in its cache while *remote read* and *remote write* that mean operation are in the other cache.

1. When there is "read miss" at the cache, the core will load the data and change the state of the cache line to:
   - Exclusive if block is not available in any of the caches:
   - Shared if:
     - block is cached in cache of another core
     - another core modifies the block
     - block is exclusive to one of the remaining core(s)
2. When the operation is "write miss", the core will load the data, and the state is Exclusive. Also, the core will send the message to invalidate the other copies.
3. When the operation is "write hit",
   - if write hit to a Modified block, then there is no change in state
   - if write hit to an Exclusive block, then the state of block changes from Exclusive to Modified
   - if write hit to a Shared block, then the state of block changes from Shared to Modified and the core will send the message to invalidate the other copies.
4. On the eviction of the Modified block, we have to write-back the data

A high-level state transition diagram of MESI CCP is shown in Figure 3.2. In accordance with the terminology adopted in [2], GetS/GetM refers the action to obtain block in Shared/Modified state while PutS/PutM refers the action to evict a cache block in the Shared/Modified state. In other words, GetS and GetM mean read and write permission

respectively. In this representation, "own" indicates the requests (loads/stores) arriving at a cache which have been made by that particular core while "other" indicates the requests originated by other cores of the multi-core system. For some change of cache line states, no coherence transaction is required and hence, those are referred to as "silent".[4]

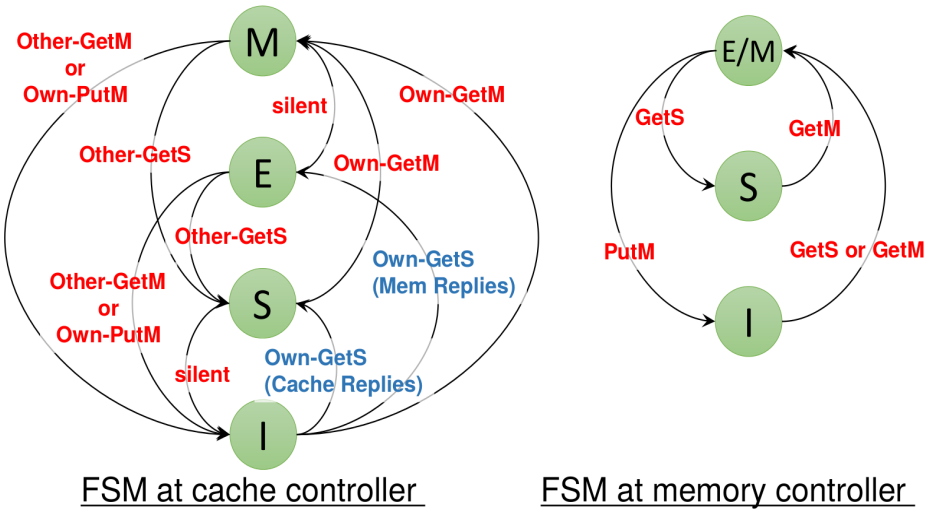

Figure 3.2: High-level FSM of MESI CCP[2, 12]

As per [11] and [133], the actions taken by the directory controller in various conditions can be described as below:

1. *Block is in Uncached (Invalid) state:* copy in memory is the current value. Upon receiving the following requests, the required actions are:

   - *Read miss:* requesting processor sent data from memory & requestor made the only sharing node and state of block made **Shared**.
   - *Write miss:* requesting processor is sent the value & becomes the Sharing node. The block is made **Exclusive** to indicate that the only valid copy is cached. Sharers field in the directory indicates the identity of the owner.

2. *Block is in Shared state:* the memory value is up-to-date. Upon receiving the following requests, the required actions are:

[4]More details can be followed from [2].

- *Read miss:* requesting processor is sent back the data from the memory & the requesting processor is added to the sharing set in the directory entry.
- *Write miss:* requesting processor is sent the data value. All the processors (cores) in the Sharers set are sent invalidate messages, & Sharers is set to the identity of requesting processor. The state of the block is made **Exclusive**.

3. *Block is in Exclusive state:*[5] the current value of the block is held in the cache of the processor identified by the Sharers set in the directory entry.

   - *Read miss:* the owner processor sent data request message, which causes state of block in the cache of owner to transition to **Shared** and causes the owner to send data to the directory and it is written to memory & then sent back to the requesting processor. Identity of the requesting processor is added to Sharers set in the directory, which still contains the identity of the processor that was the owner.
   - *Data write-back:* the owner processor is replacing the block and hence must write it back. This makes the memory copy up-to- date which leads to the block being uncached, and the Sharer set in the directory becomes now empty.
   - *Write miss:* block has a new owner (core-id/processor-id). A message is sent to the old owner causing the cache to send the value of the block to the directory from which it is sent to the requesting processor. Because of this, the requesting processor becomes the new owner. Sharers field in the directory is set to the identity of new owner, and state of block is made **Exclusive**.

3.3 Proposed Methodology of CCP Validation

Post-silicon validation and debugging of complex functionalities such as the correctness of CCP implementation requires addressing two important issues:

[5]In Figure 3.2, Exclusive and Modified states are merged as one at the directory controller (maintained at the main memory).

- What type of internal signals are to be observed?
- How the correctness of the implementation can be ascertained in the absence of golden responses?

An implementation of a complex CCP in a modern processor utilizing an effective on-chip routing system would have thousands of internal signals. Clearly, identifying error detection related useful signals out of thousands from the design description is not trivial. Hence, we formulate the identification problem based on invariant-related analysis and then extend this analysis for devising an on-chip error detection scheme.

3.3.1 Basic Premise

For ensuring correct operation of a CCP implementation, the authors in [2] describe the most important property, **SWMR invariant** as *For any given memory location, at any given moment in time, there is either a single core that may write it or certain number of cores that may read it. Thus, at any time, it does not occur that a given memory location may be written by one core and simultaneously either read or written by any other cores.* Hence, the name of this invariant is given as single writer, multiple readers (SWMR).[6] Therefore, for checking correctness, we need to observe the actions of the single writer (particular core-id) along with multiple readers (respective core-ids) for assisting in the design validation/debugging process. The overview of this methodology is presented in Figure 3.3. The exercise to derive the signals to be observed on-chip is manual in nature. We mostly relied on the detailed explanation of directory-based CCP in [2] and [133]. Once the type of signals are identified, they need to be selected from the RTL design available from [134]. The invariants [2] merely act as a guiding principle behind the identification of RTL signals.

It is difficult to ensure CCP implementation correctness by merely observing events at one or more caches. We also need to observe the different requests coming from

[6]The other coherence invariant from [2] is that of Data-value integrity. As per it, the value of the memory location at the start of an epoch is the same as the value of the memory location at the end of its last read-write epoch.

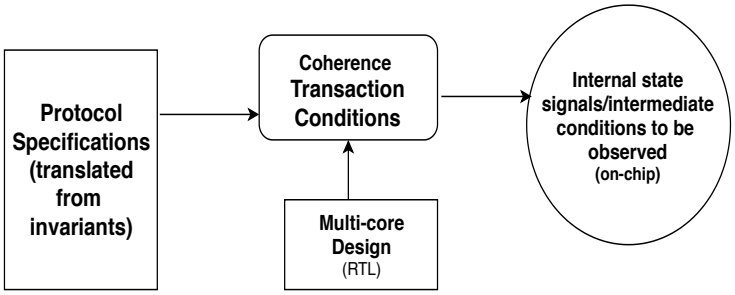

Figure 3.3: Deriving on-chip logging scheme

memory and the requests to be processed from one cache to another after passing through the on-chip network. Typically, in any CCP implementation, there is a finite state machine (FSM) at controller that has few stable states and many transient states. As the number of cores increase, the number of transient states becomes large and the possible number of interactions among them becomes intractable. For observing whether a CCP implementation conforms to the SWMR invariant or not, it is obvious that we need to observe signals related to the CCP finite state machine. Depending on whether the CCP implementation is directory-based or snoopy-based, we have to decide other signals for observing. Suppose, we consider a directory-based CCP wherein the information regarding state of cache lines is maintained in a directory. Typically, the directory is distributed across the CMP system. Hence, each core has a slice of the directory. Each entry of such a directory slice consists of at least 3 fields- status of memory block, owner-id, list of sharer nodes (cores). Therefore, to obtain an on-chip checker module which can detect illegal coherence transactions, it would be beneficial to observe the actions of the directory such as whether read/write is being performed in it or memory update is getting performed, etc. Based on these observations, for a given CCP implementation, we derived that following type of signals should be logged (on-chip):

- *state of memory block belonging to individual caches*: for knowing the current state of the CCP finite state machine (FSM) for a memory block, we need to observe the **state** of that particular block in the directory.

- *status of directory controller*: current action performed by the directory controller (which is present along with memory controller) is to observed for checking what

kinds of transactions are allowed corresponding to each action. Note that for snooping-based implementation, broadcast messages (on the bus) should be observed.

- *incoming coherence messages*: various coherence messages (incoming in the memory sub-system) i.e., invalidation requests, write/read requests by cores etc. sent by the processor core need to be tracked.

- *current coherence messages*: various coherence messages (present in the memory sub-system) i.e., invalidation responses etc. received by the processor cores need to be tracked for finding the next valid cache line state.

- *tracking of sharer requests*: for sharing of cache lines, various requests are generally transacted between cache and memory. So, we need to observe the sharer-id (i.e., processor/core-id etc.) and related messages.

- *tracking of sharing relations*: for checking legality/validity of coherence transactions (i.e., address matching between requester and sender caches, sharing conditions set true/not, etc.) and some miscellaneous messages between the caches to memory need to be observed.

3.3.2 Description of Logging Structure & Detection Mechanism

Table 3.2 shows the on-chip logging scheme (i.e., the group of signals to be observed) in the proposed methodology. Out of these observable signals at any T^{th} clock cycle, the first three serve as signal states to be compared with expected (correct) signal states within the detector module (and thus are stored in logging register at $(T+1)^{th}$ cycle, shown in Figure 3.4) and remaining three are validity conditions which assist in fully checking the generation of a valid/legal coherence transaction output. Essentially, we call them so to differentiate between the signal which constitutes coherence transaction output and signal conditions which are reasons behind coherence transactions (i.e., state transitions and other status changes). The selection of the entries in Table 3.2 is particularly aimed

at the maximization of design/electrical bug detection and subsequent bug localization.

Table 3.2: Signals to be observed

b_{state}	m_{status}	cur_{msg}	in_{msg}	$share_{req}$	$sharer_{cond}$

In the above observability scheme, b_{state} depicts state of the memory block (or, equivalently each cache line) which is maintained at the directory, m_{status} denotes the state[7] of directory action being performed (by directory/memory controller), cur_{msg} denote the current coherence message being processed, in_{msg} represent the incoming coherence message from the memory sub-system network, $share_{req}$ depict the sharers (cache id's) and $sharer_{cond}$ stands for checking various address matches and conditions on which the cache coherence transactions are true. Typical CCP implementation enforces that at any given time, b_{state} depends on previous b_{state} and m_{status}. Additionally, to minimize detection latency m_{status} helps as the errors can be detected early in the form of mismatch between directory actions, without waiting for a mismatch between b_{state} values. The proposed detection scheme is shown in Figure 3.4. The detector module is to be incorporated for each core. On a per-core basis, the error detection is performed and then rollback and recovery option can be activated. Note that the rollback and recovery scheme is labeled optional because in the case of design error validation, this is not needed. If it is intended to provide dependable operation in the presence of design errors, then this scheme is required along with bypass path mechanism to avoid re-occurrence of the design errors. Alternatively, during the validation of first silicon, after error detection, some limited trace (in the form of contents of logging register shown in Figure 3.4 stored into on-chip trace buffers with a fixed width and depth) and then they can be dumped off-line for bug localization. In the case of soft-errors (due to transient effects), to ensure dependable operation after customer shipment, an effective rollback and recovery scheme is definitely needed.

[7]We deliberately name it as m_{status} to avoid confusion with b_{state} as both of them belong to different components in the system although both of them point to states of finite-state machines (FSMs). Specifically, b_{state} points to the cache line state and m_{status} points to the state of the memory controller.

The detection mechanism (shown in Figure 3.4) contains a logging register, validity condition matcher, comparator and a combinational circuit which implements the coherence transaction output logic. Generally, for checking cache coherence transactions, we need to match the tuple of {Current state, Request/Message, Response} between the CCP specification and the respective implementation. Consider the requests arriving from the **core and memory sub-system** in clock cycle T which are input to the **detector module**. Depending on CCP specifications, the expected signal states are computed corresponding to $(T+1)^{th}$ clock cycle which are stored in a register (say, $lr1$) internal to the **detector module** (and not explicitly depicted in Figure 3.4). Essentially, this computation is performed by a combinational logic block based on the signal states observed at T^{th} clock cycle. These computed signal states/values are compared with the contents of **logging register** (say, $lr2$) which have logged these signal states from **core and memory sub-system** at $(T+1)^{th}$ clock cycle. So, the contents of $lr1$ and $lr2$ are to be matched for detecting the error. This detection mechanism is intended at minimizing error detection latency for different kinds of errors(design errors/soft errors). Note that signal values of $lr1$ are computed in accordance with the conditions of cur_{msg}, $share_{req}$ and $sharer_{cond}$. Therefore, these three fields of Table 3.2 act as validity conditions. Specifically, both $lr1$ and $lr2$ contain b_{state}, m_{status} and cur_{msg} of $(T+1)^{th}$ clock cycle. Comparing cur_{msg} assists in detecting errors related to the request/message response generation logic. This is because cur_{msg} at $(T+1)^{th}$ clock cycle serves as the response of the requests which may have arrived at the memory sub-system till T^{th} clock cycle. The high-level working of the detector module is described as Algorithm 1. The validity conditions (in the form of *incoming coherence messages*, *sharer requests* and *sharing conditions*) serve as important part of the invariant checking and comparing the contents of $lr1$ and $lr2$.

As explained earlier, the widely popular MESI protocol [131] has four stable states-Modified (M), Exclusive (E), Shared (S) and Invalid (I). Transitions from any state s of the cache line to the state t must proceed only as per the specifications of MESI CCP. However, having dedicated checkers for these state transitions for each cache line

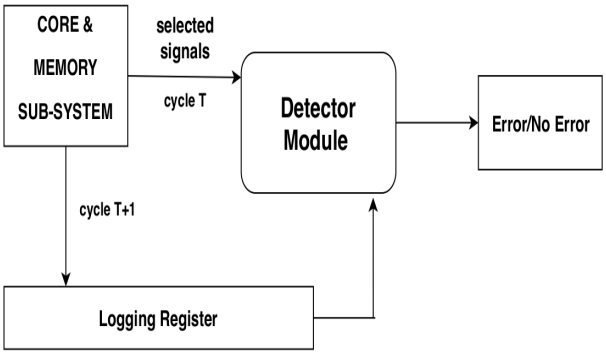

Figure 3.4: High-level overview of proposed error detector module incorporation into multi-core design

is not an easily affordable solution. The proposed design of the detector module is intended to minimize network traffic overhead compared to the techniques which employ distributed/global checkers for CCP verification. Note that in the proposed technique, checking is confined inside the module itself, so, there is no injection of traffic into the on-chip network for the purpose of error detection.

The idea behind the check condition ($cur'_{msg} == in_{msg}$) is that in_{msg} of T^{th} cycle becomes cur_{msg} of $(T+1)^{th}$ cycle. In absence of any error, cur'_{msg} (the expected message) equals the previous incoming message (in_{msg}). Note that merely checking for the state transitions between a pair of cache lines may not assist in the detection of errors related to the directory or the logic components related to the generation of proper requests/messages. This is the reason behind observing the different signal types listed in Table 3.2. For instance, for the 7-stage pipeline processor design in our experiments, the memory controller (which incorporates the directory controller also) implementation has 11 states (depicted by m_{status}) such as IDLE, WAIT, DONE_READ, DONE_WRITE, UPDATE, UPDATE_DONE, RE_TRY and so on.[8] At each of these directory states, depending on in_{msg} and cur_{msg}, cache line states (b_{state}) changes or remains in the previous b_{state}. Therefore, one of the important features in the proposed approach (Algorithm

[8]This would definitely vary from one particular implementation to the other. The name of different states of this controller is self-explanatory. The high-level specifications typically do not engage such detailed states of the memory controller (implementing the directory functionality). However, from RTL, we can know about these states of the memory controller. We assume that trivial errors such as a missing state in memory controller have already been detected at pre-silicon verification stage.

Algorithm 1: *DetectionModule*

Input: $\{b_{state}, m_{status}, in_{msg}, cur_{msg}, share_{req}, sharer_{cond}$ (at T^{th} clock cycle)$\}$, $\{b'_{state}, m'_{status}, cur'_{msg}$ (at $(T+1)^{th}$ clock cycle)$\}$

Output: *OutSig*

```
1  for each item of m_status do
2      for each item of in_msg do
3          Match in_msg, share_req and sharer_cond to their validity conditions;
4          Compute m1_status, b1_state corresponding to (T+1)^th clock cycle);
5          if cur'_msg == in_msg && m1_status == m'_status then
6              if b1_state == b'_state then
7                  OutSig ← NoError;
8              else
9                  OutSig ← Error;
10             end
11         else
12             OutSig ← Error;
13         end
14     end
15 end
```

1) is that we are observing memory controller states in addition to the signals observed only in the cache coherence mechanisms. Because of this, the design bug coverage is expected to be higher than the techniques which target error detection only with the signals involved in CCP mechanism operation.

3.3.3 Deriving Validity Conditions Inside Detector Module

For checking of CCP implementation, we need to maintain a list of sharers (i.e., processor cores/caches which are sharing memory blocks). The error detection module has to utilize this list which is maintained and updated at the directory. We need to check for a given memory block, if the particular request is coming from a core (i.e., it's core-id) is equal to the core-ids present in the list of sharers. Further, the address of the cache line in any one particular state (say, shared) is to be compared with the memory address for which requests are currently arriving. Thereafter, we can check if the received responses are proper or not. It may happen that the only sharer might be the requestor for data, in that case no invalidation messages are sent. Therefore, for bug detection, merely checking

invalidations (in the form of cur_{msg}) would lead to false positives. Specifically, following checks (although not exhaustive) can serve as validity conditions for the analysis of a directory-based MESI CCP implementation:

- we need to check if the requests ($sharer_{req}$) for a memory block arriving at the directory controller is already present in the list of sharers in the directory. Depending on this checking, expected (i.e., correct) memory block status (b_{state}) can be computed by detector module.

- we must check if new requests (in_{msg}) are arriving for an address which is currently being processed. While waiting for actions corresponding to one address, the directory needs to respond to other addresses as well.

- since the directory sizes are limited, a large number of sharers can not be stored in it. The position of sharers ($sharer_{cond}$) need to be checked as when a directory entry cannot handle all sharers, other sharers are evicted.

With the help of above checking conditions, the detector module is able to differentiate between the large number of possible coherence related events at the directory coherence controller. Furthermore, it is difficult to specify cache line state transitions without the above validity checking conditions. Specifically, following five conditions are identified for the error detection as per Algorithm 1 (here, sharer[pos] indicates the list of sharers which is maintained at the directory):

- $sharer_{cond1}$: check if sharer[pos] $\neq$ $core_{id}$: when sharer[pos] is not equal to $core_{id}$, it means data is available in other core, the expected b_{state} is shared, however, if this condition is equal, data is in the same core and the expected b_{state} is modified

- $sharer_{cond2}$: check if sharer[pos] = $core_{id}$: when sharer[pos] is not equal to $core_{id}$, it means that same core is modifying the data, the expected b_{state} is modified

- $sharer_{cond3}$: check if $in_{address}$ for the incoming request equals the $address$ of the memory block which is shared

- $sharer_{cond4}$: check if sharing condition is true for the cores (which are the potential sharers) in the directory entry

- $sharer_{cond5}$: check if core(s) originating a coherence transaction request is present in the sharer list

The major conditions inside the $DetectionModule$ (written after the analysis of RTL description and with high-level specifications) are shown in Tables 3.3 and 3.4. These tables are deliberately divided into two parts to provide better clarity on the change of different signals with changing conditions. Specifically, the message names (in_{msg}) are obtained from the analysis of multi-core design RTL.[9]. For the sake of brevity, only the important messages in_{msg}/cur_{msg}, state/status transitions (m_{status}/b_{state}) and tracking conditions ($sharer_{cond}$) are depicted.[10]

3.3.4 Coverage Issues Under Proposed Methodology

The proposed methodology ensures that the tuple of {Current state, Request/Message, Response} between the CCP specification and the respective implementation is checked continuously till the completion of test program. There is a close relationship between the signals observed (i.e., {Current state, Request/Message, Response} and validity conditions), the detector logic and the design bug detection coverage. The same is true for soft errors too. However, because of overhead issues, we have not covered the network/router errors. As stated previously, if the error in the communication network results into manifestation at the directory or the cache controller FSM, then the proposed technique can detect those error(s). We strongly believe that the design bug/soft error coverage

[9]It is assumed that the corresponding message names are properly written in the RTL design description. Most of these message names are self-explanatory. For instance, REQ indicates request, INV indicates invalidation, WB indicates write-back, M indicates memory, W indicates write, R indicates read, and so on.

[10]A vacant entry in these tables means that the respective entries are not important for the working of detector module for the checking of corresponding transitions of m_{status}/b_{state}. This is because the respective entries remain same as the previous transaction and no change is observed for those fields. In other words, the transition of m_{status} to m'_{status} or b_{state} to b'_{state} does not depend on those entries/fields. Also, completely specifying the entries in this table can lead to additional synthesized gates in the resulting on-chip hardware.

Table 3.3: *DetectionModule* Conditions-1

m_{status}	b_{state}	$share_{req}$	in_{msg}	$sharer_{cond}$	b'_{state}
READ_DIR	INVALID				EXCLUSIVE
READ_DIR	SHARED				SHARED
READ_DIR	SHARED				SHARED
READ_DIR	EXCLUSIVE				EXCLUSIVE
READ_DIR	MODIFIED				MODIFIED
WRITE_DIR	INVALID				MODIFIED
WRITE_DIR	SHARED				SHARED
WRITE_DIR	SHARED				MODIFIED
WRITE_DIR	EXCLUSIVE				MODIFIED
WRITE_DIR	EXCLUSIVE				EXCLUSIVE
WRITE_DIR	MODIFIED				MODIFIED
WAIT	EXCLUSIVE		INV_RESP	(sharer[u] == src_id) & v_sharers[u]	INVALID
WAIT		shari-ng(v_sharers)	INV_RESP	”	SHARED
WAIT		!(shari-ng(v_sharers))	INV_RESP	”	INVALID
WAIT			SH_RESP	”	SHARED
WAIT			WB_RESP	”	SHARED
WAIT			WB_INV_RESP	”	INVALID

also depends on the quality of tests applied. Since, the design is a multi-core system, we apply test (assembly programs) which have shared memory addresses among them. Specifically, the tests could contain only load and store instructions, however, in our experiments, we utilized tests containing all instructions of the ISA. The exact analysis of design bug/soft error coverage and their dependence on test program quality is a part of our future work. As stated earlier, the dumped traces (i.e., contents of logging register, $lr2$) can be analyzed off-line for bug localization through invariant-based checking for CCP specifications. For achieving quick dumping of these logged contents, a hashing mechanism can be implemented. A content addressable memory can also be utilized for the storage of these contents.

3.3.5 Test Programs for RTL Simulation

For triggering coherence related errors, we need multi-core programs (i.e., parallel programs) with the condition that these programs contain shared memory addresses. Figure

Table 3.4: *DetectionModule* Conditions-2

m_{status}	b_{state}	$sharer_{cond}$	in_{msg}	cur_{msg}	m'_{status}
IDLE			R_REQ		READ
IDLE			W_REQ		WRITE
IDLE			M_WB_REQ		READ
IDLE			INV_RESP		UPDATE
IDLE			WB_INV_RESP		UPDATE
RE_TRY				R_REQ	READ
RE_TRY				W_REQ	WRITE
RE_TRY				M_WB_REQ	READ
READ_DIR	INVALID				DONE_READ
READ_DIR	SHARED				DONE_READ
READ_DIR	SHARED				WAIT
READ_DIR	EXCLUSIVE				WAIT
READ_DIR	MODIFIED				WAIT
WRITE_DIR	INVALID				DONE_WRITE
WRITE_DIR	SHARED	sharer[pos] != pe_id			WAIT
WRITE_DIR	SHARED	sharer[pos] = pe_id			DONE_WRITE
WRITE_DIR	EXCLUSIVE	sharer[0] == pe_id			DONE_WRITE
WRITE_DIR	EXCLUSIVE	sharer[0] != pe_id			WAIT
WRITE_DIR	MODIFIED				WAIT
WAIT	EXCLUSIVE		INV_RESP	check for in_address == address	RE_TRY

3.5 shows the procedure adopted for converting the parallel programs into Hex file which is utilized with ModelSim simulator for the multi-core RTL simulation (using .tcl scripts which have commands for loading the individual test program Hex codes on to the I-caches of individual cores). We utilize the tool-chain available at [134] for carrying out this conversion. We utilized the multi-core Fibonacci series computation and matrix multiplications programs which are supplied with the software tool-chain from [134]. These programs have all kinds of instructions as per MIPS ISA. However, most important instructions which are concerned with coherence mechanism validation are loads and stores. Therefore, in our random constraints-based test programs, we have mix of load and store instructions with few addresses shared among all the cores. It is obvious that higher the sharing of memory addresses between the cores, better is the quality of the test programs.

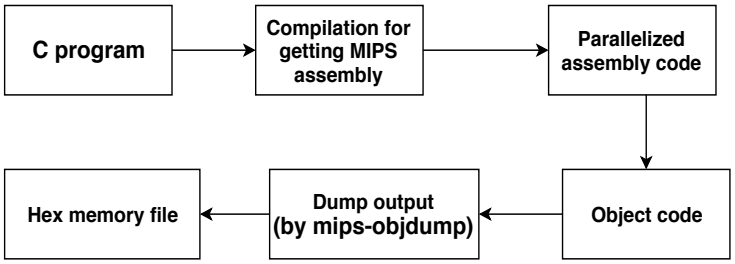

Figure 3.5: Steps involved in Hex file generation (for multi-core RTL simulation)

3.4 Experimental Setup, Observation & Results

3.4.1 Experimental Setup Details

We considered the open-source multi-core MIPS processor implementation [13] for our experiments. The design consists of a seven-stage pipeline processor and utilizes directory version of MESI CCP. We performed experiments with a 4 core version of the design in [13]. The memory organization in this design is of cache coherent non-uniform memory access (ccNUMA) type. Each processor core has its own local cache and memory. Specifically, we consider a 4 core processor with each core having ccNUMA main memory. The caches are split into I-cache and D-cache.

3.4.2 Overview of Multi-core Design Framework

For better understanding of the multi-core design, an expanded view of the memory sub-system architecture from [13] is shown in Figures 3.6 and 3.7. The left side expansion in this figure shows the local memory structure (of the NUMA setup) and the right side expansion shows the routing structure of the system. Figure 3.7 shows the detailed view of the memory sub-system architecture. The Address Resolution Logic works with the Packetizer module to facilitate the interaction of the caches and the local memory of the rest of the system. All cache traffic goes through the Address Resolution Logic, which determines if a request can be served at the local memory, or if the request needs to be sent over the network [13]. The Packetizer is mainly responsible for converting data traffic (such as a load) coming from the local memory and the cache system into packets or flits that can be routed inside the Network-on-chip (NoC), and for reconstructing

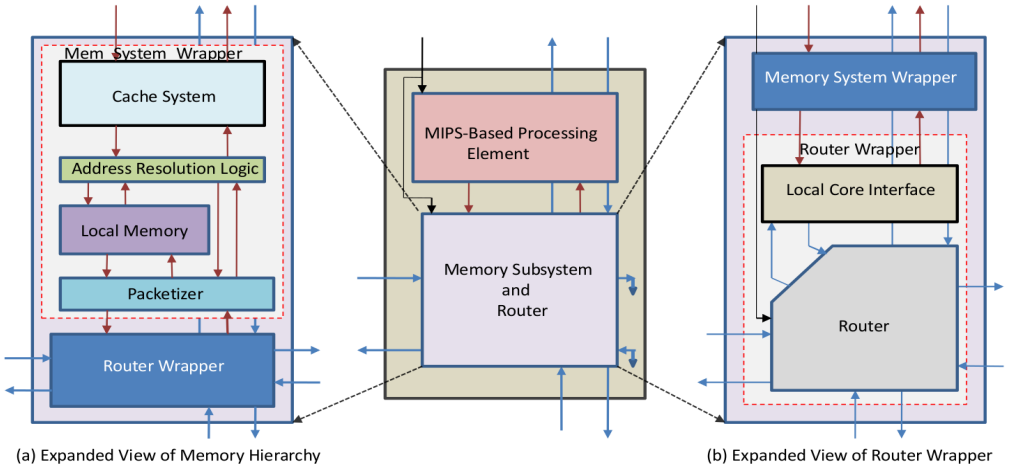

Figure 3.6: Memory sub-system architecture [13] utilized in experiments

packets or flits into the data traffic at the opposite side when getting out of the NoC.

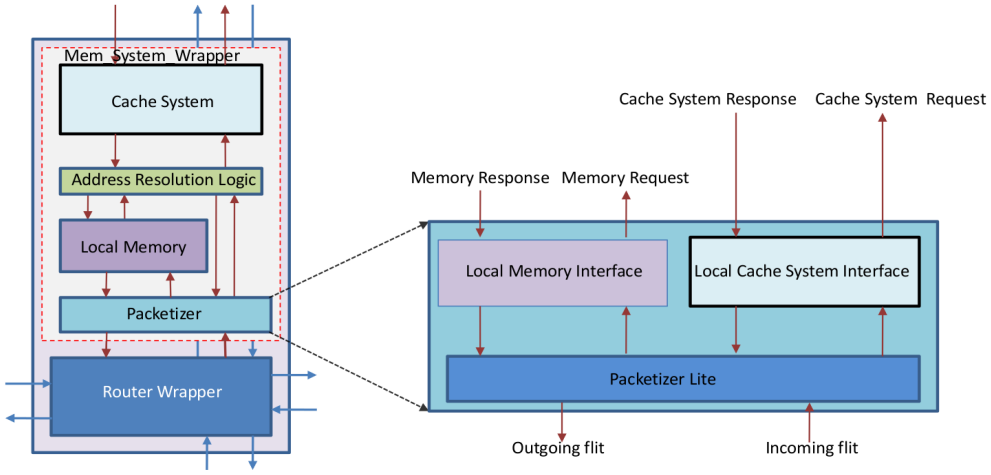

Figure 3.7: Detailed view of memory sub-system architecture [13]

By using the approach shown in Figure 3.3, we identified 23 bits for Table 3.2, out of which 5 bits depict the presence/absence of validity conditions ($sharer_{cond}$). For the structures elaborated in Section 3.3.2, $lr1$ and $lr2$ are of 10 bits only (m_{status} is of 4 bits, cur_{msg} is of 4 bits and because of only 4 stable states in MESI, b_{state} is of 2 bits). Similarly, the messages, in_{msg} and $sharer_{req}$ are each of 4 bits. For the purpose of comparative evaluation, we repeated the error injection experiments and analyzed their detection with a previous method [3] proposed in the literature. The technique in [3] involves the incorporation of an additional core into the multi-core system for computation of the correct signal transitions and protocol states when the related coherence messages

(identical to the observed signals in our approach, b_{state} and in_{msg} as shown in Table 3.2) are received as described in Algorithm 2.

3.4.3 RTL Implementation Analysis of Proposed Technique

We synthesized the RTL implementation with Synopsys Design compiler tool utilizing a 32-nm standard library (Synopsys educational library). The overhead results for each core are shown in Table 3.5. The second column shows the numbers for the original implementation which includes the 7-stage pipeline core (Instruction Fetch1, Instruction Fetch2, Instruction Decode, Execute, Memory Access1, Memory Access2 and Write-back)[11] and the third column depicts the numbers for the proposed detector scheme. The fourth column indicates the overhead percentages for the proposed detection module.

Table 3.5: Overhead Analysis

	Core	Prop.	Ov.(%)	mFSM	Ov.(%)
Area(um^2)	39158.80	202.07	0.51	894.91	2.28
Power(uW)	714.75	5.05	0.70	22.65	3.16

We also implemented a duplicate memory controller finite state machine which is mainly responsible for coherence transaction output (i.e., next m_{status}, cur_{msg}) generation at the directory to mimic double modular redundancy error (DMR) detection scheme. We assume that the second (i.e., the duplicated) FSM is design bug free and hence capable of bug detection. For soft errors, the additional FSM (denoted by mFSM) can assist in error detection under the conditions that the soft error impacts only the actual memory controller.[12] The fifth column shows the numbers for this additional FSM. The overhead percentages due to this extra FSM is reported in the sixth column. Total power (static + dynamic) is reported in the third row of Table 3.5. Compared to DMR, the proposed technique has lesser overheads. It is obvious that as the size of core increases, the overhead numbers would proportionately reduce.

[11]The processor is a single-issue in-order MIPS core, fully bypassed, no branch prediction or branch delay slot, running MIPS-III instruction set architecture (ISA) without floating point support.

[12]In a strict sense, for design errors, this FSM would not be useful as the same design error may be present in it also.

3.4.4 Comparative Evaluation with Literature

Among various approaches to tackle coherence mechanism validation proposed in the literature, the work closest to our approach is that of Rodrigues et al.[3]. The authors have proposed a special-purpose diminutive core (called as Sentry Core shown in Figure 3.6) which takes as inputs the current messages and coherence transactions and produces the expected state transitions as output. The method is described in [3] for a snoopy protocol system to detect soft-errors. We implemented this detection technique for a directory-based CCP.

Table 3.6: Previous approach for checking multi-processor coherence [3]

	Core1	Core1	Core2	Core2	...	...		
Addr	*Curr. state*	*Exp. state*	*Curr. state*	*Exp. state*	*...*	*...*	*Mem. op*	*Reques. ID*
A	E	S	-	-	..	..	write	1
..	..	..	..	..	..	..	..	..

The high-level working of the approach in [3] is shown as Algorithm 2. Note that this is the modified version of the technique in [3] for a directory implementation, unlike the broadcast-based snooping approach in the original proposal. We have performed experimental studies for the technique in [3] and the proposed methodology for both design errors and soft errors (bit flips).

Overhead Analysis: A major reason behind the area overhead in [3] is because of *CenLogStruct* structure in the alternate core (called as *SC* in [3]). From Figure 3.6, the bits required are : Address (16 bits), Current & expected states (4 bits for one core totaling as 16 bits for 4 cores), *Mem. op* (2 bits) and *Reques. ID* (2 bits). Therefore, if we consider length of *CenLogStruct* as 2K, the total requirement is 36 bits for each line in 2K cache (which is basically the local cache of the alternate diminutive core, *SC*). Additionally, *SC* has logic for generation of the *Exp. state* for each coherence transaction of each core and the address tag search hardware (which is part of the cache structure in *SC*). Therefore, the total overhead in case of [3] is expected to be more than the combined overhead for the proposed hardware (i.e., considering the detector module

Algorithm 2: *SentryCoreApproach*

Input: {b_{state} of each core, Address of $CacheLine$, $in_{msg}(Mem.\ op, Reques.\ ID)$

Output: $OutSig$

```
1  Store b_state of each core, Address of CacheLine, in_msg(Mem. op, Reques. ID in
     CenLogStruct (SC Cache) ;
2  for each b_state entry in each core do
3  |  Compute expected state (b^e_state) and store in CenLogStruct ;
4  end
5  for each coherence transaction in each core do
6  |  lAdd ← Search for Address of CacheLine in CenLogStruct ;
7  |  b'_state ← observed coherence transaction of the core ;
8  |  if lAdd matches && b^e_state == b'_state then
9  |  |  OutSig ← NoError;
10 |  else
11 |  |  OutSig ← Error;
12 |  end
13 end
```

overhead of all the cores.)[13]

3.4.4.1 Design Bug Injection Study

We injected design bugs as per the case studies reported in previous literature, particularly [39, 85] along with other scenarios. These error injection cases are reported in Table 3.7. ND denotes that error could not be detected by the respective technique. All simulations were done for a duration of 0.1 to 1 million clock cycles[14] for either random assembly tests or C applications. The numbers reported in Tables 3.7 and 3.8 are reported for simulation for 0.2 million clock cycles of the application for Fibonacci series

[13]We did not implement the complete cache structure as described in [3]. It is definitely true that the described cache structure (as in [3]), would incur less area overheads than our implementation of *SC* structure (for each core) because the alternate core utilizes the same address search mechanism (as in the other caches of the cores) for comparison tasks and other associated logic circuitry. However, based on our observations of RTL implementation of a separate module (considering the length of *CenLogStruct* as 2K) representing the alternate diminutive core (*SC*), we speculate that the technique in [3] would incur at least 10 to 50 times more area overhead (in overall terms for all the cores in the complete system) than the proposed technique.

[14]It is true that such test programs can be easily run during pre-silicon verification, however, during post-silicon validation, we can target electrical effects which can not be simulated accurately during the pre-silicon verification stage.

computation. We consider first 0.05 million cycles for initialization of the design. Thereafter, the error detection analysis is performed for the proposed technique as well as the methodology in [3]. Note that these design bugs can occur because of any reason or combination of reasons/effects. The second column in Table 3.7 shows the manifestation of the particular error. This reasoning is true for electrical error scenarios illustrated in Table 3.8. The errors are injected at the directory controller (or, the memory controller), cache controller and the interface between cache and the local memory (i.e., main memory in ccNUMA configuration).

Table 3.7: Comparative evaluation of error detection cycles for design bugs

Design Bug Name	Description	Cycle	Cycle[3]
Memory block state	state of memory block is stuck-at shared state	146	2995
Memory out message	out message is stuck-at memory request in directory	127	2243
Deadlock condition 1	directory state machine is stuck-at at a state	2798	23038
Deadlock condition 2	state of cache line becomes invalid in one state of directory controller	149	3715
Cache to memory invalid	cache2mem_invalid signal is set to zero	269	5426
L1 Store race	L1 transitions from S to M while another cache issues a store	3216	ND
L1 WBack + load	L1 evicting dirty data while another cache issues a load	5390	35830
Delayed-message	cache to memory messages	6042	ND
Dropped-message	out_msg from main memory	6096	42560
Delayed-writes	messages for store operations	3323	28266

As is shown in third and fourth columns of Table 3.7, the proposed technique achieves much lower error detection latency (shown in the third column) compared to [3]. This is because the technique in [3] employs a logging structure for observing important signals

for a fixed number of clock cycles. In our implementation, we have considered the depth of this logging structure (*CenLogStruct*) as 2000 clock cycles. Since this logging is done for per-address basis, for every checking of a coherence transaction involving a memory address, the methodology involves search of the the logging structure until an address match is found, and hence it can require a maximum of 2000 cycles for that purpose. Thus, the overall detection latency (which requires searching for a number of memory addresses) increases significantly compared to the proposed technique. Additionally, it can be observed that for two design bugs, the technique in [3] fails to detect them (which are denoted by ND).

3.4.4.2 Soft Error (Electrical Error) Injection Study

To measure the efficacy of the proposed scheme for soft-error scenarios, we consider few case studies in Table 3.8. Note that these bit-flip errors can act as electrical errors (due to different design bug issues), too, which are very difficult to debug.[15] The clock cycle at which the flip is injected is shown in the second column. The third and fourth column

Table 3.8: Comparative evaluation of error detection cycles for soft errors

Flipped Signal(s)	Injection	Cycle	Cycle[3]	Reduction
directory controller state	4997	4998	33679	6.73X
out_msg in translator	3999	4001	29683	7.41X
cache2mem_rq_msg in translator	3000	3002	24022	8.00X
cache2mem_rq_data in translator	5000	ND	ND	–
translator state and mem2cache	6000	6067	39350	6.48X
network2mem_msg	2500	6462	ND	–
cache2network_msg	3000	3522	32869	9.33X
local2cache_msg	6000	10424	40113	3.84X
from_core_empty and to_core_empty in packetizer	1750	ND	ND	–
mem2network_data	2000	ND	ND	–

denote the clock cycle in which the error is detected by the proposed methodology and

[15]Therefore, we do not target here dependable operation of the design. Instead, we are aiming only at the detection of a failure arising out of such scenarios. It is assumed that the bit-flip is not getting masked and therefore has the potential of creating functional failures during the design execution.

technique in [3]. Similar to the case of design bugs, lower error detection latency is achieved with the proposed technique. There is a significant difference of cycles between both the methods. This is because the technique in [3] relies on detecting the error (i.e., bit-flip) manifestation in the protocol state only. Contrast to this, in the proposed methodology, since more signals (Table 3.2) are covered, we succeed in detecting the bugs close to their origin in these cases (i.e., different messages/requests). Additionally, the technique in [3] needs complete search of the logging structure until an address match is found, as already explained previously. Note that the proposed methodology succeeds in detecting only seven out of ten cases, while the technique of [3] succeeds in six. These cases are linked to the errors at components (or at respective messages/requests) which are not covered under the proposed observability mechanism.

3.5 Discussion on Previous Methods

A study of CCP verification techniques (particularly, through formal methods) is presented[16] in [73]. These techniques involve model checking of coherence protocols for different properties through reachability analysis. The model checking exercise is capable of exposing subtle errors in the design of the protocol. To address scalability issues, parameterized verification of CCP has been explored in [135]. On similar lines, a design methodology to ease out verification efforts has been proposed by Zhang et al. in [41]. However, this scheme requires significant changes in the widely used CCPs thereby hindering its application. Test generation for multi-core system for simulation-based verification has been proposed in [136]. Effective test programs have the potential to discover bugs at the pre-silicon verification stage. The technique in [40] proposed a simulation-based method for abstracted CCP finite state machine. However, carrying out such abstraction of a complex CCP implementation is not trivial. Typically, the verification of complete cache coherent system (i.e., all components of a CCP implementation) is more difficult than the verification of only the protocol or the memory controller [7]. A

[16]Previous methods of CCP validation are already discussed in Section 2.1 of Chapter 2.

Table 3.9: Comparative evaluation with other runtime/post-silicon CCP validation methods

	Methodology/ Technique	Merits	Drawbacks
[74]	simplified version of CCP for checking	succeeds in covering some bus transactions at cost of overhead	additional network messages need to be transmitted
[39]	string matching for consistency between state of L1 & L2 cache blocks	minimal area overhead (<0.005%)	not suitable for single level cache hierarchy; network traffic overhead ↑
[85]	bug-pattern matching & distributed event detectors	low performance penalty	not useful for new bug scenarios (bug-patterns are not known)
[75]	utilizing watchdog processors, put checker for each core	low area & performance overheads	not scalable; poor bug/fault coverage as only stable cache line states can be checked
[3]	an alternate core for correct coherence transaction computation	low performance overhead	high error detection latency for searching cache of the alternate core
[137]	a fault-tolerant directory to handle wrong/loss of messages	capable of handling network related errors also	needs new message types network traffic overhead ↑
[138]	signature-based checking by verifiers at every cache and memory controller	capable of detecting global conflicts in coherence transactions	not scalable as a hierarchy of verifiers is needed
Prop.	a detector & logging scheme for each core to predict correct signal states	low area overhead, applicable for complex directory-based CCP	fails to detect network related errors

qualitative comparison of the proposed technique with previous methods (dealing with the complete verification/validation of a cache coherent system) from the literature is presented in Table 3.9 where we present their merits and demerits. We strongly believe that the proposed methodology can be applied in collaboration with some of these techniques. It is worth to note that all of the techniques in Table 3.9 are implemented at the level of architectural simulation (high level system representation) whereas the proposed technique has been evaluated within RTL framework. This makes a direct comparison of proposed methodology with all the techniques in Table 3.9 extremely difficult.

3.6 Applicability to Other System Configurations

3.6.1 Variation with Number of Cores

Since the proposed technique assumes one detector module per core, there is no alteration in the scheme even when the number of cores are changed. As stated previously, the multi-core design utilized in our experiments adheres to a non-uniform memory configuration (ccNUMA), with the addition of each core (and its memory), the corresponding detector needs to be incorporated. For a shared memory configuration also, similar modification needs to be applied.

3.6.2 Variation with Cache/Memory Organization

For a multi-level cache organization (for instance- local $L1$ and shared $L2$ caches), the detector module can be incorporated at the memory controller (where the directory is maintained) which interacts with the L2 cache controller (depending on the incluisvity maintained by the caches). As already explained, with minor modifications, the proposed scheme can be applied to the shared unified main memory configuration as well.

3.6.3 Implications on Performance Overhead

If dependable operation (in the event of soft-error occurrences) is desired, then the proposed detector module (along with a suitable rollback and recovery mechanism) needs to be enable permanently. However, typically for the validation purpose, the proposed scheme does not incur any performance overheads as stated before. In the case of permanent on-chip deployment of the proposed scheme (detector module and logging register), the performance degradation (during the step of validation) can be brought down with following improvements:

1. The CCP checking can be done between T^{th} and $(T+p)^{th}$ cycle where $p > 1$. We believe p can take values from 2 to 5 or 10 depending on the design implementation. This can lead to elongation of error detection latency in some cases while

maintaining the same bug coverage in most cases.

2. Some of the checking conditions can be relaxed. For instance- for an illegal/unwanted transition from *Exclusive* to *Shared*. This transition is harmless, because the local processor/core can still read valid data from the cache line, and does not write to this line [75]. Similarly, an illegal change to *Modified* state, it will result in write back when this line is evicted from the cache, but the data does not get corrupted. On the other hand, an illegal change of cache line in *Modified* state to a different state must be detected as upon eviction the cache line will not be written to memory and therefore, memory will no longer be up-to-date [3].

An effective by-pass mechanism for dependable operation (in both design and soft-error cases) needs to be designed such that performance overhead due to all the additional activities (logging in register, error detection, rollback/recovery) is minimized. Such mechanisms have been elaborately discussed in [132, 139].

3.7 Conclusion

Complete functional pre-silicon verification of cache coherence mechanism implementation in modern multi-core systems is a difficult task. With ever increasing processor design complexity, some of the design bugs escape into the first silicon which must be caught before customer shipment. Further, the restricted observability and controllability makes the debug process of cache coherent systems very difficult. This chapter proposed an in-system validation approach to detect design and electrical errors with minimal latency. The proposed methodology utilized high level CCP specifications for the incorporation of an on-chip detector module.

– ∗ – ∗ –

Chapter 4

SAT-based Silicon Debug of Electrical Errors

4.1 Introduction

Error localization from post-silicon debug data is severely challenging and is generally done in a semi-automatic manner. For processors, automated solutions have been developed as identification of internal signals (which are to be traced) is relatively easier and the collected traces can be analyzed with the help of proper classification and segregation at RTL abstraction. However, debugging at granularity of low level (i.e., gates/flip-flops) under restricted visibility is still a difficult task. Debug of electrical errors is very challenging because of several factors, most pertinent of them, is the lack of golden responses [69, 140]. A self-checking method can offer some respite in these debug scenarios. Because of nature of electrical errors, the design netlist is correct. Satisfiability(SAT)-based methods are adept in handling analysis of design netlists through a structural analysis. For automatic electrical error diagnosis, after creating a Boolean formula from the CNF of the unrolled design, we need to apply appropriate constraints to solve it. These constraints assist in the logical inconsistencies in the SAT (satisfiability) instance leading to the root-cause discovery. In this chapter, we propose a graph-based signal tracing methodology to address the signal selection problem and then utilize satisfiability-based

debugging of the obtained post-silicon signal dumps. This provides the most essential benefit as we succeed in debugging without requiring golden responses. We rely on the assumption that the error has been detected in the first silicon either through built-in assertion checkers or application crash while running real-world applications. We also assume that an error trace is available to us for the debugging purpose. Nevertheless, error detection in post-silicon environment and collection of error traces are equally challenging problems. Since post-silicon execution has extremely limited visibility, obtaining constraints for solving the SAT instances is a challenging task and involves trade-offs of debug time, deployed design-for-debug features and required off-chip dumping. In this chapter, different SAT formulations with respect to the constraints required for SAT-based error localization of data collected from the chip execution are analyzed. With different observability enhancement mechanisms, the constraints required for SAT solving differ appreciably resulting into different solutions. The proposed signal selection methodology is agnostic of the type of error and the deployed observability enhancement scheme. Additionally, few selection strategies which rely on graph-based centrality measures are also evaluated. We also discuss the scalability issues of SAT-based debugging under post-silicon environment for electrical error scenarios. The remainder of the chapter is organized as follows. Section 4.2 presents the preliminaries on satisfiability-based error localization methodologies. Section 4.3 presents the proposed signal tracing methodology and its application to different observability expansion techniques. Section 4.4 describes the satisfiability based error localization methodology in post-silicon environment. This section also presents a list of detailed evaluation metrics which are utilized for assessing the efficacy of the proposed technique towards error localization. Experimental setup, results and observations are presented in Section 4.5. An elaborate discussion on the scalability issues is presented in Section 4.6. Section 4.7 describes the related work on post-silicon SAT-based error localization. Section 4.8 finally concludes the chapter.

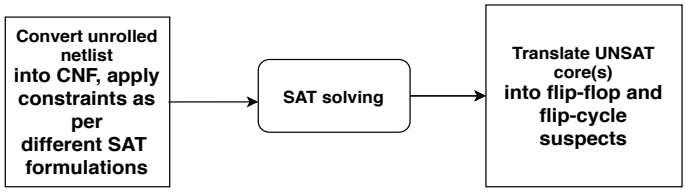

Figure 4.1: Overview of SAT-based error localization

4.2 Satisfiability-based Post-silicon Error Localization

Applying satisfiability (SAT) during design debugging, Suelflow et al.[97] proposed the usage of UNSAT core (clauses which are responsible for unsatisfiability of a SAT instance) for the purpose of error localization. Typically, any SAT-based methodology for a sequential design involves unrolling the netlist and applying various constraints(which correspond to particular debug scenario) on the resulting Boolean formulae. For error localization, we need to compute the UNSAT core(s) after forcing constraints (obtained from the error trace) on the CNF expression of unrolled netlist.

Given a Boolean formula in Conjunctive Normal Form (CNF), if it is UNSAT, any subset of clauses in the instance that is also unsatisfiable is called as an *UNSAT core*[97, 101]. Let's consider the following unsatisfied CNF formula (here, $\cdot$ indicates that the clauses are combined in conjunctive fashion):

$$\phi = (e + g) \cdot (e + f) \cdot (\bar{g}) \cdot (\bar{e}) \cdot (\bar{f} + \bar{g}) \tag{4.1}$$

For the above formula, the UNSAT core(s) include $\{(e+g),(\bar{g}),(\bar{e})\}$. Analysis of these clauses leads to the root-cause of the error. For bit-flip kinds of errors, we need to localize the infected flip-flop (and the corresponding flip cycle). For other kinds of error scenarios such as stuck-at[1] only particular flip-flop (in the logic cone of which the *sa*-0 or *sa*-1 nets/wires are present) is localized. In the post-silicon environment, one challenge is to

[1]the notion of stuck-at here refers to electrical error manifestations because of different reasons. This does not essentially refer to the stuck-at fault of manufacturing testing

force constraints on the SAT instance in such way that the resulting CNF formula turns out to be UNSAT. The generated UNSAT core(s) from the unsatisfiable SAT instance(s) hint towards error localization. However, if the error has not propagated to the obtained signal dumps (from traced signals) or primary input/output values, the error localization process is ineffective. For illustration, Let's consider the following UNSAT core(where the super-script in all literals denotes the respective time-frames/clock cycles):

$$(\overline{a^3} + b^4) \cdot (\overline{g^4} + \overline{b^4} + \overline{h^4}) \cdot (\overline{e^3} + a^3) \cdot (\overline{k^2} + e^3) \cdot (\overline{i^2} + k^2) \cdot (g^4) \cdot (h^4) \cdot (i^2) \quad (4.2)$$

As per the netlist description, the flip-flops are represented by e^3 and b^4 while rest are combinational nets. Therefore, the suspects of bit-flips are either e in the third clock cycle or b in the fourth cycle.

4.3 Proposed Signal Clustering Methodology

As mentioned previously, we assume that the error trace is available and the error has been detected by mechanisms such as assertion checkers or manifestation at the primary outputs. Thereafter, the goal is to localize the error source (at gate-level) given the available error trace (which can be internal signal contents dumped off-line or the primary output values). The primary input values are also assumed to be traced. In this section, we describe the proposed graph-based signal grouping/clustering methodology. The terminology utilized in this chapter is introduced in Table 4.1. As mentioned earlier, for application of SAT-based debugging, chip execution trace is needed which can be utilized in creating a Boolean instance for the localization problem. In a post-silicon environment, a detailed execution trace is impossible to obtain. Thus, to extract useful error trace from the chip execution is of crucial importance. This can be done by off-dumping the contents of on-chip trace buffers. However, whether the collected debug information is profitable or not depends on the trace signals selected for this purpose. We propose a methodology that can be utilized in either static (fixed tracing of signals) or dynamic

manner (through the usage of multiplexers [111, 118]). The proposed grouping/clustering of the signals is aimed at minimizing the number of trace signals required for SAT-based error localization. The proposed methodology explores the logical connectivity among different signals of the design. This merging-based greedy methodology has been adopted in [35] for determining the position of multiple input signature registers which are used for compacting responses stored in shadow flip-flops of the design. In comparison to a selection methodology [34] aimed at certain kinds of errors such as bit-flips, the proposed methodology is agnostic of the targeted error scenarios. We show in the experimental results that different SAT formulations play an important role in error localization for some circuits. Specifically, we demonstrate that only one kind of SAT formulation is unable to achieve exact error localization for all types of circuits.

Table 4.1: Symbols and their meaning

Term	Meaning
N_{clu}	number of clusters
T	length of error trace
B	Boolean formula(for SAT solving)
F_i	flip-flop (where bit-flip is injected)
C_i	clock cycle (where bit-flip is injected)
net_i	net (where stuck-at is injected)
TBw	trace buffer width
S_{Tr}	set of trace signals
k	number of partitions
Su_{fc}	suspect flip candidates (flip-flops)
Su_{cc}	suspect flip candidates (clock cycles)
Su_{sa}	suspect stuck-at candidates (nets)
L_{sc}	length of smaller scan chains
N_E	no. of error injection experiments
F_{tot}	total flip-flops in the netlist
G_{lc}	logic cone connectivity graph
UD_i	unrolled design (netlist) in i^{th} cycle
STB_i	state of traced signals in i^{th} cycle
$SRES_i$	state of restored signals in i^{th} cycle
PI_i	state of primary inputs in i^{th} cycle
PO_i	state of primary outputs in i^{th} cycle

Note that unlike the techniques introduced in [1, 9, 22, 49, 102, 112, 141], the proposed

methodology is not aimed at maximization of the restored signal states ($SRES$).[2] However, during our localization experiments, we utilize the concept of signal state restoration with the signals obtained by the proposed clustering methodology.

Algorithm 3: *MakeGraph*

Input: $Netlist$
Output: G_{lc}

```
1  G_lc, graph_edges ← ∅;
2  FFlist ← all flip-flops in Netlist;
3  FFrem ← FFlist-FF_i;
4  for each flip-flop FF_i in FFlist do
5  |  Add a node to G_lc corresponding to FF_i;
6  end
7  for each flip-flop FF_i in FFlist do
8  |  Cone_i ← Compute fan-in cone of FF_i;
9  |  for each flip-flop FF_j in FFrem do
10 |  |  if edge between nodes for FF_i and FF_j not exists in graph_edges then
11 |  |  |  Cone_j ← Compute fan-in cone of FF_j;
12 |  |  |  if overlap exists in Cone_i and Cone_j then
13 |  |  |  |  edge_wt ← number of overlapping gates between Cone_i and Cone_j;
14 |  |  |  |  add an edge (graphEdge, gE) between nodes corresponding to
   |  |  |  |   FF_i and FF_j with edge_wt as weight of this edge;
15 |  |  |  |  graph_edges ← graph_edges ∪ gE
16 |  |  |  end
17 |  |  |  else
18 |  |  |  |  no overlap exists among these nodes;
19 |  |  |  end
20 |  |  end
21 |  |  else
22 |  |  |  analysis for this edge done;
23 |  |  end
24 |  end
25 |  G_lc ← nodes corresponding to all flip-flops and edges between them
   |   representing logical connectivity overlap between them;
26 end
```

[2]The problem of trace signal selection in post-silicon environment is dealt in a detailed manner in Chapter 5. However, unlike this chapter, we consider debugging of design errors in Chapter 5.

4.3.1 Description of Clustering Methodology

The methodology segregates all the flip-flops (signals) in the design into a certain number of clusters. After the formation of signal clusters, depending on the trace buffer width, the required number of trace signals can be selected from these signal clusters. The proposed methodology consists of two parts- first, the netlist connections are represented by a graph (G_{lc}) where the flip-flops are represented by nodes, second, a merging procedure is followed to cluster the candidate signals and finally select trace signals. The first part is presented as Algorithm 3. The clustering steps are illustrated in Figures 4.2 and 4.3.[3] Typically, each flip-flop has some gates common between its logic cone and logic cone of

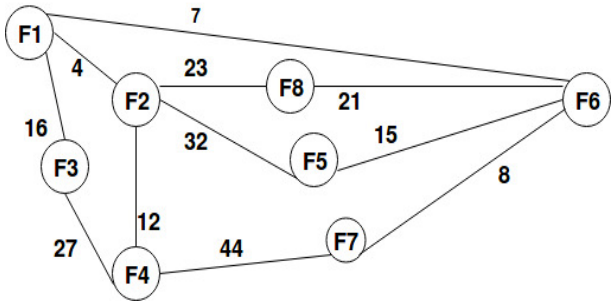

Figure 4.2: Circuit graph(G) illustration

other flip-flop(s). This overlap of logic cones is quantified as *cone sharing*. For the graph (G_{lc}), the *cone sharing* is represented as weights of edges between the different nodes. Thus, the weight on the edges of G_{lc} corresponds to the logic cones of the corresponding nodes measured in terms of the number of gates that are shared between these two cones. Note that G is an undirected graph because an edge between any two nodes does not indicate the availability of any path/direction between these two nodes(flip-flops). Thus, these flip-flops may not be related by any other means except the fact that they share some gates in their respective logic-cones.

A sample G_{lc} obtained from the analysis of a netlist is illustrated in Fig. 4.2 which has eight nodes and eleven edges. This serves as a running example to illustrate the proposed methodology. Since a trace signal becomes highly useful for debugging if the error propagates to it, we need to find effective flip-flops from this perspective. Conditional

[3]Such graph construction is more elaborately discussed in Section 5.2.1 of Chapter 5.

probability-based error propagation methods rely on the assumption of 0/1 at the inputs of the gates. Compared to these methods, we measure the error transmission capability of signals (flip-flops) from the heuristic of logical connectivity, without accounting for sensitization of the errors. With the proposed approach, we aim to reduce the signal tracing overhead. In other words, we are targeting error localization with a reduced number of signals to be traced (particularly 32).

The number of clusters(N_{clu}), which depends on trace buffer width (TBw) is an input to the clustering algorithm. The edge of G_{lc} with the largest weight is selected as the initial edge for the formation of the first signal cluster. Since the edge-weight in G_{lc} represents the logic cone overlap size, the largest edge-weight has a higher probability of error propagation to both the graph nodes. Because of varied types of logical connectivity in different netlists, ensuring the equal size of each cluster turns out to be difficult. Further, we do not wish to constrain the choice of only one element from each cluster for inclusion in the list of trace signals. Therefore, if we get too large clusters, we need to select a larger number of entries. Also, we propose to select a minimum one element(signal) from each cluster. Ranking of the trace signals(flip-flops) inside a given cluster is explained in the latter part of this section.

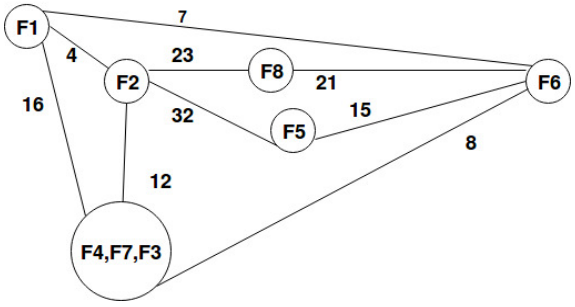

Figure 4.3: Illustration of grouping of nodes

We introduce a second parameter $MinSize$ in the proposed graph-based selection algorithm. The grouping algorithm attempts to assign that many number of elements as minimum to the clusters when they are getting formed. Clearly, the clusters which are formed towards the end fail to full-fill the criterion of $MinSize$. Thus, few clusters can have different number of flip-flops in them. This is especially true for the last cluster which contains all left-over signals. The numerical value of $MinSize$ is given by:

$$MinSize \leq \frac{F_{tot}}{N_{clu}} \tag{4.3}$$

The proposed methodology of trace signal selection is formally presented as Algorithm 4. If a multiplexed signal tracing scheme[118] is available, content of two (or, more) signals/flip-flops from each of the clusters can be dumped into the trace buffers. This scheme can increase the variability in dumping of internal signal contents. Alternatively, similar temporal variation can be observed via the smaller scan chain stitching mechanism as illustrated in Table 2.5.

Algorithm 4: *GroupAndSelectSignals*

Input: G, TBw, $MinSize$, $Netlist$, q
Output: S_{Tr}

```
obtain G_dir from Netlist;
DegRank ← Rank all nodes(flip-flops) of G_dir by in-degree centrality measure;
numGroup ← ∅;
Nodes_G ← nodes in G;
TraceSignals ← ∅;
while numgroup != TBw do
    Edge_hwt ← edge with highest edge-weight;
    N1, N2 ← any 2 nodes of Edge_hwt edge;
    {wt_1, wt_2} ← weights of edges connecting to N1, N2 with the remaining
     nodes of G;
    Node_m ← Merge nodes {N1, N2} into one node;
    Choose maximum out of {wt_1, wt_2} for edge connecting Node_m to other
     nodes of G;
    Remove N1, N2 from Nodes_G;
    Add Node_m to Nodes_G;
    TempGroup ← Node_m;
    if size of TempGroup == MinSize then
        numGroup = numGroup+1;
        begin formation of another cluster;
    end
end
for each cluster g_i in numGroup do
    Rank signals (nodes) of g_i in the decreasing order of their rank in DegRank;
end
S_Tr ← top q signals from each numGroup for equal distribution or select
 different number of top signals from clusters(for unequal distribution);
```

4.3.2 Illustration of Clustering Methodology

Let's consider the graph G for illustration in Fig. 4.2. The edge between node F4 and F7 is selected since this edge has the highest weight, and node F4 and F7 are merged to begin forming the first cluster. Subsequently, the edge weights in G is updated after these two nodes are merged. Thereafter, node F3 is merged with F4 and F7 to form a composite node in the graph G_{lc}. This composite node is shown in Figure 4.3.

The process of adding additional nodes to the obtained composite node continues greedily by selecting the edge with the larger weight attached to the composite node until the size of these clusters reaches minimum size. This essentially means that the current cluster only looks for merging opportunities among its neighbors. Suppose, we fix $MinSize$[4] as 4, then the present cluster formation stops after merging F1 with {F4,F7,F3}. Next, for the formation of second cluster, we observe that F2 and F5 are connected with the highest edge-weight. So, they are merged into one cluster. Note that node F6 is connected to the node containing F2,F5 and F8 through an edge where two edge-weights (i.e., 15 and 21) are possible because of the merging. Out of this, the highest edge-weight (i.e., 21) is selected as the final weight for this edge because of the larger extent of overlap. Thereafter, the last node F8 is also merged with the second cluster {F2,F5,F6}. If there are multiple edges between a composite node and any other node of G_{lc}, the highest edge-weight among them is selected for the final (or, combined) edge between the composite node and that particular node. The final trace signal selection depends on the formation of clusters. For the graph (G) shown in Fig.4.2,

- with $MinSize$ as 4, we obtain two clusters as {F4,F7,F3,F1} and {F2,F5,F6,F8} after the merge-and-update procedure (Algorithm 4) ends. Depending on ranking inside the cluster, the top-ranked signal should be chosen.

- when we fix $MinSize$ as 3, then the three clusters are given by {F4,F7,F3}, {F2,F5,F8} and {F1,F6}. Under such scenario(s), an unequal number of flip-flops(trace signals) can be chosen from each of the three obtained clusters or from

[4]We are assuming a TBw of 2 which equals N_{clu}.

two of them.

4.3.3 Ranking within Individual Clusters

Once the clusters are formed, the final decision regarding the selection of trace signals is done based on graph centrality measures. We consider "degree centrality" for this purpose which is defined as the number of neighbors a node of the graph has. Specifically, we utilize in-degree centrality for ranking of all signals of the flip-flop connectivity graph (G). Note that G is a directed graph where all the nodes are flip-flops similar to the representation in G_{lc}. Edges of G denote the connectivity between flip-flops and the directions on the edges denote the incoming/outgoing logical connection(s). Towards the end of cluster formation, this ranking is utilized to select trace signals from respective clusters. Note that similar graph centrality measures have been utilized in literature for trace signal selection. Eigen vector centrality and Pagerank centrality has been utilized in [142] and [23] respectively.[5] An illustrative graph G is shown in Fig. 4.4 where the nodes FF1, FF2, FF3, FF4, FF5, FF6, FF7 and FF8 denote the eight flip-flops of the design and the weights on these edges depict the number of parallel paths between a pair of flip-flops(nodes). Note that this graph is different from G_{lc} in its representation as it is a directed graph and its edge-weights have a different interpretation as compared to the edge-weights in G. By ranking with the in-degree centrality measure, FF7 is ranked highest followed by FF8 and then FF6. FF2, FF3 and FF4 all have same score, followed by FF5 and the lowest score (which is zero) belongs to FF1.

It is worth to note that degree centrality can be obtained for any undirected graph such as G_{lc} also. However, we prefer the computation of in-degree centrality taking a

[5]Since signals corresponding to the techniques of [142] and [23] are not available publicly, we implemented these procedures and obtained those signals after the analysis of G for different benchmarks utilized in our experiments.

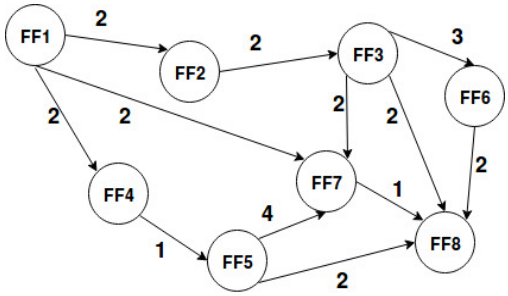

Figure 4.4: G illustration

directed graph(G) for the purpose of ranking flip-flops(nodes) inside a cluster.[6] We also utilize G later for one of the metrics used for evaluation of bit-flip error localization.

4.3.4 Algorithmic Complexity of Clustering Algorithm

The worst-case time complexity of the above clustering algorithm can be expressed as $\mathcal{O}(p^2)$ where p is the number of nodes in the graph G_{lc}. The time spent in clustering for a variety of benchmarks is reported in Figure 4.6. Based on this result, we believe that the complexity is significantly dependent on the number of edges in G_{lc}. Note that the complexity of degree centrality measure computation is also expressed as $\mathcal{O}(p^2)$. However, obtaining degree centrality is independent of the steps in the clustering algorithm and can be directly computed before the start of the clustering procedure.

4.4 SAT-based Post-silicon Error Localization

We formulate SAT-based error localization in four different ways which are outlined in Table 4.2. The methodology in [34] performs the error localization only for the case when constraints corresponding to traced and restored signal state values are enforced on the unrolled netlist. It is worth to note that the evaluation in [34] has been done

[6] An interesting proposition is to select final trace signals by simply utilizing in-degree centrality measure instead of cluster formation and then subsequent selection. However, this graph (G) does not take into account of gates present along these paths and may not help us in selecting trace signals such that a flip anywhere in the circuit could be localized. For the sake of completion, we evaluated signals obtained by in-degree centrality in our experiments too. They did not perform better than the signals obtained through the clustering-based approach (Algorithm 4). This is primarily because of coarse approximation of error transmission capabilities through the in-degree measure.

Table 4.2: Different SAT formulations

Name	Expression	Key feature
B_1	$\prod_{i=1}^{i=T}(UD_i \cdot PI_i \cdot PO_i \cdot STB_i)$	traced, PI, PO
B_2	$\prod_{i=1}^{i=T}(UD_i \cdot PI_i \cdot PO_i)$	only PI,PO
B_3	$\prod_{i=1}^{i=T}(UD_i \cdot STB_i)$	only traced
B_4	$\prod_{i=1}^{i=T}(UD_i \cdot PI_i \cdot PO_i \cdot STB_i \cdot SRES_i) \cdot FF_T$	traced, restored PI,PO

on few circuits (corresponding to ISCAS'89 benchmark suite only). We observed in our experiments that such a formulation with restored and traced signal values only fails to perform error localization in case of other types of circuits (ITC'99,Opencores[143]). As per the terminology in Table 4.1, UD_i represents the unrolled circuit in i^{th} time-frame, PI_i and PO_i respectively represent the primary input and primary output values in i^{th} clock cycle. STB_i and $SRES_i$ respectively represent the signal values of traced signals and restored signals in i^{th} clock cycle. Note that for each value of i, the number of signal states is fixed in STB_i whereas the number of signal states is variable in the case of $SRES_i$. If the restored signal values are not utilized for creating the Boolean formula (B_j[7]), $SRES_i$ is dropped from the above equation. However, we show in the experimental section that usage of state restoration technique assists in obtaining a lesser number of *suspect* candidates. By providing B_j to the SAT solver, the UNSAT core can be obtained which hints towards the error root-cause. Since the sequential circuit needs to be unrolled for T time-frames (i.e., clock cycles), the size of B_j in terms of number of clauses increases which can lead to serious implications on the solver time (and memory usage) for large designs. For example- in the case of $s38584$ circuit, the number of clauses after the netlist is unrolled for 256 clock cycles reaches close to 13.4 million. It has also been observed that a ten times larger SAT instance requires approximately hundred times larger run time and ten times higher memory usage while solving [34]. Therefore,

[7]It is clear that j can take the values of 1,2,3 or 4.

to perform error localization in the case of large error traces, we break the complete error trace ($ETrace$) into smaller sections of the trace (i.e., chip execution). The goal is to obtain resulting SAT formula/instance, B_j in such a manner that it becomes relatively smaller leading to reduced requirement of runtime and memory for SAT solving. It is clear that the number of such partitions would vary depending on the trace length and the debug scenario.

4.4.1 Methodology for Debugging Large Error traces

The collected chip execution trace, $ETrace$ of length T (consisting of traced signal states and/or PI and PO signal values) can be viewed as a composition of smaller error traces ($eTrace_j$). This combination can be represented by the following equation:

$$ETrace = \cup_{j=1}^{j=k}(eTrace_j) \tag{4.4}$$

where, k is the number of divisions required. However, while applying constraints on the SAT instances for each division of $ETrace$, we need to know the initial values of the circuit state for each smaller division. Assuming the reproducible behavior of the bugs, this can be obtained with the help of scan chains. For example- Consider T as 1000 and k as 10, to know the initial state for the second division ($eTrace_2$), the contents of the flip-flops at the end of 100^{th} clock cycle are simply the scan chain values (at 100^{th} cycle) which can be dumped off-line. This is the link between different partitions and information is carried over from one partition to another using it. In the SAT formulations shown in Table 4.1, we additionally utilize this initial state while generating B_j expressions. There are two possible scenarios which can emerge when SAT-solving of each of the partitions (i.e., $eTrace_j$) is attempted:

1. The Boolean formula corresponding to only one of the partition ($eTrace_j$) is unsatisfiable(UNSAT) while the formulae for all other partitions are satisfiable.

2. The Boolean formulae corresponding to more than one partition ($eTrace_j$) are UNSAT.

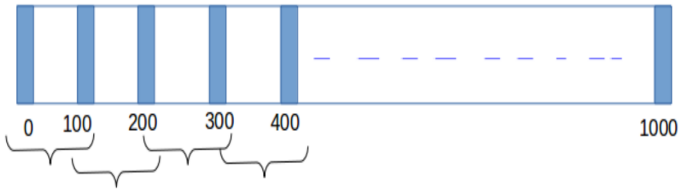

Figure 4.5: Breaking large error trace into smaller ones

For the second case discussed above, the error has propagated to the subsequent partitions in addition to the partition containing the flip-cycle i.e., the actual partition, $eTrace_j$ and $eTrace_{j+1}$. In our experiments, we observed lesser occurrences of the second case. The complete SAT-based localization analysis is then performed for these smaller error traces ($eTrace_j$) leading to the unrolling of the circuit for only 100 cycles compared to the case when the netlist needs to be unrolled for a total of 1000 cycles (which is complete error trace length).

4.4.2 Description of Evaluation Metrics

4.4.2.1 Metrics for bit-flip error localization

For bit-flip errors, we need to achieve both temporal localization and spatial localization with the mentioned SAT-based methodology. Thus, for quantifying the efficiency of this methodology, we must devise some metrics. Let $n(v)$ denote the number of times condition v occurs during a run of error injection experiments (represented earlier by N_E). We obtain Su_{fc}, Su_{cc} and Su_{sa} as solution(s) after translating the UNSAT cores(s) obtained by solving the respective B_j formulations. In the notations Su_{fc} and Su_{cc}, the sub-script fc means flip-flop candidates and cc means cycle candidates respectively. Therefore, suspect candidates are the net names/flip-flop names of the circuit which are annotated with the time-frame number. Given this notation and the terminology introduced earlier in Table 4.1, two important evaluation metrics for bit-flip localization can be described as follows:

$$\phi_1 = \sum n(B_j \ is \ UNSAT) \tag{4.5}$$

The above metric tells the number of iterations (out of a total of N_E experiments) in which the Boolean formulae becomes UNSAT. It is obvious that we prefer a value as high as possible for ϕ_1 which can attain a maximum value of N_E. To measure the quantity of exact error localization($C \in Su_{cc}$, $F \in Su_{fc}$) for bit-flip errors, we define the following metric:

$$\phi_2 = \sum n(F = F_i \; and \; C = C_i) \tag{4.6}$$

The above metric gives the number of iterations (out of a total of N_E experiments) in which both the suspect flip-cycle and flip-flop are obtained. However, we observed during our experiments (especially on opencore circuits[143]) that the above metrics do not capture the variation in efficacy of the different formulations of B in terms of error localization, especially in the case of incomplete error localization. Hence, we devised a set of detailed evaluation metrics (in Table 4.3) which also assist in capturing the variation of the efficacy of SAT-based debugging with different trace signal selections.[8]

In the case of incomplete localization, we evaluate the spatial localization indirectly by measuring the distance between the bit-flip(F_i) and the obtained suspect candidates (Su_{fc}). The motivation behind this is to analyze all flip-flops within a topological distance for the injected bit-flip(F_i). By analyzing the graph G, we obtain O as the topological arrangement of all flip-flops (F_{tot}) of the design. The topological position of any flip-flop (F) is denoted by $O(F)$ for a given G. The metric is formally defined as ϕ_7 in Table 4.3. Note that obtaining a topological order of nodes of a graph (which represents the flip-flops of the circuit) becomes a non-trivial exercise when the graph contains cycle(s). Therefore, for approximation, we consider all the cycles as removed and then compute the topological order. This is a deviation from the exact meaning of topological order, however, we believe that our purpose can still be full-filled to some extent.[9] For the graph shown in Figure 4.4, one topological ordering is given by FF1, FF4, FF5, FF2,

[8] In the published conference version of this work ([32]), the evaluation of localization efficacy was done only in terms of size of Su_{fc}. This kind of evaluation becomes clearly an over-simplified quantification of the methodology and did not consider the cases of inexact localization. Additionally, the detailed set of metrics in Table 4.3 helps us to analyze the input sensitivity in error localization.

[9] This method of obtaining a topological order for cyclic graphs is similar to the methodology presented in [51], which is explained in Chapter 5.

Table 4.3: Detailed Evaluation metrics

Metric	Expression	Meaning
ϕ_3	$\frac{\sum\lvert C_i - C\rvert}{N_E}$	avg. distance of flip-cycle
ϕ_4	$\frac{\sum n(F)}{N_E}$	avg. no. of flip-flop suspects
ϕ_5	$\sum n(F = F_i)$	exact flip-flop localization
ϕ_6	$\sum n(C = C_i)$	exact flip-cycle localization
ϕ_7	$\frac{\sum\lvert O(F_i) - O(F)\rvert}{N_E}$	avg. topological order distance of the suspect flip-flops

FF3, FF7, FF6 and FF8. Therefore, the averaged topological distance shows the effort in analyzing the suspect flip-flops towards localizing culprit flip-flop (F_i).

In the computation of metrics ϕ_3[10] and ϕ_6, $C \in Su_{cc}$. Similarly, in case of metrics ϕ_4, ϕ_5 and ϕ_7, $F \in Su_{fc}$. The metric ϕ_3 measures how far away the obtained suspect cycles (in Su_{cc}) are from the actual flipped cycle (C_i) for all the suspect flip-flops present present in Su_{fc}. Similarly, metric ϕ_4 counts the averaged number of obtained suspect candidates (flip-flops) each corresponding to a different flip-cycle. We show in later sections that these metrics are important for measuring the quality of SAT-based silicon debugging of bit-flip errors. These metrics assume more significance in efficiency evaluation across different SAT formulations when the methodology succeeds in achieving only partial/inexact error localization.

4.4.2.2 Metrics for stuck-at error localization

In accordance with the terminology introduced in Table 4.1, by translation of the UNSAT cores after solving, we obtain the suspect stuck-at nets, Su_{sa}. Similar to the bit-flip error case, the following metric is defined to measure the efficacy of SAT-based stuck-at error

[10] Another way of evaluation could be dividing the numerator of metrics ϕ_3, ϕ_4 and ϕ_7 by ϕ_1 instead of N_E as those iterations which turn out SAT, can be removed.

localization (where $net \in Su_{sa}$ and net_i is the wire/net in the netlist which is stuck-at):

$$\phi_8 = \sum n(net = net_i) \tag{4.7}$$

Another metric worth consideration is the total number of suspect nets obtained after analysis of UNSAT cores in the error injection experiments:

$$\phi_9 = \frac{\sum n(net)}{N_E} \tag{4.8}$$

It is obvious that lower the values of ϕ_9, better is the stuck-at error localization. Note that the opposite trend is true for metric ϕ_8 i.e., maximum iterations (out of a total of N_E) should turn into UNSAT so that the stuck-at error can be localized after analysis.

4.5 Experimental Results and Observations

We used *PicoSAT*[144] as the SAT solver for extracting the UNSAT core from the Boolean formulae. For simulation under different error conditions, we utilized our in-house developed simulator. We performed time-frame expansion in a fashion similar to [24]. For all other scripting tasks (viz. SAT formula encoding), we utilize Python and bash shell functions. We performed localization experiments for different benchmark circuits of varying sizes as reported in Table 4.4.

4.5.1 Chosen Benchmark Circuits

Across all the chapters in the thesis, proposed algorithms/techniques[11] are evaluated on a variety of benchmark circuits, characteristics of which are reported in Table 4.4. These circuits are chosen from 3 different widely popular suites- ISCAS'89 [145], ITC'99

[11]The evaluation in Chapter 3 is based on an open-source RTL framework and hence does not have scope of application to benchmark circuits. Chapter 7 utilizes benchmark circuits in RTL format as well as their .bench descriptions, whereas Chapters 4, 5 and 6 utilize only the .bench description of these benchmark circuits.

[146] and Opencores [143] which have been utilized in experiments in [23, 96]. Some more details on them are available in Appendix A. We deliberately omit the results on the smaller benchmarks from ISCAS'89 and ITC'99 for our experiments as they contain lesser number of sequential elements (flip-flops) in them. Because of lesser number of flip-flops (i.e., < 100 or < 200) in the design, tracing of 32 flip-flops (which is typical trace buffer width in literature) is not needed. From our experiments, we deduced that for such smaller circuits, 16 (or even less than 16) flip-flops are sufficient. The circuits elaborated

Table 4.4: Characteristics of benchmark circuits

Type	*Name*	*FFs*	*Gates*	*PI*	*PO*
ITC'99	*b*15	417	10241	37	69
ITC'99	*b*17	864	13988	39	96
ITC'99	*b*21	429	13098	33	22
ITC'99	*b*14	245	9767	32	54
OpenSparc[23]	dmu_{clu}	2893	14850	697	1344
Opencore[143]	*p*16*c*5*x*	739	4107	26	191
Opencore[143]	*mips*	1859	38190	74	106
Opencore[143]	*usb*	1761	10650	128	106
Opencore[143]	*softusb*	317	4718	34	50
ISCAS'89	*s*5378	179	2779	35	49
ISCAS'89	*s*9234	228	5597	19	22
ISCAS'89	*s*13207	669	7951	62	152
ISCAS'89	*s*15850	597	9772	77	50
ISCAS'89	*s*38417	1636	22179	28	106
ISCAS'89	*s*38584	1452	19253	12	278
ISCAS'89	*s*35932	1728	16065	35	320

in Table 4.4 serve as representatives of digital blocks inside complex SoCs because of varying number of sequential and combinational elements. For industrial designs, trace buffers may be distributed across the entire chip [125]. This distribution would follow a partitioning formulation resulting into specific number of blocks (or, modules). The proposed techniques of signal selection and automatic bug localization can be applied to each of these internal blocks (i.e., modules) of the large design.

All experiments were conducted with a machine having 16GB RAM and Intel i7 processor operating at 2.80 GHz. Note that the flip-cycle varies in all the experiments

unlike a fixed flip-cycle utilized in the experiments in [34]. We report the time[12] spent in clustering process for final signal selection in Figure 4.6. When we attempt to choose from 8 or 64 clusters, we observed the same trend (i.e., minuscule variation) for the time spent in clustering. We observed that some circuits having lesser number of flip-flops

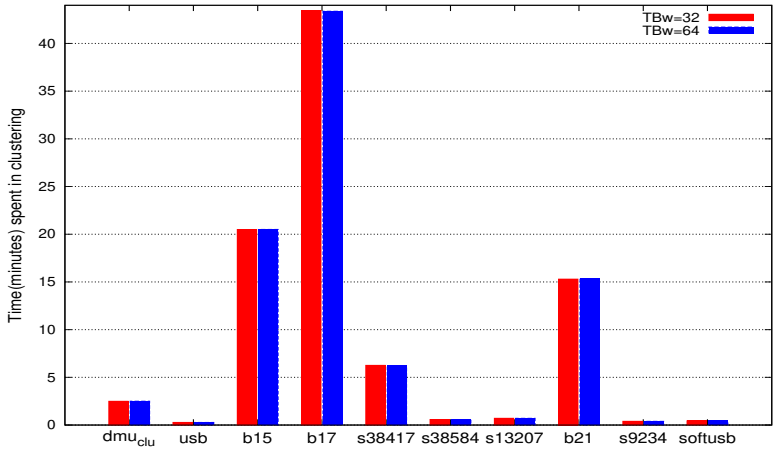

Figure 4.6: Time(minutes) spent in clustering

(which correspond to nodes in G) require less time as compared to those having higher number of flip-flops. We believe that this is because of the structure of the circuits.

4.5.2 Comparison with Other Signal Selection Methods

4.5.2.1 Exact localization of bit-flip errors with different SAT formulations

With different selection of signals for tracing, variation in the number of suspects(ϕ_2 or ϕ_4) is expected. Considering N_E as 102, T=100 and TBw as 32, results of metric ϕ_2 for s38417 and s38584 circuits are shown in Fig. 4.7 and 4.8 respectively. We obtained the same values for ϕ_1 and ϕ_2 for these circuits. The values of metrics ϕ_5 and ϕ_6 were also identical to that of ϕ_2 (Note that ILP-based, RATS, Hybrid, Graph-based and Pagerank refer to [9], [1], [141], [142] and [23] respectively, Clustering denotes the proposed method). We observed that the clustering-based signal selection technique performs slightly better than some of the other selection methods (including the graph centrality based measures [23, 142] and those aimed at restorability maximization [1, 9, 141]) for

[12]The reported numbers on Y-axis denote the wall clock time (elapsed time) taken by the particular command to get executed.

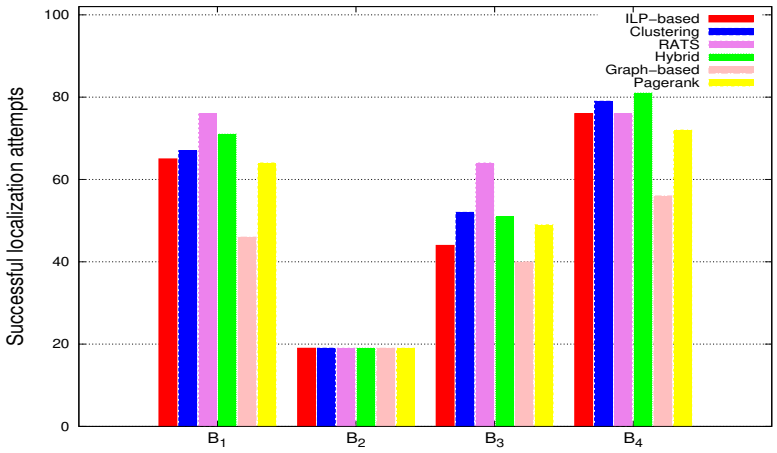

Figure 4.7: Successful bit-flip localization results (ϕ_2) for s38417

$s38417$. Clustering-based technique offers better localization than some of the previous techniques for B_4 formulation. Furthermore, the clustering-based signal selection can be done in quick time (shown in Table 4.6) as compared to other techniques, particularly that of [9]. It is worth to note the effectiveness of B_4 towards error localization as compared to the other formulations. The same observation has also been noted in [34] based on their analysis of experiments on ISCAS'89 circuits only. As expected, B_2

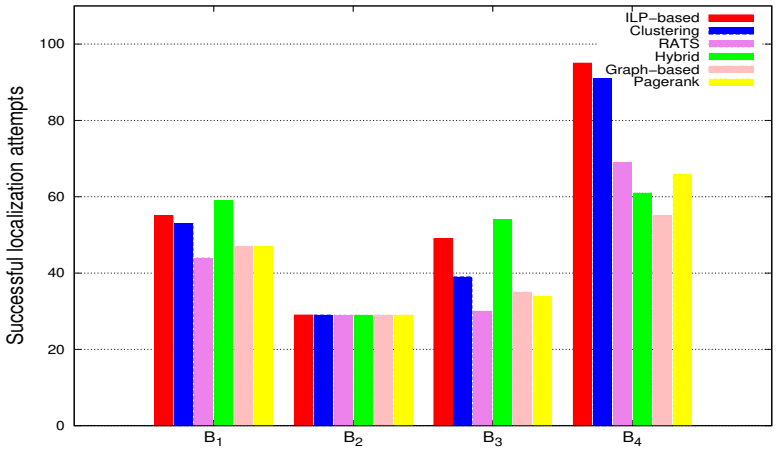

Figure 4.8: Successful bit-flip localization results (ϕ_2) for s38584

performs poorly for both the circuits, $s38584$ and $s38417$. However, we show in results in later part of this section that this formulation(B_2) is useful for other type of circuits, especially when other formulations fail to achieve exact localization of bit-flip. In fact, the same holds true for stuck-at errors (Figure 4.18).

We increased N_E to 250 for $s38584$ circuit and performed the localization analysis for each SAT formulation. With the clustering-based selection, the values of metric ϕ_2 for B_1, B_2, B_3 and B_4 are obtained as 133, 47, 99 and 153 respectively. This shows that for larger values of N_E, the different SAT formulations follow the same trend as described above for N_E as 102.

4.5.2.2 Variation in localization parameters with SAT formulations and selection techniques

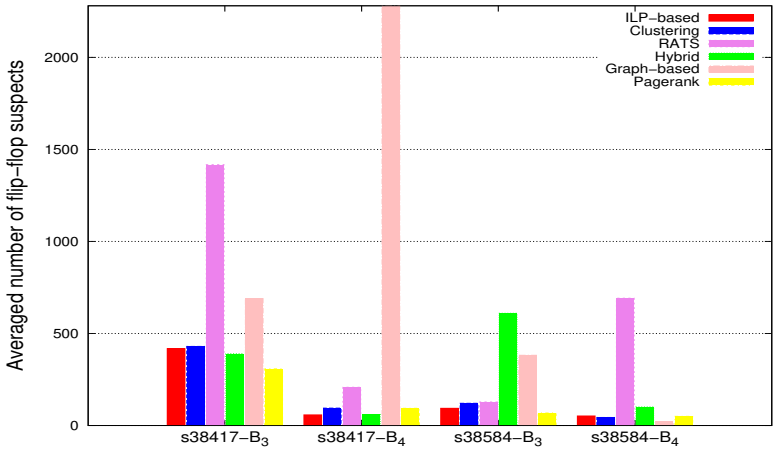

Figure 4.9: Avg. number of bit-flip suspects (ϕ_4) for s38417, s38584

As stated before, we need to achieve both spatial and temporal localization for bit-flip errors. Since we could not achieve exact localization for each iteration in N_E, we measured ϕ_3 and ϕ_4 for the clustering-based selection, restorability-based and graph measures-based selections. As defined in Table 4.3, ϕ_4 measures the average number of suspects(flip-flops) obtained in each error injection experiment. Therefore, the lesser the values of ϕ_4, the better is the bit-flip error localization. The same trend is true for other metrics like ϕ_3 and ϕ_7. We observed that clustering-based technique provides lesser values of ϕ_4 as compared to other techniques in majority of the cases. However, ϕ_3 values with clustering-based technique are slightly higher as compared to other methods. In this context, it must be stated that clustering-based techniques provides slightly higher ϕ_1 values as compared to other methods. Because of this, techniques which have lower

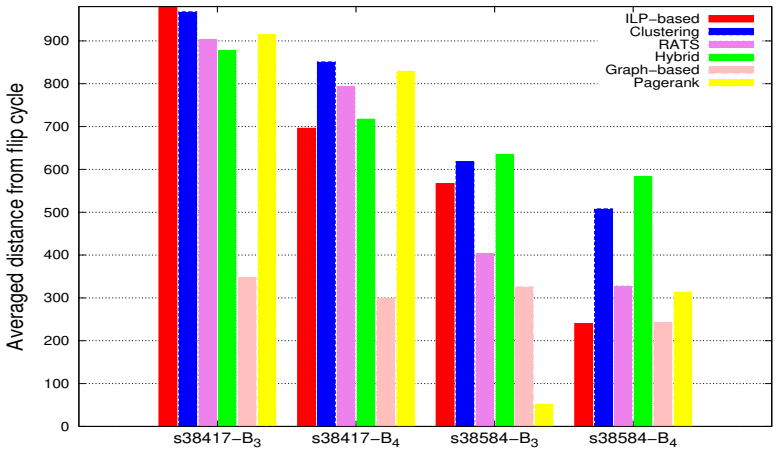

Figure 4.10: Avg. flip-cycle distance (ϕ_3) for s38417, s38584

values of ϕ_3 and ϕ_4 are not always useful as a lower value of ϕ_1 means that B_j's are not very effective towards error localization.

The strong dependence of localization efficiency on SAT formulations is an important takeaway from our detailed analysis of experiments on $s38417$ and $s38584$. We report results of metrics ϕ_1, ϕ_2, ϕ_3, ϕ_4 and ϕ_7 for rest of the benchmark circuits in the next sub-section (results for circuit $s9234$ are discussed later in a different sub-section). Note that B_4 performs better in temporal localization as lower values of ϕ_3 are obtained as compared to other formulations. Hereafter, we present error localization results obtained with the clustering-based signals only(Algorithm 3). We ran experiments with other techniques also to study the localization sensitivity with trace signals. In cases of inexact localization, varying trace signal selection has very less impact.

4.5.3 Error Localization Results with Fixed Tracing

Unless stated otherwise, we have always considered N_E as 102, T=100 and TBw as 32 in our experiments. For bit-flip errors, we obtained varying degree of success in error localization for the circuits reported in Figure 4.11. We observed no major difference in ϕ_1, ϕ_3, ϕ_4 and ϕ_7 for different signal selection techniques for these circuits.

As per definition of ϕ_1, its higher values are desirable. However, the higher values of ϕ_1 along with relatively lower values of ϕ_2 hint towards inexact temporal and spatial

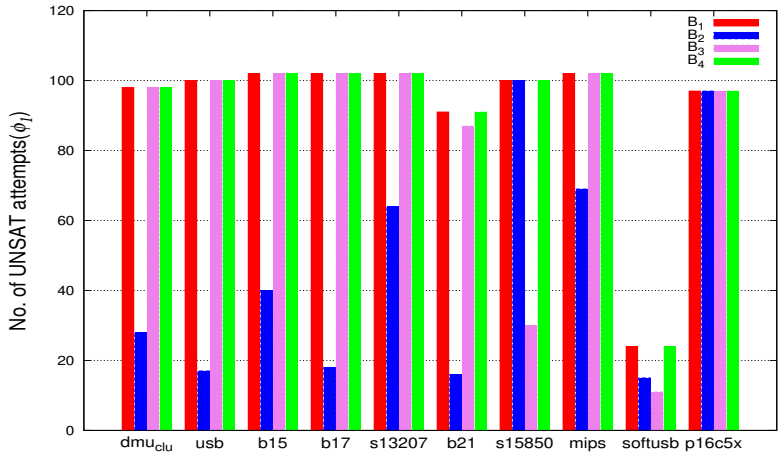

Figure 4.11: Number of UNSAT attempts(ϕ_1) with different SAT formulations

error localization. As observed from Figure 4.11, except for $softusb$, B_2 provides very lower ϕ_1 compared to other three formulations. However, as observed from Figure 4.12, B_2 provides higher amount of exact error localization as compared to other formulations. We observed minor difference in ϕ_2 values for B_1 and B_4. Nevertheless, these values were lower than that of B_2 and B_3. As stated earlier, trace signal selection has minimal in-

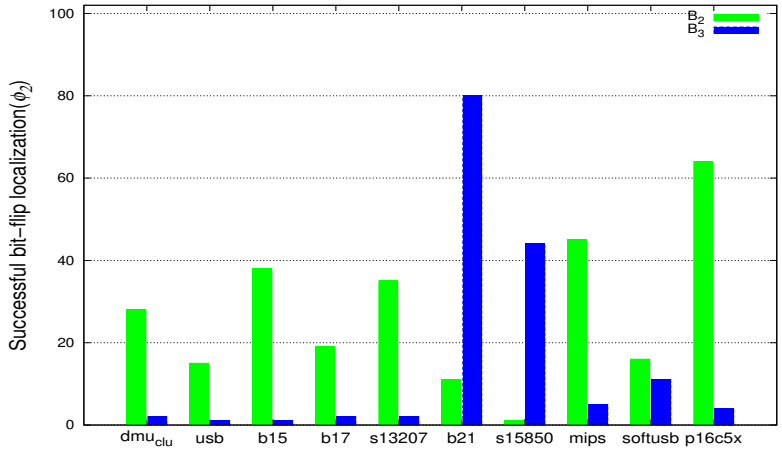

Figure 4.12: Successful bit-flip localization (ϕ_2) with B_2 and B_3 formulations

fluence on error localization of these circuits. Therefore, we believe that the failure of suitable UNSAT core(s) extraction from the Boolean formulae constructed with unrolled circuit netlist and PI_i, PO_i, STB_i or $SRES_i$ constraints is the reason behind this inefficiency (i.e., lower values of ϕ_2). As expected, changing primary inputs has impact on the achieved temporal, spatial localization, avg. number of flip-flop and flip-cycle suspects

and the avg. topological distance from F_i. We observed that for circuits giving lower values of ϕ_4 or ϕ_3, generally less than five suspect candidates are obtained in each iteration of N_E. These suspects do not match with the actual culprit flip-flop (F_i) indicating that the suitable UNSAT core(s) were not extracted from the respective B_j formulations. We

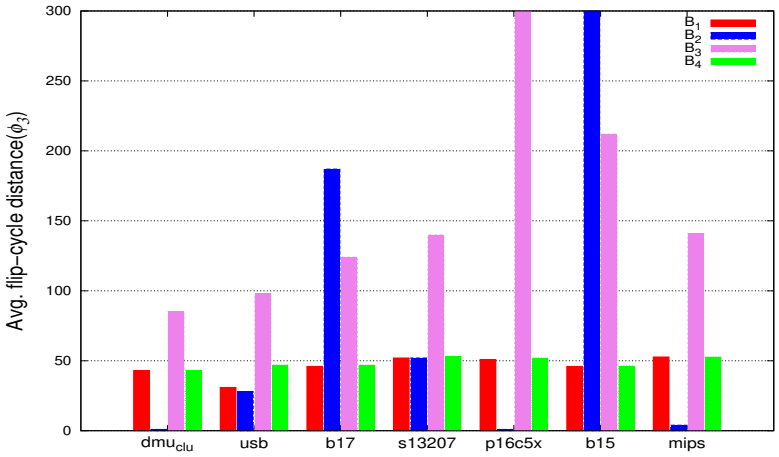

Figure 4.13: Avg. flip-cycle distance (ϕ_3) with different SAT formulations

observed a strong dependence of inputs (along with SAT formulations) on the localization efficiency for these circuits. Available description of these netlists reveal that these are mostly processor designs containing structures such as registers, instruction and address pointers, multiple ready/enable signals and so on. Since we rely on random inputs (along with explicit specification of few inputs like clock/reset), the formulations fail to discover the proper logical inconsistencies[13] which in turn would have led to suitable UNSAT core(s). In some of the formulations for few circuits, we were able to achieve exact only temporal localization (which is measured by ϕ_6 metric). Similar trend is observed for other circuits such as $softusb$, $s15850$ and $s35932$.

One of the interesting observations of our experiments is the poor performance of B_4 or B_1 even if B_2 or B_3 succeeds in error localization. The primary reason behind this is the meddling in UNSAT core(s) when additional constraints in the form of the restored signal states or the traced signal states are added to obtain B_4 from B_2 or B_3

[13]We did another set of error localization by properly assigning the reset/clock values. However, there was only minuscule improvement in the error localization efficacy. This observation hints towards interplay of a range of different other factors also.

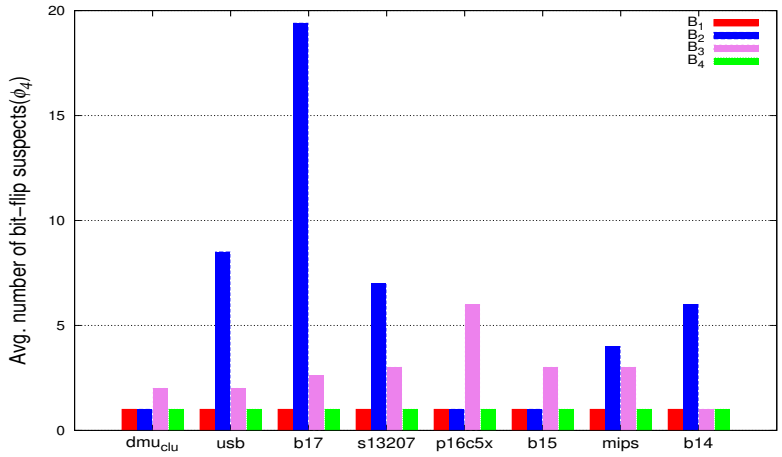

Figure 4.14: Avg. number of bit-flip suspects with different formulations for different circuits

formulation. Figure 4.15 shows the values of ϕ_7 reflecting the averaged number of flip-flops(which are represented by nodes in G) required to be analyzed for identifying the actual flip-flop(F_i) affected by the bit-flip. Metrics such as ϕ_5 and ϕ_6 follow the same trends as depicted by the above figures.

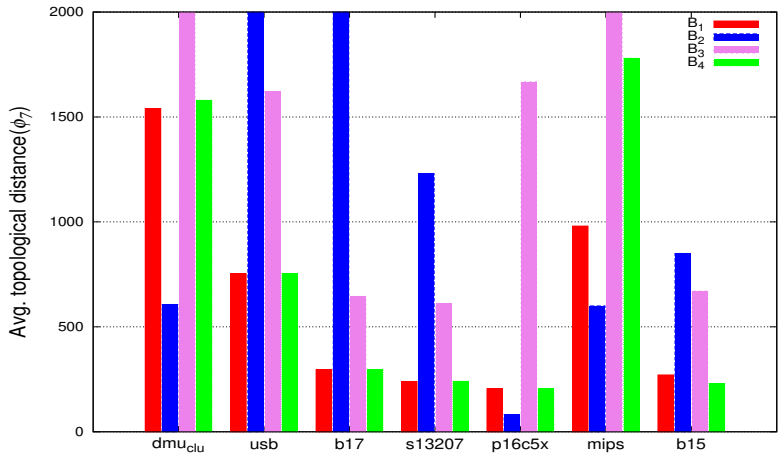

Figure 4.15: Avg. topological distance (from culprit flip-flop) with different SAT formulations

Achieving better error localization for these circuits is one of the directions of future extension of this work. Obtaining suitable input constraints is needed which can help in enhancing the quality of error localization. We believe that properly chosen inputs (taking the design hierarchy into account) can improve the performance of all four SAT formulations for error localization because the traced/restored signal states

($STB_i/SRES_i$) also depend on the inputs(PI_i) to a large extent.

4.5.4 Impact of Increasing Trace Buffer Width on Bit-flip Error Localization

An increase in trace buffer width(TBw) increases the amount of internal signal visibility. We performed experiments on $s9234$ circuit for 4 different trace buffer widths as 16, 32, 48 and 64. The values of ϕ_2 which denote the number of exact solutions (out of N_E) are reported in Figure 4.16. In most of the cases for all the three formulations of $B1$, $B3$ and $B4$, there is an increase in the number of exact solutions as trace buffer width is increased. It is obvious that with change of TBW, there is no impact on B_2. Note that upon TBw increase, there is a slight decrease in ϕ_2 for B_4. This can be because increase in signal state restoration leads to SAT instance(s) becoming satisfiable from which no UNSAT core(s) could be extracted. Similarly, we observed an increase in ϕ_1, ϕ_5 and ϕ_6

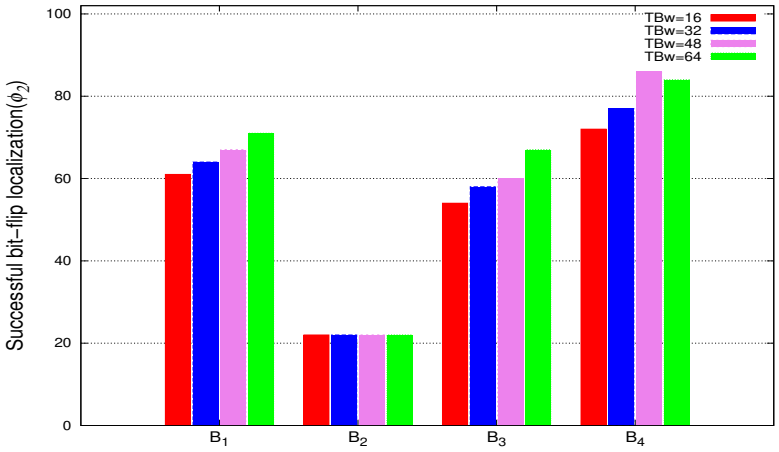

Figure 4.16: Successful bit-flip localization for s9234

when TBw is increased for other circuits also. Additionally, we observed a decrease in ϕ_3 and ϕ_4. The decrease in these metrics range from 2% to 50% as we increase the trace buffer width. Also, we observed a decrease in the topological distance from flip-cycle (ϕ_7) as the error localization improves on TBw enhancement.

4.5.5 Impact of Different Selection from Clustering Choices and Ranking Inside Clusters

As was previously stated, selecting different signals from obtained clusters can have implications on error localization. We performed a detailed experiment on four circuits, for which the variation is shown in Figure 4.17. For $softusb$ and $b21$ circuits, we considered two configurations(denoted by C1 and C2) with TBw as 32 and 64 respectively. For $softusb$ circuit, we consider C1 as selection of one flip-flop from each of 32 clusters, C2 as the selection of two flip-flops from each of 16 clusters. Inside the individual clusters, the ranked signals as per the in-degree measure are chosen as final trace signals. Except B_2 formulation, other 3 formulations provide better bit-flip localization with C2 grouping than C1 grouping. For $b21$, in C1, we had put $MinSize$ as 9 and for C2, $MinSize$ was chosen as 19. Similarly, for $b21$ circuit, we consider C1 as selection of one flip-flop from

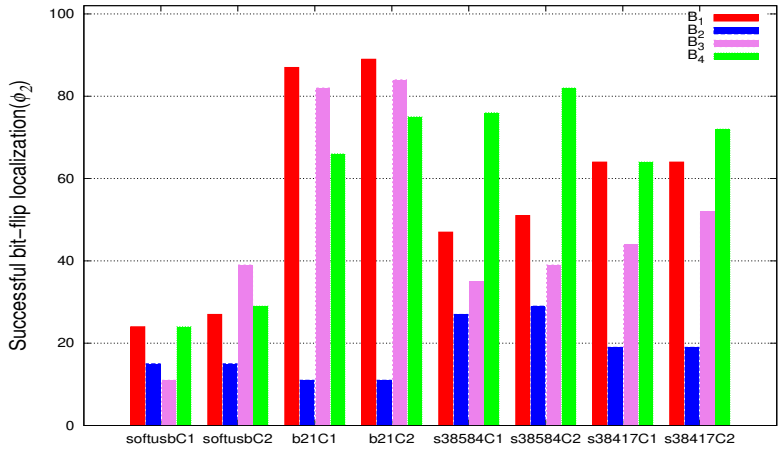

Figure 4.17: Successful bit-flip localization for different clustering configurations

each of 64 clusters, C2 as selection of two flip-flops from each of 32 clusters. For $b21$, in C1, we had put $MinSize$ as 6 and for C2, $MinSize$ was chosen as 13. We observed that for the second configuration(C2), ϕ_2 values are higher as compared to that of C1. In other words, configuration C1 resembles the choice of trace signals in [32]. This means that the ranking-based selection of two flip-flops from each cluster is more beneficial over the selection of one signal from clusters. We observed similar trends during our experiments on $s38417$ and $s38584$ circuits. For $s38417$, C1 and C2 are chosen with respective $MinSize$ as 24 and 48 respectively. Similarly, for $s38584$, C1 and C2 are chosen

with respective $MinSize$ as 21 and 42 respectively. As expected, there is no impact of clustering variation on error localization with B_2. Analyzing impact of unequal selection of flip-flop/signals from each cluster is also one of the directions of future work.

4.5.6 Localization of Stuck-at Errors

We injected random *stuck-at* fault at one of the nets (n_i)in the netlist. Since, the UNSAT core(s) contain nets and flip-flops together, after translating the obtained UNSAT core(s), we can obtain a list of suspect nets. We carried out the localization efficiency evaluation by using metrics ϕ_8 and ϕ_9. Note that in addition to an electrical scenario, a stuck-at condition can also resemble a design error[100]. The flip-flops (signals) in the outward logic cone of the net n_i are computed and can be matched with the translated UNSAT core(s). We did not utilize this approach (of identifying the flip-flops in the outward logic cone of n_i) because a subsequent analysis needs be performed to localize the particular net. Once again considering $N_E = 102$, $T = 100$ and TBw as 32, the values of metric ϕ_8 are reported in Figure 4.18. For some of the circuits, the SAT formulations

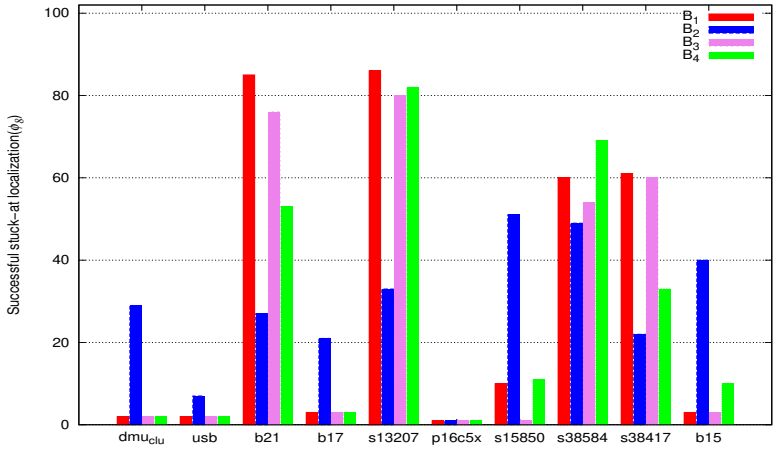

Figure 4.18: Successful stuck-at localization

succeed in localizing the particular net (n_i). Note that stuck-at error localization(Su_{sa}) is also strongly dependent on the suitable sensitization by the primary inputs. In fact, the requirement of sensitization by primary inputs make the stuck-at error localization difficult. To measure the inexact error localization, values of metric ϕ_9 are reported in Figure 4.19. For dmu_{clu} and usb, only few nets are obtained after translating the UN-

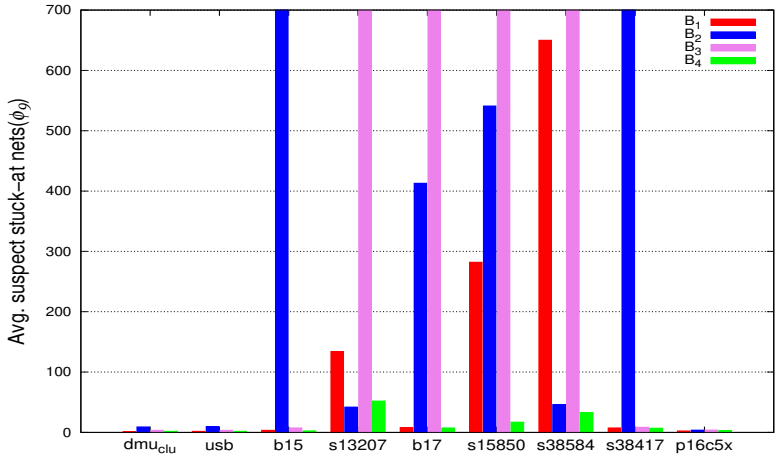

Figure 4.19: Avg. suspect stuck-at nets with different formulations for different circuits

SAT core(s) even though for a large number of error iteration, the constrained Boolean formula (B_j) turns into UNSAT. We observed a strong dependence of ϕ_9 on the trace signal selection technique which means that different trace signal lists provide different kinds of UNSAT core(s) during extraction from the unsatisfiable instances. For $p16c5x$ circuit, the exact suspect net localization is poor. The localization for these circuits is poor because of improper sensitization by inputs which need to be identified carefully. Nevertheless, stuck-at fault localization with a limited error trace is not a trivial task. We aim to refine the stuck-at localization by utilizing the technique of [100] as future enhancement.

4.5.7 Error Localization with Temporally Variable Visibility Enhancement

We consider small chains (each of length two/four which means that two/four flip-flops are stitched together) with the scheme outlined in [22, 119]. For TBw as 16, 32 signals can be dumped as explained in Section 2.3.3. Figure 4.20 shows the values of metric ϕ_2 for such an observability mechanism for bit-flip errors. Here, TBw of 16 is considered, however, because of temporal tracing, 32 and 64 flip-flops(signals) are traced when length of smaller scan-chains are considered as two(denoted as scan2, L_{sc}=2) and four(denoted as scan4, L_{sc}=4) respectively. Note that as stated previously, the dumping of the traced signals is not continuous which leads to a respective dumping period of two and four

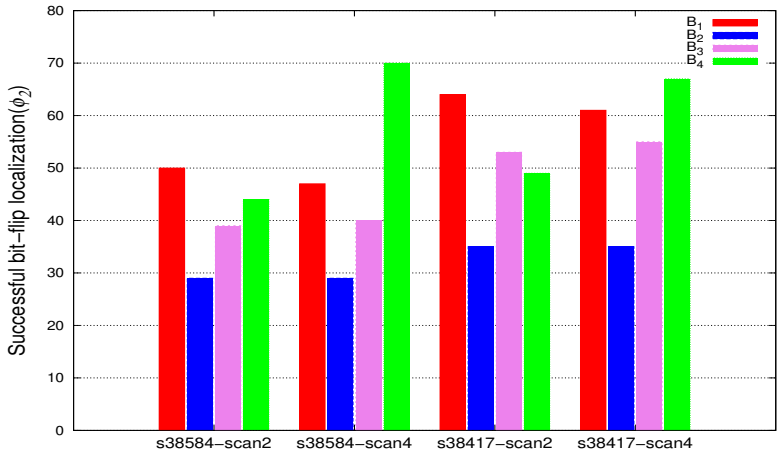

Figure 4.20: Successful bit-flip localization variation for different scan configurations in temporal tracing

respectively. Since temporal tracing introduces variability in dumped signals, the amount of signal state restoration significantly enhances when L_{sc} is increased from 2 to 4. Due to this in scan4 configuration (L_{sc}=4), B_4 performs better than the other three formulations for both the circuits. We observed similar trends while experiments on other circuits also. Temporal tracing has a significant impact on the metric ϕ_9 also. For instance- we observed a maximum decrease of 90% in ϕ_9 values for $b21$ circuit when the tracing configuration is changed from scan2 to scan4 for bit-flip errors. It is worth to note that because of the change of inputs, the overall success in bit-flip localization (ϕ_2) values shown in Figure 4.20 are lower than those of Figures 4.7 and 4.8.

Since the circuits in Table 4.4 can be considered as blocks within a complex SoC, with a fixed trace buffer width of 64, four such blocks can be traced. Thus, in addition to an appreciable increase in signal restoration with this observability enhancement scheme [22], SAT-based error localization also improves. Moreover, in industrial designs, it is common to have distributed trace buffers wherein each trace buffer can be connected with a different type of temporal tracing depending on the circuit. We performed the temporally variable tracing for stuck-at errors by considering L_{sc} as 2 and 4 for two circuits, $b21$ and $s13207$. Similar to the analysis presented previously for bit-flip errors, we consider TBw as 16. Because of temporal tracing, 32 and 64 flip-flops can be traced albeit with non-continuous dumping into the trace buffers. By definition, ϕ_9 should be

as low as possible to meet the targets of exact localization.

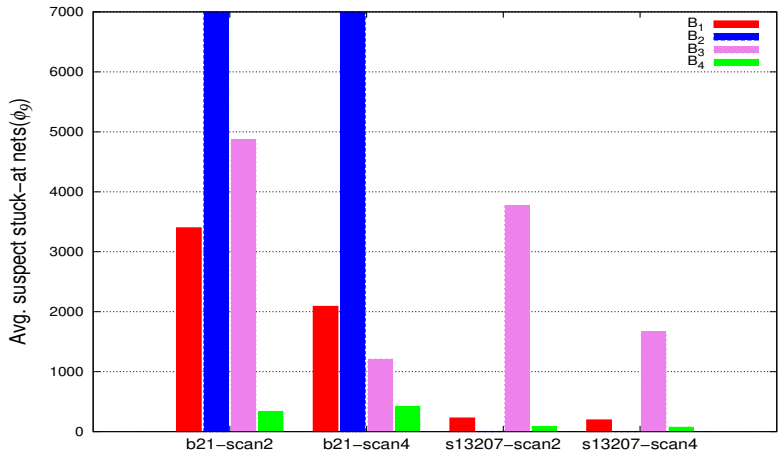

Figure 4.21: Avg. suspect stuck-at nets variation for different scan configurations in temporal tracing

We did not observe any major change in ϕ_8 with change in L_{sc}. Due to temporal tracing, B_4 provides lower or comparable ϕ_9 values when L_{sc} is changed from 2 to 4. B_1 and B_3 also follow the similar trend.

4.5.8 Summary of Variation in Localization Results with Different SAT Formulations

Since different SAT formulations have different type of constraints in them, the error localization efficiency varies significantly as the nature of UNSAT core(s) obtained after their solution differs from one another. Major observations regarding the utility of different SAT formulations can be summarized as follows:

1. Trace signal selection has no impact on localization with B_2. For few circuits (s38417, s38584, s9234), the remaining three formulations perform better than B_2. However, for other circuits (mainly of opencores and ITC'99 type), B_2 succeeds in error localization in a more effective manner compared to other three formulations.

2. B_4 performs best in error localization for some circuits, justifying the contribution of restored signal states in error localization. B_4 provides lower average flip-cycle

(ϕ_3) compared to B_3 which means that the flip-cycle candidates are closer to the actual flip-cycle with former formulation compared to the latter formulation (B_3).

3. For all the circuits, the number of UNSAT attempts with B_2 is less that of other formulations which means that error does not propagate to primary outputs (PO) in those cases. In most cases, B_1, B_3 and B_4 always result in UNSAT.

4. In cases of inexact localization, compared to other formulation, B_4 provides lower topological distance (ϕ_7) of flip-flop suspects from the actual flip-flop. Where B_2 has lower ϕ_7 than B_4, it is because the former (B_2) has lower number of attempts turning UNSAT than the latter (B_4).

5. The behavior of different SAT formulations is similar for bit-flip and stuck-at errors.

Hence, with different SAT formulations, fine-grained electrical error localization can be achieved with varying degree of success. In spite of scalability limitations of SAT solvers, we believe that SAT-based debugging methodology in post-silicon environment can be applied to any large circuit provided the length of error trace is reasonably smaller. In the next section, we discuss in detail the impact of various steps of the proposed methodology on the scalability issue.

4.6 Addressing Different Scalability Issues in SAT-based Error Localization

4.6.1 Scalability of Design Unrolling Step

We measured the increase in the formulae sizes(i.e., number of clauses in B_j) over the formula size corresponding to the unrolled netlist(UD) for different SAT formulations. For dmu_{clu} circuit, the relative increase (over the formula for only the unrolled netlist) corresponding to B_1, B_2, B_3 and B_4 is 4%, 4%, 0.1% and 8% respectively. For usb circuit, the similar increase for different formulations stand at 0.8%, 0.6%, 0.1% and 0.9% respectively. Note that the above computations are done for $T = 100$ and $TBw =$

32. Thus, it can be observed that the increase in formulae sizes is reasonable for large circuits. Other benchmark circuits also follow same trend for any selection of S_{Tr}.

4.6.1.1 Time Spent in Unrolling

Time spent in the design unrolling stage is not very critical for applying the methodology to large circuits or applying it to long error traces. This is because time-frame expansion corresponding to a certain length of cycles needs to be done only once. Thereafter, different type of SAT formulations (B_j) can be repeatedly applied on the expanded netlist (UD). Different type of error traces (obtained by changing input assignments etc.) can be applied as constraints on the same expanded netlist for SAT solving. We

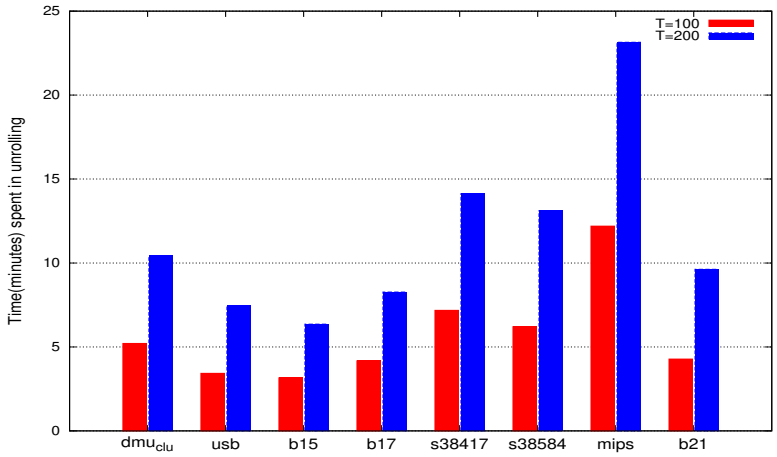

Figure 4.22: Time spent in design/netlist unrolling

observed a linear increase in the time required for unrolling of the design netlist when the error trace length (the number of clock cycles) is doubled. For the larger benchmark circuits, the time required to unroll for 100 and 200 cycles is shown in Figure 4.22.

4.6.1.2 Memory Usage in Unrolling

Figure 4.23 shows the memory usage (in MB) during time-frame expansion for the larger circuits for a error trace length ranging from 50 to 1000 cycles. When the time-frame expansion duration is increased from 500 to 1000 cycles, there is less than double increase in the memory usage. The same trend is observed for all the circuits when T is increased

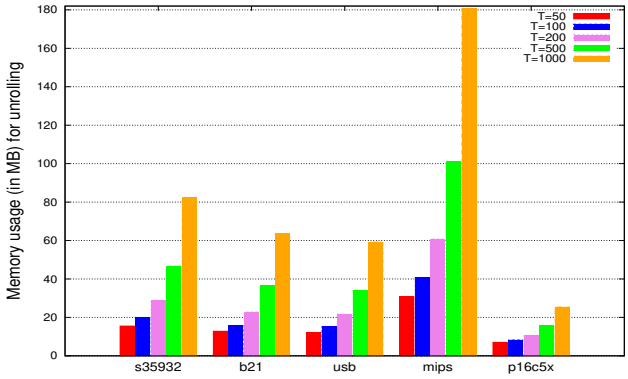

Figure 4.23: Memory usage in design/netlist unrolling

from 50 to 100 or 100 to 200. This shows that time-frame unrolling step can be completed even for larger circuits for debug scenarios with T exceeding 1000 cycles also.

4.6.2 Scalability of SAT Solving Step

To address the scalability issue further, we assessed the variation in SAT solver efforts in solving for all the circuits. Figures 4.24 and 4.25 denote the average time spent (in seconds) during SAT solving step with different formulations and the memory usage (in MB) respectively for all the benchmark circuits. The increase in time and memory usage

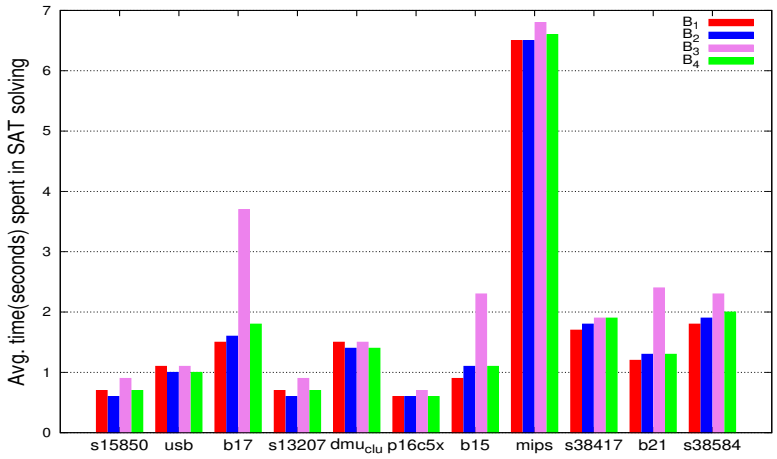

Figure 4.24: Average time usage(in SAT solving)

is linear with respect to the circuit size. When we doubled the error trace length (T) to 200, the time and memory usage were observed to be close to double the time and memory required for $T = 100$ (Figures 4.24 and 4.25). Thus, time and memory usage increase

is linear with respect to error trace length also. Nevertheless, one of the restrictions of this debugging methodology is the requirement of unrolling the netlist into time-frames equal to the length of an error trace. Hence, if a shorter error trace can be obtained, the time-frame unrolling exercise is less computationally intensive. However, this exercise needs to be done only once. Thereafter, different SAT formulations can be attempted with different constraints depending on the debug scenario. We did not observe any

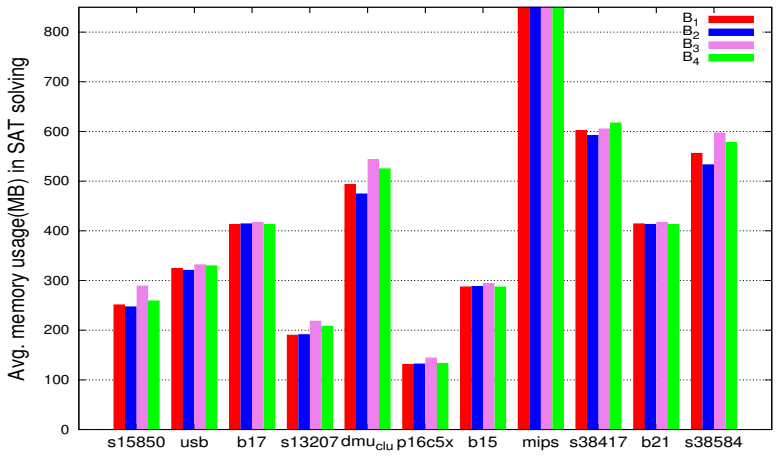

Figure 4.25: Average memory usage(in SAT solving)

appreciable difference in the time and memory usage across all the benchmark circuits for SAT solving when the trace signal selection technique is varied. Additionally, there is not much difference in time and memory usage between bit-flip and stuck-at error localization scenarios. We believe that satisfiability-based formulation would be successful for error localization of logic modules even if a directed trace signal selection(such as that of [34]) is not utilized. However, suitable inputs must be provided to the SAT formulations. In addition to this, the internal signals (such as control signals) must be set properly so that formulations (other than B_2) succeed in error localization. We are currently working in this direction by a proper analysis of the circuit structure. For design components having a large number of sequential elements such as memories/caches, time-frame unrolling may pose a severe bottleneck as the expansion requirement can range from 5000 to 10000 cycles or more. However, such structures are generally not prone to bit-flip errors and are already provided with suitable detection and correction mechanisms.

4.6.3 Analysis of Localization in Large Error Traces

Error traces of length (T) equal to 500 and 1000 clock cycles were divided into 5 divisions(k) resulting in smaller error traces of 100 and 200 clock cycles respectively. We injected a bit-flip randomly somewhere between the first and T^{th} cycle. As explained in Section 4.4.1, the SAT instances are created for these smaller divisions and provided to the solver. It is obvious that these instances (Boolean formula, B_j) corresponding to some divisions($eTrace_j$) would be satisfiable. However, in the division which contains the flip-cycle, the SAT instance turns out to be unsatisfiable from which the UNSAT core is to be extracted. The translation of these UNSAT core provides the suspect flip-flops and flip-cycles. After division into smaller traces, the time and memory usage was comparable to that of Figures 4.25 and 4.24.

4.7 Comparative Evaluation with Previous Silicon Debug Methods

4.7.1 Post-silicon Debug Methods at Architecture-level

One of the notable techniques in post-silicon error localization is the methodology presented in *Instruction Footprint Recording and Analysis*[69] work which performs error localization at block-level granularity in processor systems using low cost on-chip recorders as an observability mechanism. However, typically an architectural block is quite large which renders localization at the netlist level very difficult with such techniques [66][69]. Therefore, additional methodologies are needed for fine-grained error localization of the processor cores. Further, for achieving exact error localization of other digital blocks (peripherals, uncore components etc. in a complex SoC) from the collected error traces in an automatic manner, we require other approaches as lower level design information is missing. In absence of golden response, satisfiability-based method can assist in better bug localization as it achieves so in a self-checking manner. Therefore, with the

satisfiability-based technique, the methodology of IFRA (or, QED[66]) can be complemented to achieve exact error localization after a shorter error trace is obtained with these architecture-level validation methodologies.

Table 4.5: Comparison with other SAT-based localization methods

Method	Type of error	Spatial localization	Temporal localization	Trace width	Length of error trace	Signal selection
[34]	bit-flip	within 20 to 40 FFs	4 to 20 cycles of flip-cycle	128, 256	100 cycles	bit-flip detection-driven
[101] (pre-silicon)	RTL errors	max. 3% of modules	10 to 15% of total trace	groups of 16 registers	2,4,8,16 cycles (session-based)	design heirarchy-based
[48]	stuck-at	actual net/wire	-	5% of all signals	upto 2000 cycles	random selection
Proposed	bit-flip, stuck-at	mostly <10 to 20 flip-flops	within 1 to 20 cycles	16, 32, 64	partitions of 100, 200 cycles	graph-based clustering

4.7.2 SAT-related Post-silicon Error Localization Methods

The proposed signal selection technique can be applied in conjunction with the debug methodology of [101], [48] and [100] for design errors also. A qualitative comparison of SAT-based gate-level localization techniques with the proposed methodology is reported in Table 4.5.[14] Some of the trace signal selection techniques which have attempted trace signal selection aimed at error localization in the post-silicon environment[33, 50] are only for design errors. A dynamically reconfigurable selection technique for detecting bit-flip errors was proposed by Basu et al.[111] using an arrangement of few multiplexers. Sabaghian-Bidgoli et al.[147] proposed probability-based error calculation methods to

[14]Based on the reported results in the respective works. For the proposed approach, the temporal and spatial localization numbers reported have been obtained by averaging the results of s38417 and s38584 circuits. Note that area overhead of the techniques is proportional to the trace width. Bug localization accuracy is reported in temporal and spatial localization columns of Table 4.5. The length of error trace for which the respective techniques are best suited to, are reported in the second last column. Ideally, the length of these error traces should not be very small (i.e., < 100 cycles) as acquisition of such small traces may not be feasible for large designs.

improve SAT-based fault diagnosis of stuck-at nets. One of the merits of the method in [147] is that it can be applied to both pre-silicon and post-silicon debugging. This methodology can be utilized in conjunction with the proposed technique for effective post-silicon error localization as the trace signal selection can be enhanced through probability analysis. Signal selection techniques such as [51, 55] can also be utilized along with the graph-based clustering to obtain a refined list of trace signals. Particularly, a combined signal selection approach like that of [33, 116] can be attempted with a mixing of signals from previous techniques of [1, 9, 102] and the graph-based topological analysis for obtaining trace signals [23, 142].

4.8 Conclusion

This chapter proposed a signal tracing methodology for satisfiability based error localization in the post-silicon environment. A grouping based methodology is presented for signal selection and it is used with different observability enhancement schemes to obtain a smaller execution trace. The grouping methodology is intended to minimize the trace buffer width. The error trace obtained from chip execution is forced on the time-frame unrolled circuit for satisfiability solving as constraints. For the larger error traces ranging in the range of thousands of clock cycles, SAT-based debugging can run into memory and solving time issues. To alleviate this, the error trace was partitioned into smaller traces. Experiments on a variety of benchmark circuits show that SAT-based debugging methodology succeeds in the post-silicon environment for two kinds of error scenarios.

– ∗ – ∗ –

Chapter 5

Effective & Combined Trace Signal Selection

5.1 Introduction

As stated in the earlier chapters, one of the key challenges during post-silicon validation of the first released silicon is the limited observability of internal states of the design. Debug of design and electrical errors in the post-silicon environment becomes very difficult due to the poor accessibility of internal signals. Scan chains help in enhancing the observability and controllability for the purpose of manufacturing testing. However, they are less effective during post-silicon validation and debugging as the chip execution has to be halted to enable read-out of the states of the flip-flops. To achieve a non-destructive reading of internal states, trace buffers are incorporated on-chip which store few signals for a limited number of clock cycles as the chip execution progresses [45]. However, given the constraint of area overhead, only a small set of signals can be selected for tracing. These signals are decided at the design stage itself. This increases the difficulty of selection of appropriate trace signals. For small designs, the design engineers can suggest a few important signals based on the knowledge of the circuit behavior. However, there is a need for automated signal selection for large designs. Signal state restoration enables increased visibility as some of the untraced states can be inferred from the

traced flip-flop values [114]. This technique has been the guiding principle in most of the proposed solutions for automatic trace signal selection [1, 9, 50, 102], although none of these methods report the efficacy of their selection techniques from the point of localizing or detecting the bug/error. Therefore, devising signal selection with the mere objective of maximization of signal state restoration is not a very good choice [55, 96, 116]. The primary reason behind this lies in the fact that the restoration-aware signal selection techniques favor signals that may lead to little value addition to the debug process.

Some approaches have presented signal selection techniques based on the ease of error detection [34, 50, 55, 111, 148], error localization with the restored and traced states has not yet been explored in detail. By and large, the signal state restoration-based trace signal selection techniques are not sufficient for debugging different kinds of functional and electrical errors [55, 116]. Thus, trace signal selection process should aim at objectives such as enhancement in the debug process instead of focusing only on maximization of the internal state visibility. Let's consider a design having few large arrays of registers in its netlist structure. Prevalent trace signal selection techniques would prefer selecting elements from those large arrays as restorability is highly favored with this selection because of logical connectivity. However, these signals (i.e., register elements) are not very useful for debugging scenarios because of their simple logical relationships. This necessitates a trace signal selection methodology aimed at identifying signals most useful for error localization. Additionally, as stated previously, the signal state restoration-based signal selection algorithms rely on the assumption that restoring higher number of untraced signal states results in a larger quantity of debug data which may be useful for localizing the incorrect functional implementation. However, as is clear from the above example scenario, this assumption is not true as signals favorable for restorability maximization are not very effective in error detection/localization [34, 51, 55, 96, 116].

We present an approach of trace signal selection for observability enhancement in terms of their effectiveness for detecting the bug and the latency with which its effect can be observed at the traced flip-flops. A methodology of signal selection is proposed

which takes into account the combinational paths between the flip-flops and their topological order. The main benefits from this signal selection methodology are reduced error detection latency and an increase in the number of times the traced flip-flops capture the erroneous responses. Trace buffers incur area penalty and may lead to routing congestion. A routing-aware signal selection technique can help reduce the associated routing overhead compared to only restorability maximization-based signal selection techniques [107]. Hence, another important consideration is to take into the ease of routing of trace signals to the trace buffers. To comply with off-line dumping constraints, trace buffers can not be arbitrarily placed closer to the traced signals. Due to this, some trace signals could not be routed. For instance- we observed in our experiments that for $s35932$ circuit, as many as 7 (out of total 32) trace signals obtained from technique outlined in [1] can not be routed to the trace buffers when routing is carried out within specific constraints (such as congestion limit and wire length budget). We propose a trace signal selection methodology considering the combination of three important parameters-enhancement of signal restoration-based on the traced signal states, increasing error detection/localization with the traced (and restored) signal states and routing overhead reduction (in terms of wirelength).

The remainder of this chapter is organized as follows. The proposed methodology of topology-based signal selection is explained in detail in Section 5.2. Experimental formulation and results on the topology-based signal selection are discussed in Section 5.3. Section 5.4 puts forward the preliminaries behind our combined trace signal selection approach. Section 5.5 presents the proposed 2-Parameter combined signal selection. The proposed routing-aware signal selection technique is mentioned in detail in Section 5.6. Section 5.7 presents the proposed 3-Parameter combined signal selection methodology. Section 5.8 describes the proposed error localization methodology using the expanded internal visibility. Section 5.9 elaborately analyses the routing and error localization results obtained for benchmark circuits. Section 5.10 finally concludes the chapter.

5.2 Proposed Methodology of Topology-based Trace Signal Selection

The restorability-based signal selection approaches consider whether untraced signals are recoverable from the traced ones or not. However, the error propagation capability of the flip-flops is generally not considered in these approaches.[1] The propagation of an error to flip-flops (internal signals) depends on the presence or absence of combinational path(s). We exploit this principle for our methodology by considering the circuit topology. For the proposed[2] approach, a *S-graph* needs to be constructed to analyze the propagation of bugs from one flip-flop to another flip-flop of the design. Note that actual sensitization of bugs is a factor of the gates present on combinational paths. However, analysis of these gates makes the methodology tedious as computation of conditional probabilities of signal transitions needs to be performed. In the *S-graph*, flip-flops are represented as graph nodes and combinational paths between them as directed weighted edges. Apart from considering flip-flop interconnections, we take into account their topological ordering too. Accounting for the order of flip-flops in the design structure helps in the direct optimization of error detection and the latency. To ensure that error detection latency is minimized, the traced flip-flops must capture the error within few clock cycles.

We make two propositions here. First, it is not necessary that the flip-flops in the vicinity of primary outputs (PO) are traced as the effects of the bug can be captured at the PO although with a higher latency. Second, the flip-flops in the proximity of primary inputs can be excluded from the set of candidate trace signals as the flip-flops succeeding them can observe the erroneous behavior. Thus, after performing a topological sort of the candidate trace signals (*S-graph* nodes), a list (of flip-flops) can be obtained which

[1]Based on experiments with ranking of trace signals through detectability preference of randomly injected errors, reported results in [149] and [55] show that trace signals selected for restorability maximization fail to detect errors in many iterations.

[2]Note that the signal selection approach outlined in Chapter 4 could also have been used here. However, because of clustering, chances are there that design bugs may go undetected. Indeed we had attempted error localization with signals from Chapter 4 in the experiments conducted in this chapter. We observed that clustering-based signals perform inferior. The primary reason we believe is the difference in the modeling of errors in case of bit-flips and gate-level design errors.

is to be pruned until the number of elements in the list is comparable with width of TB (TBw). The objective behind above propositions is to select flip-flops which can capture the maximum number of erroneous responses (due to bugs) as early as possible. We aim at a joint optimization of both these factors through pruning of *S-graph* nodes. However, this approach has a major limitation since topological sort can be performed only on acyclic graphs whereas practical circuits often contain cycles. Due to this reason, we group flip-flops forming cycles in *S-graph* as one node and make it a directed acyclic graph (DAG). Any of the constituent flip-flops can be selected for the purpose of tracing as a representative of this node. This definitely has implications on the bug propagation and in turn bug detection. However, this serves as a crude solution to the limitation incurred by topological sort in case of cyclic graphs.

The proposed methodology can be broadly divided into four steps: (a) *S-graph* creation of design ($GraphCreate$), (b) Score assignment of flip-flops ($ScoreAssign$), (c) Arranging flip-flops based on their order, and, (d) Trace signal selection ($SelectSignal$) based on *score*.

5.2.1 S-graph Creation and Score Calculation

The *S-graph* (represented as G) construction steps are outlined in Algorithm 5. A node in a graph G has a certain number of weighted in-coming and out-going edges. The weights on edges account for the different paths through which two flip-flops are connected. Note that $Convert$ function in the above algorithm leads to building G as a directed acyclic graph by grouping those flip-flops which form a cycle into an individual node. This node contains a number of flip-flops (FFs), one out of them can be selected for tracing depending on the *score* of this node in comparison with other nodes. For large cycles (i.e., cycles involving a large number of nodes), $Convert$ function in the above methodology breaks the cycles. To achieve this, $Convert$ function utilizes a threshold (N_{th}) can be decided in term of the number of nodes in the S-graph. For the purpose of flip-flop selection, we need to assign priority to different nodes through *score* calculation. The steps for this is outlined as Algorithm 6. Although other scoring mechanisms can

Algorithm 5: *GraphCreate*

Input: *Design*
Output: G

```
1  FFtot ← total flip-flops in Design;
2  for each FF F_i in Design do
3  |   FFrem ← FFtot - F_i;
4  |   for each FF F_j in FFrem do
5  |   |   n=0;
6  |   |   if path exists from F_j to F_i then
7  |   |   |   n = n + 1;
8  |   |   end
9  |   end
10 |   wt ← n;
11 |   n = 0;
12 |   connect F_j to F_i by an edge;
13 |   weight of edge ← wt;
14 end
15 G' ← directed weighted graph of Design;
16 G ← Convert(G');
```

be attempted, this calculation of *score* is aimed at the ranking of signals through the connectivity of each trace signal with other signals (i.e., flip-flops).

Algorithm 6: *ScoreAssign*

Input: G
Output: *scoreList*

```
1  for each node N_i of G do
2  |   E_in ← no. of in-coming edges;
3  |   wt_i ← weight of in-coming edge;
4  |   E_out ← no. of out-going edges;
5  |   wt_o ← weight of out-going edge;
6  |   score = E_in*(Σ wt_i) + E_out*(Σ wt_o);
7  |   scoreList ← score of each node;
8  end
```

The flip-flops are assigned *score* for the purpose of ranking in terms of error collecting ability. The *score* is determined by the total number of connections with the other flip-flops of design (i.e., remaining nodes of G). For obtaining the total connections, we multiply the number of in-coming and out-going edges (i.e., E_{in}, E_{out}) with their

respective weights (i.e., wt_i, different number of parallel paths between a pair of FFs). The worst case time complexity of the *score* assignment procedure (Algorithm 6) is $O(n_{path})$ where n_{path} is the number of paths in the S-graph.

5.2.2 Arranging Flip-flops and Trace Signal Selection

As per our propositions, after topological sorting of flip-flops (or nodes), this list (represented by *Glist*) has to be pruned down from both the ends (corresponding to PI and PO ends of the actual circuit). To account for this, a pruning factor (pf) is introduced. We describe next the methodology of obtaining the optimal pruning factor for any circuit.

5.2.2.1 Methodology for Obtaining Pruning Factor

The factor, pf represents the nodes which are to be removed from the *S-graph* nodes list (*Glist*). Thus, pruning factor (pf) plays a vital role in selection of candidate trace signals (FFs). The algorithm $Findpf$ (outlined below) searches for the optimal pruning factor in a fine-grained manner.

- Algorithm 7 has two user-defined inputs: ipf (initial pruning factor) and $pfinc$ (pruning factor increment) which are to be assigned based on the level of granularity desired.
- The algorithm begins with a complete list of topological list of *S-graph* nodes ($TempList$) and depending on ipf (which is first pf), pruning is done for the first time (i.e., some nodes get removed from both ends of the list).
- Thereafter, with the remaining list ($TempList'$), the *score* calculation is done for these nodes and the average *score* (labeled as $Normscore$) is obtained.
- Depending on the chosen $pfinc$, pf gets incremented (pf+$pfinc$) and the process continues till $TBwidth$ becomes comparable with the number of nodes remaining in the list ($TempList$). At the end, the pruning factor chosen for the design is the pf that provides highest $Normscore$.

It is worth to note that the cycles in the S-graph (G) do not actually inhibit the error propagation as some of the nodes connected to the ones (in the cycle) can capture the erroneous behavior through any of the combinational paths between them. The implementation of *topologicalsort* in Algorithm 7 takes into account the cyclic nature[3] of the created S-graph (G) and then provides the topological order of nodes ($Glist$). The worst case complexity of *topologicalsort* is $O(V + E)$ where V and E denote vertices and edges respectively in the S-graph.

Algorithm 7: *Findpf*

Input: G,$TBwidth$,$scoreList$
Output: pf

```
Glist ← topologicalsort(G);
TempList ← Glist;
pf ← ipf;
while TempList > TBwidth do
    TempList' ← TempList pruned by pf;
    Totscore = Σ(score of entries of TempList');
    num ← number of entries of TempList';
    Normscore = Totscore / num;
    List ← store pf and Normscore;
    pf = pf + pfinc;
    TempList ← TempList';
end
pf ← pf for which Normscore is highest;
```

As an analogue to pruning factor term, "window size" can be defined as set of flip-flops of the list $Glist'$ after the original flip-flop list ($Glist$) is pruned by a certain pf from both ends. Thus, **window size (ω)** of 100%, 80%, 60% and 40% means that the pf is set at 0, 0.1, 0.2 and 0.3 respectively. ω of 100% means no pruning of the *S-graph* nodes. Note that the topological sorting must be done before the pruning exercise. With the combination of pruning and *score* assignment, the proposed technique attempts to capture a large number of bugs within a lesser number of clock cycles. The final step to select the trace signals depending on the topological order is shown as Algorithm 8.

[3]This surely serves as a limitation of the proposed approach prohibiting its applicability to all kinds of designs/circuits.

After pruning of topologically ordered nodes, the left-over nodes are ordered by *score*. As per requirement (i.e., TBw), signals having highest *score* are to be selected. The

Algorithm 8: *SelectTopoSignal*

Input: $Glist$, pf, $scoreList$, $TBwidth$
Output: $FFlist$

1 $score \leftarrow score$ of each node from $scoreList$;
2 Prune $Glist$ as per pf and obtain $Glist'$;
3 Sort $Glist'$ by respective $score$ to get $FFlist'$;
4 $FFlist \leftarrow$ entries from $FFlist'$ as per $TBwidth$;

proposed methodology is illustrated for an example circuit in the next section assuming trace buffer width (TBw) as 2.

We analyzed derivation of pf for 3 benchmark circuits with ipf as 0 and $pfinc$ as 0.05. As per Algorithm 7, pf is to be iteratively searched such that highest **Normalized Score** ($Normscore$) is achieved. Figure 5.1 shows the variation of $Normscore$ with ω. $Normscore$ is observed to be maximum when ω is 60-80% for all the three benchmark circuits. Thus, maximum error detection with less clock cycles is expected if flip-flops of this window size (ω) are traced. The selection of flip-flops as per given TB width with in this range of flip-flops is decided by their individual *score* obtained by $ScoreAssign$. The results pre-

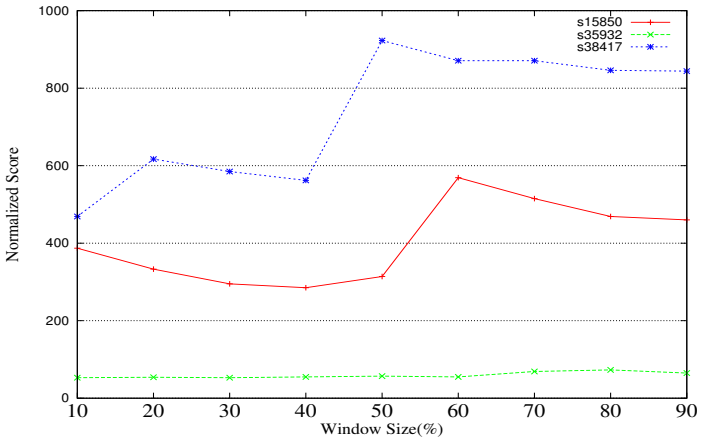

Figure 5.1: Variation of Normalized Score with Window Size

sented in Table 5.2 are as per the nature of variation of $Normscore$ reported in Fig. 5.1. Thus, central to the success of idea of topological pruning is the derivation of optimal

pruning factor (which can be easily translated to window size, ω). Note that wide variation in Normalized Score ($Normscore$) among these three circuits is because of varying number of connections between the flip-flops for each circuit.

5.2.3 Topological Selection Methodology Illustration

We consider a small design with eight flip-flops. For brevity in presentation, we consider the simple case of a directed acyclic graph. The *S-graph* (G) created for this design[4] is shown in Fig. 5.2. The graph has eight nodes, numbered FF1 to FF8 (corresponding to each FF i.e., trace signal).[5]

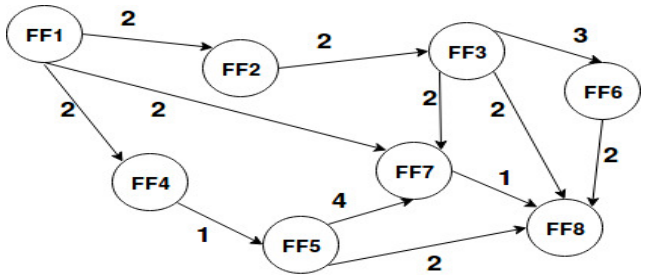

Figure 5.2: *S-graph* for illustrating proposed methodology

Different steps of the proposed methodology are described below and their results are reported in Table 5.1.

- By the *score* assignment process outlined in Algorithm 6, the corresponding *score* of FF1-FF8 are 18, 4, 23, 3, 13, 5, 23 and 28 respectively.
- For different nodes, values of *score* computed by Algorithm 6 are shown (in decreasing order) in third column of Table 5.1.
- For the *S-graph* (Fig. 5.2), topological order is given by FF1, FF4, FF5, FF2, FF3, FF7, FF6, FF8 (second column of Table 5.1).
- As pruning starts (pf increases), window size (ω) shrinks, we obtain pruned list of nodes which are ordered as per their *score* (third column of Table 5.1).

[4]This design is a slightly modified version of $s208$ ISCAS'89 benchmark circuit.
[5]The flip-flop numbers depicted as FF1 to FF8 are not themselves indicative of any sort of ordering.

Table 5.1: Illustration of proposed methodology

Window Size(%)	topological order	arrangement by *score*
100	FF1,FF4,FF5,FF2,FF3,FF7,FF6,FF8	FF8,FF7,FF3,FF1,FF5,FF6,FF2,FF4
80	FF1,FF4,FF5,FF2,FF3,FF7,FF6	FF7,FF3,FF1,FF5,FF6,FF2,FF4
60	FF4,FF5,FF2,FF3,FF7	FF7,FF3,FF5,FF2,FF4
40	FF5,FF2,FF3	FF3,FF5,FF2

- The pruning stops when ω reaches 40% as number of left-over nodes is comparable to $TBwidth$ (=2). Thus, FF3 and FF5 are selected as trace signals since their *score* is greater than the other signal (FF2).

Note that ω as 40% means that 0.3 is value of pruning factor (pf) as per Algorithm 7.

5.3 Experimental Formulation and Results for Topology-based Selection

5.3.1 Description of Evaluation Metrics

For evaluating the efficacy of the proposed trace signal selection (TSS) with respect to restorability techniques, following 3 parameters are identified for each trace signal: (a) State restorability (α), (b) Error detection latency (β), and (c) Number of bug injections not detected (γ).

As explained earlier, from the viewpoint of error localization/detection, restorability approach is inefficient. However, principles of signal state restoration can be applied on the signals selected from the proposed technique too. With the proposed technique, the visibility of internal states is enhanced albeit lesser than techniques which particularly aim at state restorability [1, 9, 50, 102]. To evaluate how a diminished state restorability relates to the number of errors detected and their latency, following 3 parameters can be derived from the parameters considered above:[6]

[6]SRR (State Restoration Ratio) is mentioned simply as RR (Restoration Ratio) also at some places in this thesis.

1. $\Delta\alpha = \alpha\{SRR\ based\ TSS\}$ - $\alpha\{proposed\ TSS\}$
2. $\Delta\beta = \beta\{SRR\ based\ TSS\}$ - $\beta\{proposed\ TSS\}$
3. $\Delta\gamma = \gamma\{SRR\ based\ TSS\}$ - $\gamma\{proposed\ TSS\}$

$\Delta\alpha$ is the difference between signal state restoration ratio for the two set of selected trace signals, averaged over the TB width (which is considered as 32 in our experiments). Similarly the other two parameters, $\Delta\beta$ and $\Delta\gamma$ report the difference for error detection latency (per error) and number of error (bug) iterations not detected by each trace signal. The trace signals are selected by two methods, restorability approach ($SRR\ based\ TSS$) and the proposed approach ($proposed\ TSS$). For the purpose of comparison, we consider region growth based heuristic trace signal selection [1] which aims at maximizing SRR as a representative of $SRR\ based\ TSS$.

5.3.2 Evaluation Results

The three derived parameters (1,2 and 3) are evaluated for 1000 iterations of error injection through random gate replacements. Table 5.2 presents the evaluation of these parameters. The **Window Size (ω)** has been kept at 60-80% for the evaluation of these 3 parameters. The salient observations are listed as below:

Table 5.2: Evaluation of parameters on different circuits

$Circuit$	$TotalFF$	$TBwidth$	$\Delta\alpha$	$\Delta\beta$	$\Delta\gamma$
s15850	534	16	0.062	7.35	191
		32	0.059	15.61	157
s13207	638	16	0.396	8.69	423
		32	0.136	9.36	536
s38584	1426	16	0.961	3.88	152
		32	0.474	3.85	217
s38417	1636	16	-0.271	5.69	491
		32	-0.188	7.43	529
s35932	1728	16	-0.046	40.86	229
		32	-0.245	17.04	393

- Results indicate minuscule difference in restorability per signal, $\Delta\alpha$ while $\Delta\beta$ and $\Delta\gamma$ values are fairly large.

- Negative values of $\Delta\alpha$ in Table 5.2 mean that the signals with the proposed approach has higher restorability than restoration based approach [1].[7]
- For s38417 circuit, the signals with proposed methodology detected the erroneous behavior for all 1000 error injection iterations (γ=0).
- Note that an effective trace signal selection should give low values for both the parameters: β and γ. In Table5.2, positive values of $\Delta\beta$ and $\Delta\gamma$ parameters indicate their low values for β and γ with the proposed methodology.

We performed another set of experiments on a wider range of circuits for finding out the difference in average detection latency (ϕ_0) for the proposed technique and the methodology presented in [1] (depicted here as **RATS**) for TBw as 32. The metric[8] utilized for this evaluation is shown below (total number of iterations is represented as T_{iter}, considered here as 100):[9]

$$\chi = \sum_{i=0}^{i=TBw} first\ cycle\ in\ which\ each\ signal\ detects\ error \tag{5.1}$$

$$Avg.\ cycle = \frac{\chi}{T_{iter}\ -\ iterations\ with\ no\ error\ detection} \tag{5.2}$$

As can be observed from Figure 5.3, with the proposed method achieves lower values of average cycle of error detection as compared to the technique in [1] for all circuits except $s38584$ circuit. The number of iterations in which there is no error detection by trace signals (which is denoted by denominator in Equation 5.2) for [1] is lower than that of the proposed methodology. Out of 100 iterations, trace signals with the proposed methodology succeeded in 78 iterations while the technique in [1] succeeds in only 42 iterations. The same trend can be observed from the values reported in Table 5.2.

[7]Since the computation of parameter results is done per signal, the results corresponding to trace buffer width of 32 and 16 do not correlate among themselves.

[8]We devise this as the previous metric β does not capture the iterations in which the bug/error detection fails by all the trace signals

[9]We consider 50 iterations of random wire exchange and 50 iterations of random extra inverter insertion.

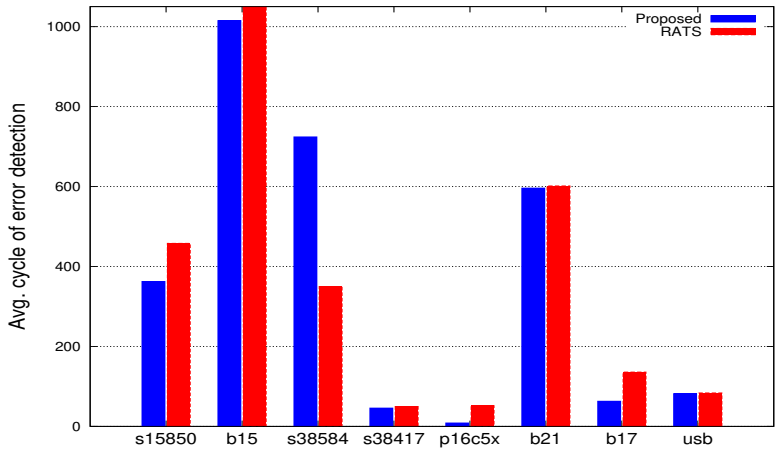

Figure 5.3: Avg. cycle of error detection for different circuits

A weighted combination of the individual parameters (i.e., β and γ) for signal selection metric can serve as a better option. We believe that deriving the particular weights is not trivial and it would depend on the debug process. For instance, bugs in some of the sub-modules of design may be required to be debugged early so as to enable understanding of the bug at the complete level of the design. However, in practice, identification of such sub-modules of a large and complex design is not an easy exercise. To solve the issues of combined evaluation of any signal selection, a better approach would be to consider different choices during the process of signal selection itself. As explained earlier in this chapter, trace signal selection can depend on a wide range of factors. Therefore, we propose a combined trace signal selection methodology that takes cognizance of multiple factors while deciding the final set of trace signals.

5.4 Signal Selection with Combination of Preferences

As stated in Section 5.1, we aim the combination of three different preferences for the trace signal selection. The combined signal selection methodology focuses on the selection of a fixed number of trace signals from the three groups of ranked signals: S_{res} (signals leading to higher state restoration), S_{ed} (signals capable of maximum error detection) and S_{ra} (signals which incur minimal routing overhead). The combination of these three lists (which are separately ranked) can be described through the following equation:

$$Selected\ Signals(S) = a * S_{res} + b * S_{ed} + c * S_{ra} \tag{5.3}$$

The three parameters: a, b, c in the above equation are defined by the designer (and, debug engineer) and these weights represent the importance of each parameter (*signal restoration*, *error detection* and *routing-awareness*) in the trace signal selection process.

While a separate consideration of each parameter provides a list of trace signals highly profitable for the maximum enhancement of the respective parameter, the proposed trace signal selection methodology attempts a judicious mixing of signals from each of the three sets (S_{res}, S_{ed}, S_{ra}). The overall flow of the combined signal selection methodology is depicted in Fig. 5.4.

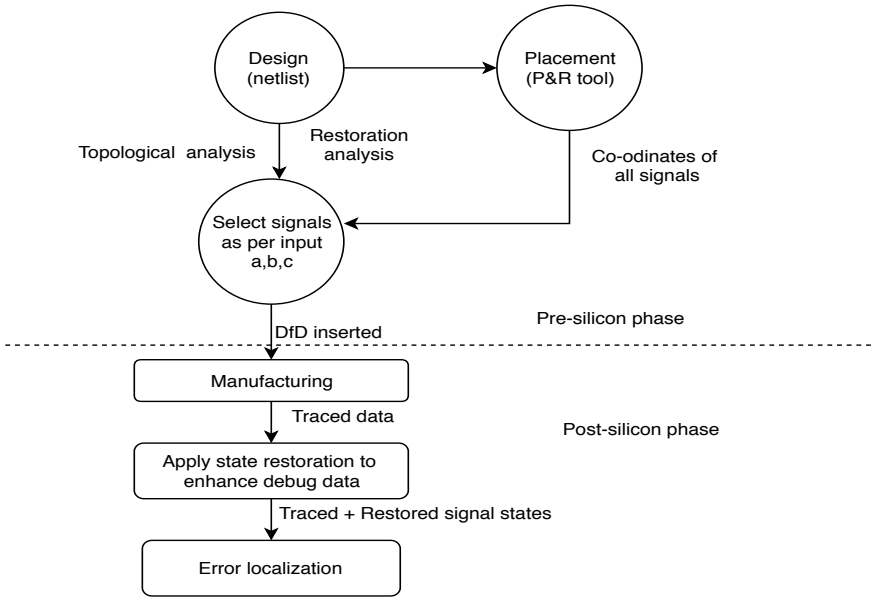

Figure 5.4: Overview of the proposed methodology of trace signal selection and debugging during post-silicon validation

As explained previously, existing research on trace signal selection seldom analyzes the impact of the selected trace signals on the process of post-silicon debug, which makes the signal selection very superficial. We address this issue (as shown in Fig. 5.4) in later sections of this chapter. In addition to a signal selection methodology, we attempt error localization (at netlist level) by analyzing the restored and traced internal

signal states of the design.[10] Specifically, our contributions in this direction are: (a) a combined trace signal selection approach is proposed which decides signal selection-based on restorability, error detection and routing considerations, (b) a methodology is proposed for routing of internal signal wires to the trace buffers, which avoids collision, reduces routing congestion and wire length[11] required for routing, and (c) a technique is proposed for error localization by analyzing the traced and restored signal states.

5.4.1 Error Detection-aware Trace Signal Selection

To capture the error at some/all of the traced FFs, the error must propagate to these flip-flops. This depends on many factors, one of them is the propagation by the gates present along the different paths between flip-flops. For determining the error propagation by the respective gates, the input values at these gates must be known. One method to obtain these values is through execution of the buggy circuit/design. However, the signal selection has to be done at the design stage itself and hence the actual values arriving at the gates can not be known a priori. It is definitely not possible for the scenario when functional input patterns are involved as exhaustive netlist-level simulations for large circuits is prohibitive. For some of the bug scenarios (single cycle errors), the bug manifestation can be propagated in a trace signal by forward propagation from its inwards logic cone. Therefore, only those trace signals which are in the fan-out cone of the erroneous signal can get affected by the error. Note that this is not fully applicable to sequential paths where errors are propagated in more than one cycle [150].

Some of the previous error detection approaches [50, 111] have modeled the circuit description as a S-graph, where an edge indicates the path between flip-flops and all nodes represent the flip-flops. Along each path, analysis of conditional error propagation needs to be done from probabilities of signal transitions from input to output of gates present

[10]Note that the methodology explained before in this chapter utilizes only traced signal states for the purpose of error localization.

[11]In practice, merely obtaining a reduced wire length may not be that profitable. However, we aim the routing within a budget of wire length so that no concerns regarding physical design arise while accommodating the on-chip debug requirements.

on paths between various flip-flops. Depending on the gates and the connections between flip-flops, we need to compute signal propagation probabilities at the output of each gate. Leveraging this approach, Basu et al. [111] proposed dynamic trace signal selection based on knowledge of some error-prone zones of the design. In their approach, ranking of flip-flops is done through analysis of conditional error propagation from probabilities of signal transitions due to the gates present in the paths between various flip-flops. However, specialized error zone information is not available for every design. A similar approach has been proposed by Liu et al.[50] considering the conditional probability of error propagation to different candidate trace signals and a multiplexer-based selection of a group of trace signals. Generally, techniques based on probabilistic calculations may not be very effective for post-silicon error detection scenarios. For ranking of candidate trace signals, profiling of the conditional probabilities of the signal transitions is needed. The signal profiling (i.e., actual determination of signal values and then computation of conditional probabilities) is based on simulations with deterministic/random inputs. Hence, the topological ranking of the candidate trace signals, and in turn the signal selection depends on the inputs applied for signal profiling.

There are few approaches for detection of particular kinds of errors in post-silicon scenario [34, 55]. However, deciding the trace signals based on detection of only certain kinds of errors [34, 55, 148, 151] leads to their restricted acceptability as they can not be generalized for other error scenarios. As explained earlier, during post-silicon validation, some of the internal signal states can be read-out from the trace buffers in a non-destructive fashion and a sizable portion of the remaining signal states can be known with the help of restoration technique. For some scenarios, an expanded observability becomes a necessary condition while in other cases, an effective (even if diminished) visibility is sufficient [51, 152]. Hence, a mixed formulation for signal selection must consider both the parameters: error detection ability and restoration capability of the candidate trace signals.

5.4.2 Layout-aware Trace Signal Selection

Existing signal selection techniques focus on optimizing parameters of restoration and/or error detection without taking into consideration physical constraints like routing wire length. Thakyal et al.[107] first developed a method to select signals taking into account the routing wire length. When a signal is selected for tracing, that signal needs to be routed to the trace buffers. Note that signal selection is generally performed at the last stage of the design, when the design is synthesized and the connections between different components of the circuit are generally fixed. Therefore, the routing of trace signals needs to take care of the existing routing congestion. Although the authors in [107] discussed about routing congestion, they did not take into consideration the congestion parameters while selecting signals. Their signal selection algorithms are based on reducing the half-perimeter wire length, measured using Manhattan distance, which is agnostic of the existing routing congestion. Moreover, the congestion caused by the initial trace signal routing was not considered in [107, 153] for the purpose of trace signal selection. In various experiments, we observed that there is a wide variation in Euclidean distance between trace buffers and the candidate trace signals [116].[12] A trace signal selection that is only restoration-aware is not really helpful from the routing perspective [107]. Thus, a combined trace signal selection can assist in selecting routing-aware signals having significantly high restoration and error localization. Additionally, such a technique ensures collision/congestion avoidance while routing of the trace signals. Some other techniques [154, 155] in the literature have utilized layout-awareness for different purposes such as scan stitching or timing characterization.

Consider the illustration of Fig. 5.5 which has been drawn in a manner similar to the example used in [107]. We have divided the entire layout area (Gr) into smaller grids, with the restriction that at most two wires can pass through each grid. The congestion areas are shaded. Let us assume that A and B are the first two signals to be traced. These two signals are traced without any difficulty in routing. Suppose, it

[12]The actual routing of signals happen through wire length similar to Manhattan distance. Hence, Euclidean distance is not a proper measure for the routing impact estimation as done in [116].

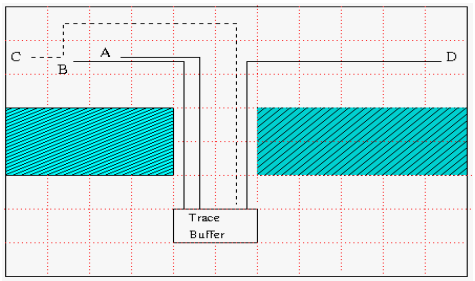

Figure 5.5: Trace signal routing conditions illustration

is given that the choice for the next trace signal is either C or D (they have the same value for other parameter of signal selection: restorability or error detection). As can be seen from Fig. 5.5, the Manhattan distance of C is lower than D. However, due to the congestion introduced by wires A and B, the wire length required to route signal C is more than that of D. Therefore, merely assuming the magnitude of Manhattan distance for routing impact estimation of trace signals is not very useful. The proposed approach of routing-aware decision making regarding choice of trace signals is shown in Figure 5.6. We consider a larger number of trace signals in the proposed combined signal selection methodology. Suppose, the trace buffer width is 32, we apply the approach in Figure 5.6 to a pool having more than 32 trace signals because some of the candidate signals could not be routed owing to the routing considerations.

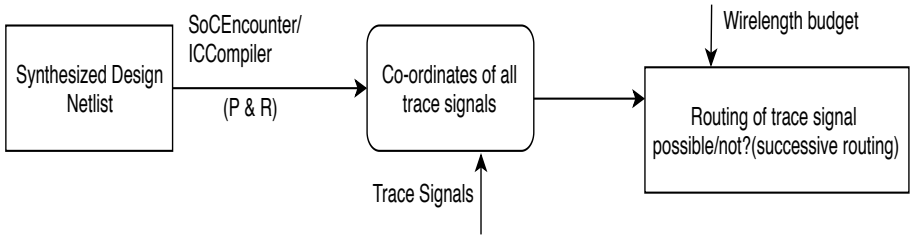

Figure 5.6: Overall flow of proposed approach

5.4.3 Heuristics for Accounting Error Propagation

In the proposed trace signal selection methodology, ranking of candidate trace signals is done through a simpler topological analysis of the circuit. The utility of the selected trace signals can be linked with the error propagation by these signals (flip-flops), which

in turn depends on the error transmission by combinational paths between them. An exact analysis of all these combinational paths for the purpose of error propagation is not very effective as explained in Section 5.4.1. Furthermore, to save on to the computational cost involved in this analysis, it is beneficial to employ approximations to account for error transmission by these combinational paths. Therefore, we utilize some heuristics to account for error propagation to the candidate trace signals, two of which are explained below. Note that these heuristics are capable of estimation of only static error transmission and are fairly inaccurate. Nevertheless, the combined trace signal selection (which is the main contribution of this chapter) can be attempted with the technique of conditional probability-based error propagation method as well for higher accuracy. Note that signal selection aimed at consideration of circuit graph centrality (the kind of which have been explored in Chapter 4) and [142] may also serve as ideal candidates for approximate accounting of error propagation to trace signals.

5.4.3.1 Heuristic-1: Error Transmission Through Incoming Paths

One method[13] to consider error transmission is to construct a S-graph from the netlist and count the number of incoming paths to each candidate trace signal. Consider the S-graph shown in Fig. 5.7 where the nodes represent different candidate trace signals (flip-flops) and the edges between them represent various combinational paths between them. Since there can be more than one combinational path between flip-flops, the weight of edges between different nodes is more than 1. The nodes which have the highest sum of all the corresponding incoming paths are to be selected as trace signals. The score of each flip-flop is given by following equation (where *total* denotes the maximum number of in-coming edges to that particular node i.e., flip-flop which is a candidate trace signal):

[13]This is similar to the topology-based score calculation of trace signals as illustrated in Section 5.2.1. However, we consider the incoming edges only for devising the heuristics.

$$scoreff = \sum_{i=0}^{i=total-1} (weight\ on\ in\text{-}coming\ edge_i) \tag{5.4}$$

Let's consider the shown below S-graph, which is a modified version of ISCAS'89 benchmark, $s208$ circuit (and has been earlier discussed in Chapter 4 and in Section 5.2.3 of this chapter). For this case, FF7 and FF8 are to be selected as these two flip-

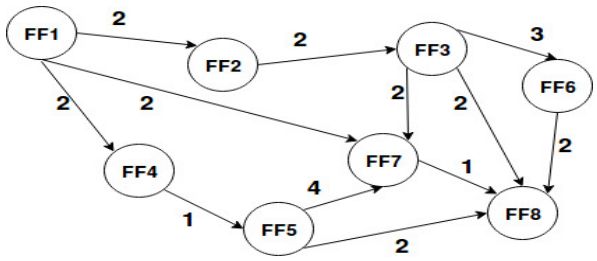

Figure 5.7: S-graph for illustrating Heuristic-1

flops have the highest $scoreff$.[14] This heuristic, however ignores the gates present on the combinational paths between flip-flops. Therefore, we consider another heuristic to account for the error transmission by gates on paths between flip-flops.

5.4.3.2 Heuristic-2: Error Transmission Through Combinational Gates on Incoming Paths

An estimation of the error transmission by each gate can be related to the number of inputs to the gate. For instance, a NOT gate always transmits the error coming at its input. However, a 2-input AND gate has its error transmission capability dependent on its inputs. This is true for different types of gates like OR, NAND, NOR etc., and with higher number of inputs.[15] The contribution of error propagation along each path is the product of the sum of error transmission score of all the gates and the number of gates

[14] We have considered the trace buffer width, TBw (i.e., the number of signals available for tracing) as 2 for this example.

[15] For accounting error propagation, the formulation in [116] generalizes contribution of each gate by $1/N_{inputs}$ where N_{inputs} is the number of inputs of the gate.

along each path. For calculating error propagation score of each flip-flop, the score of each path is added in a manner similar to Heuristic-1. Testability measures such as *SCOAP rules* [156], which assign score to each kind of gate for the ease of ATPG also perform score calculations. This scoring is more accurate than proposed heuristics as it accounts for controllability on the different signal lines according to the gate type. For paths involving re-convergence, we treat each path separately without considering the inter-dependence. If the inputs (other than that propagating the error) of a particular gate are held at the corresponding non-controlling values, the error transmission of the gate can be approximately computed. Consider the case of a 3-input AND gate (with its inputs marked as $i1$, $i2$, $i3$). For the propagation of error arriving at $i1$ to output, the other two inputs need to be held at logic value of 1. So, out of the 8 possible combinations (for $i1$, $i2$ and $i3$), there are two favorable combinations for error propagation: 011 and 111. Similarly, for any n-input gate, its n-1 inputs are to be held at the corresponding non-controlling values. The following terms are defined for the analysis of error propagation as per proposed Heuristic-2:

- *gscore*: contribution of each gate on the combinational path towards error transmission between flip-flops (where n is number of inputs to gate) is given by

$$gscore = P(error\ is\ propagated\ by\ the\ gate) = \frac{2}{2^n} \tag{5.5}$$

- *pscore*: contribution of each path towards error transmission from one flip-flop to another is given by

$$pscore = \prod gscore = \prod \frac{1}{2^{n-1}} \tag{5.6}$$

- $scoreff$: error propagation to each flip-flop is given by (*total* is the maximum

number of in-coming paths)

$$scoreff = \sum_{i=0}^{i=total-1} pscore = \sum_{i=0}^{i=total-1} (\prod \frac{1}{2^{n-1}}) \tag{5.7}$$

Consider the two paths (marked as path1 and path2) between FF A and FF B in Fig. 5.8. For this case, individual scores, *pscore* (computed by accounting for contribution

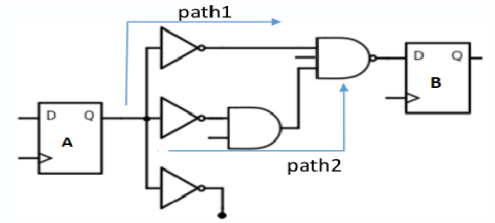

Figure 5.8: Combinational paths for illustrating scoring as per Heuristic-2

of the gates on these paths: 1-input gate = 1, 2-input gate = 1/2, 3-input gate = 1/4 and so on) is given as below:

- *pscore* (path1)= 1*1/4 = 1/4
- *pscore* (path2)= 1*1/2*1/4 = 1/8

Continuing in this manner, *scoreff* of all the flip-flops (candidate trace signals) are computed and ranking is done. For gates such as XOR and XNOR which have excellent error transmission capability (comparable to that of NOT gate), their *gscore* is assigned the maximum value of unity. This modification may not be accurate in all cases. It is represented in Algorithm 9 by checking if type of gate equals to "*XG*". Converting these gates into basic ones (such as like OR, AND, NAND etc.) worsens the complexity as number of paths (after conversion) increases. This limits the accuracy of the heuristic severely as the paths (inside XOR gate) are re-convergent. Inability to accurately handle re-convergent paths is one of the limitations of this heuristic. Since inputs arriving at re-convergent nodes are inter dependent, the proposed heuristic fails to capture the effect of re-convergent nodes accurately. For re-convergent structures, the methodology proposed

in [157] can be utilized to transform them into simpler structures (this transformation is only to estimate error propagation for ranking/scoring purpose). This technique involves converting a larger gate into a supergate structure and then performing its partitioning and subsequent logical cone clustering.

We present the algorithm to obtain trace signals by Heuristic-2 here as Algorithm 9. Signals corresponding to Heuristic-1 can be derived easily from this algorithm by appropriately omitting the additional steps (needed for Heuristic-2). The combined signal selection methodology (Algorithm 10) creates a group of signals capable of maximum error detection (S_{ed}) and combines these signals with the group of signals having highest restoration ratio (S_{res}) according to a partitioning factor. Note that the signals of S_{res} can be obtained by one of the restorability maximization techniques [1, 9, 102, 141] or any other such trace signal selection technique. The total flip-flops (candidate trace signals) are denoted by $FFlist$ in this algorithm. As explained earlier, Algorithm 1 checks for the type of gate on paths between different flip-flops, if the gates are of XOR/XNOR (denoted by XG), these are assigned a $gscore$ of unity, otherwise, scores are assigned as per Heuristic-2. In the end, this algorithm selects the higher ranked TBw signals from a ranked list of trace signals.

5.4.3.3 Algorithmic Complexity of Proposed Heuristics

Similar to Algorithm 6, the worst case time complexity of the proposed heuristics is $O(n_{path})$ where n_{path} is the number of paths in the netlist. This is same for both the proposed heuristics ($H1$ and $H2$). Thus, the computation of scores as per these heuristics is scalable. Note that the conditional probability-based error propagation calculation is more accurate than the proposed heuristics. It is obvious that the proposed heuristics require much lesser computations than that of conditional signal transition probability-based accounting for error propagation. Further, we observed that the runtime of the proposed heuristics is either comparable to or lesser than some of the restorability maximization-based signal selection techniques ([1], [9]) for the largest benchmark circuits in Table 4.4.

Algorithm 9: *build*S_{ed}

Input: $Design$,TBw,$FFlist$
Output: S_{ed}

$FFlist \leftarrow$ all flip-flops (candidate signals) in $Design$;
for *each flip-flop* FF_i *in* $FFlist$ **do**
  $P_i \leftarrow$ all incoming paths to FF_i;
  for *each path in* P_i **do**
    if *gate type* $= XG$ **then**
      $scoreg \leftarrow 1$;
    end
    else
      $n \leftarrow$ number of inputs for each gate;
      $scoreg \leftarrow \frac{1}{2^{n-1}}$;
    end
    $scorep \leftarrow \prod scoreg$;
  end
  $score_{ed} \leftarrow \Sigma\ scorep$;
end
$S_{ed}^{temp} \leftarrow$ sort each FF by its respective $score_{ed}$;
$S_{ed} \leftarrow TBw$ signals from S_{ed}^{temp};

5.5 Proposed 2-Parameter Combined Selection Methodology

For the purpose of combined signal selection, the computation of a separate group of S_{res} and S_{ed} defined in Section 5.4 is straight forward. However, computation of S_{ra} is not trivial as the routing overhead (in terms of congestion and routing wire length) depends strongly on other signals as the routing process proceeds. We overcome this difficulty in an indirect manner through a two-step procedure. We propose a trace signal routing methodology (Algorithm 11) that avoids routing congestion for a given set of signals. Minimization of the routing wire length is aimed as the subsequent step in the proposed signal selection methodology (Algorithm 12). We begin this section by describing first the 2-parameter trace signal selection which is to be utilized in Algorithm 12.

5.5.1 Combining Error Detection and Restoration for Signal Selection

To combine both the error detection and signal state restoration parameters for selection, a methodology is presented as Algorithm 10. This methodology utilizes the output of Algorithm 9 (which is a list of signals as per Heuristic-2) and then provides a combined list of trace signals. However, as mentioned before with minor modifications, Algorithm 9 can provide signals as per Heuristic-1 also which can be utilized in Algorithm 10. Since evaluation of signal state restorability (in terms of RR) of all the possible combinations is cumbersome, we obtain few combinations and evaluate RR for each of them. This

Algorithm 10: *CombSignalSelect*

Input: S_{ed},S_{res},np
Output: S_{comb}

initialize k such that $0 < k < 1$;
current partition$(cp) = 1$;
while $cp < np$ **do**
    $S_1 \leftarrow k$*S_{res};
    $S_2 \leftarrow$ (1-k)*S_{ed};
    common $\leftarrow S_1 \cap S_2$;
    if *common* != $\emptyset$ **then**
        remove *common* from S_2;
        add next element(s) from S_{ed} to S_2;
    end
    $S' \leftarrow S_1 \cup S_2$;
    Evaluate RR for S';
    $cp = cp + 1$;
    increment k by ϵ such that $0 < k + \epsilon < 1$;
end
$S_{comb} \leftarrow S'$ for which RR is highest;

2-parameter signal selection methodology takes two inputs: np and k, which depend on the level of granularity desired for selecting the final set of TBw trace signals. These parameters can vary from one circuit to another and need to be fixed at the design stage. The partitioning factor, k decides the number of signals from S_{res} and S_{ed} to be included in the final list of trace signals. For instance, if $np = 5$ and a successive increase of $\epsilon =$

10% is chosen during each partition, then k (with an initial value of 0) has the value of 0.1, 0.2, 0.3, 0.4 and 0.5 corresponding to each partition. Initially, both the signal sets, S_{ed} and S_{res} have TBw elements (signals) each. Then, the signal selection algorithm creates different partitions (by using partitioning factor, k) of the two groups ($score_{ed}$ and $score_{res}$) such that always TBw signals are selected at each stage of partition and Restoration Ratio (RR) is evaluated. Note that the parameter np decides how many such partitions need to be created for the final trace signal selection.

If TBw trace signals are to be selected by the proposed 2-parameter selection methodology, N_1 signals are chosen from S_{res} and N_2 signals are chosen from S_{ed}, leading to $TBw = N_1 + N_2$. If $N_2 = 0$, $TBw = N_1$ and for $N_1 = 0$, $TBw = N_2$. Experimental results for 2-parameter trace signal selection in the earlier version of this work [116] have shown that as N_1 increases, there is very less difference between restoration with restorability maximized signal set and the signals obtained with this combination. This implies that signal state restoration obtained from the combined signals achieves closer values to that of restorability maximization signals. Beyond a certain partition of the sets (S_{res} and S_{ed}) either decrease in restoration becomes minuscule or it even increases. This is primarily because of diminishing restoration ratio (RR) effect which means that as more trace signals are selected, there is very small addition to the total signal restoration [96]. Additionally, note that signal restoration depends on various factors like inputs applied, the ordering of signals in the trace signal list and the manner in which rules of logic implication are applied. Finally, it is worth to mention that for proposed 3-parameter signal selection (presented later in this chapter as Algorithm 12), a and b represent k and 1-k respectively and $c = 0$.

5.5.2 Illustration of Combined Trace Signal Selection

To illustrate that signal state restoration and error detection can be simultaneously analyzed for the purpose of signal selection, we consider the portion of a circuit shown in Fig. 5.9. We exhaustively explore the possible options of trace signal selection given that the trace buffer width is 2 for this circuit. Note that for brevity in illustration, only

some of the inputs of the gates are shown. For brevity, we assume here that these inputs are connected to PIs (primary inputs) of the circuit.

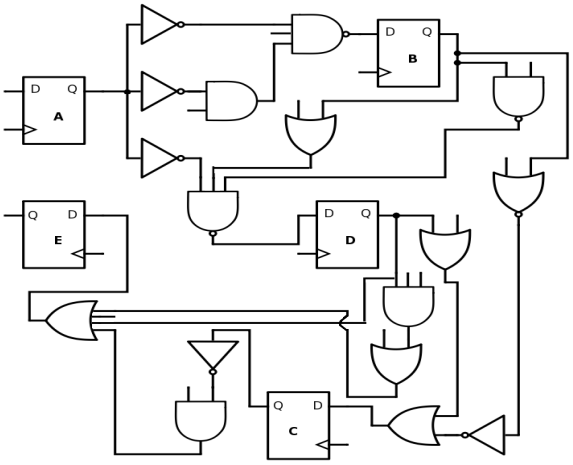

Figure 5.9: Example Circuit for illustrating combined trace signal selection

With the help of scoring mechanism explained as per Heuristic-2 in the previous subsection, the error-detection score ($scoreff$) can be computed as shown below:

1. $scoreff$(A) = 0
2. $scoreff$(B) = 1*1/2*1/4 + 1*1/4 = 0.37
3. $scoreff$(C) = 1*1/2*1/2 + 1/2*1/2 = 0.50
4. $scoreff$(D) = 1*1/4 + 1/2*1/4 + 1/2*1/4 = 0.50
5. $scoreff$(E) = 1*1/2*1/8 + 1/2*1/4*1/8 + 1/8 = 0.20

For the ranking of trace signals as per above $scoreff$, decreasing order of the candidate trace signal in the set S_{ed} is given by {C,D,B,E,A}. Next, we check RR for each possible combination of trace signals for TBw (trace buffer width) as 2. As we have five signals (flip-flops), there are a total of ten combinations which are candidates for selection. Table 5.3 shows the different trace signal combinations (TSC), the corresponding RR obtained, the $scoreff$ of the two signals (i.e., $scoreff(1)$, $scoreff(2)$) and the sum of $scoreff$ of this combination of trace signals. The RR values are obtained with simulation with random inputs for 100 cycles. The parameter ($\Sigma scoreff$) is an indication of

the error detection/localization with the help of the respective trace signal selection. As

Table 5.3: Illustration of possible TSC (with Heuristic-2)

Sl.	$TSC(1,2)$	*RR*	$scoreff(1)$, $scoreff(2)$	$\Sigma scoreff$
1	C,B	1.08	0.50, 0.37	0.87
2	C,A	1.68	0.50, 0	0.50
3	E,A	1.66	0.20, 0	0.20
4	E,C	1.25	0.20, 0.50	0.70
5	B,D	1.63	0.37, 0.50	0.87
6	E,B	1.58	0.20, 0.37	0.57
7	C,D	1.80	0.50, 0.50	1.00
8	A,D	1.90	0, 0.50	0.50
9	B,A	1.81	0.37, 0	0.37
10	E,D	1.46	0.20, 0.50	0.70

is evident from Table 5.3, TSC with maximum RR does not have the highest $\Sigma scoreff$, suggesting the need of a combined trace signal selection methodology. Therefore, for the different signal combinations presented in this table, the choice of {C,D}, which has the highest $\Sigma scoreff$ is the most profitable as its restoration ability (indicated by RR) is also higher than most of the other trace signal combinations.

5.6 Proposed Congestion-aware Routing Algorithm and Wire Length Measurement Technique

5.6.1 Basic Ideas

The proposed layout wire length measurement technique takes into consideration the routing congestion introduced by the successive signals getting routed. The proposed trace signal routing wire length measurement technique is based on Lee's algorithm [158]. Before routing the signals, the current congestion criteria of the design is updated (i.e., layout regions unavailable for routing are recorded and tracked). We route the trace signals serially in order of their importance based on restoration and error detection.

The location of the trace signal is considered the *start point* and the trace buffer location is considered the *end point*. As can be seen from Fig. 5.10, the smaller grids

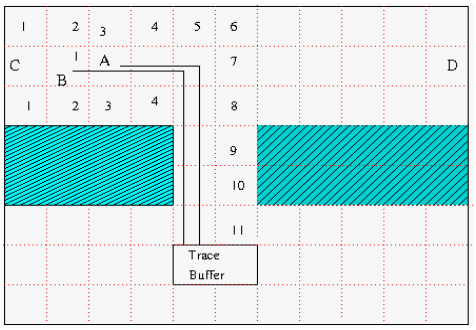

Figure 5.10: Routing conditions for signal C

adjacent to the signals are numbered 1 and the values are incremented with each adjacent grid till the *end point* is reached. For simplicity, the total wire length is measured as the maximum number of grids covered from *start point* to *end point*. The routing conditions of trace signal C (of scenario depicted in Fig. 5.5) are shown in Fig. 5.10. It can be seen that the wire length required to route C is 11 units. On the other hand, if we want to route D, the wire length required is 9 units as seen in Fig. 5.11. Further, note that after a trace signal is routed, the congestion criteria is updated with the newly laid wires (i.e., connections to the trace buffers) due to the routing completed so far.

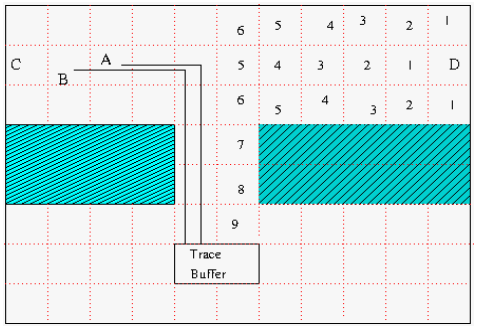

Figure 5.11: Routing conditions for signal D

To avoid routing congestion, we consider a factor $Wlimit$ (typically as 3 or 4) as the maximum lines allowable through the smaller rectangular grids in Gr (which represents the complete layout area). Collision detection is done by checking if a (smaller rectangular) grid, g[16] on its path has $Gr[g]$ equal to $Wlimit$. In cases where $Wlimit$ has been already reached, the algorithm attempts rerouting by moving it up/down and sideways.

[16]In our implementation, size of such grid is 1 micron by 1 micron.

For example, if the grid [3,4] is filled, attempt is made to route through [3,3], then move right and so on.

5.6.2 Description of the Routing Algorithm

Consider the distance of any trace signal from the trace buffer (TB) as dn and $Max(dn)$ is the Maximum Manhattan Distance of all the candidate trace signals. Consequently, for each candidate trace signal, a function F_{route} can be defined as below since it is not beneficial to route chains (which are the connections from the signal to the trace buffer) with length more than the maximum Manhattan distance, $Max(dn)$. In the proposed methodology, this function ensures that the trace signals are getting routed by Manhattan distance when there is no congestion.[17] In the proposed algorithm (shown here as Algorithm 11), congestion avoidance takes place only when there is routing congestion and thus Manhattan distance-based routing wire length can not be allowed in this case.

$$F_{route} = \begin{cases} 1 & \text{if dn<Max(dn)} \\ 0 & \text{if dn>Max(dn)} \end{cases}$$

The proposed approach of trace signal routing from a wider pool of trace signal candidates is shown in Figure 5.12. It is a graphical representation of the steps outlined in Algorithm 11.

5.7 Proposed 3-Parameter Trace Signal Selection

Consider the three parameters- a, b and c which represent the weightage for restoration parameter (RR), error detection parameter (ED) and layout-easiness parameter (LW) respectively as explained before. For devising a trace signal selection methodology based

[17] In practice, dn equals Manhattan distance when there is no congestion.

Algorithm 11: *TrSignalRouting*

Input: $placedCoord$,$MaxX$,$MaxY$,$Wlimit$,S_{tr}
Output: $routedConn$,$Wlength$

```
1  Gr ← grid of MaxX by MaxY dimension (consists of a set of smaller grids);
2  Initialize all the entries of Gr[g] to 0;
3  S_tr ← list of trace signals;
4  placedCoord ← coordinates of FF's;
5  tbx, tby ← coordinates of trace buffer;
6  Wlimit ← maximum number of wires routed through each smaller grid;
7  maxlength ← maximum distance available for routing across the complete grid
     Gr;
8  for each trace signal ff_i in S_tr do
9      ffx, ffy ← coordinates of ff_i from placed Coord;
10     Value=F_route(ff_i);
11     If Value=1, proceed below else skip to next signal;
12     For each grid g belonging to Gr that has a wire routed through it, Gr[g] =
         Gr[g]+1;
13     Across Gr, move above, below, sideways (from east to west and then from
         north to south) and check for Wlimit in each direction;
14     Obtain wire length(wl) from ffx, ffy and tbx, tby;
15     if Gr[g] != Wlimit && wl < maxlength then
16         Signal can be routed to trace buffer;
17         Update Gr[g] for smaller grids in this path to TB;
18     end
19     else
20         Signal can not be routed to trace buffer;
21     end
22     Wlength ← wl of routed signal;
23 end
24 routedConn ← routed connections of trace signals;
```

on the enhancement of all the three parameters, we define the following score function:

$$scoreFunction = a * RR + b * ED + c * LW \tag{5.8}$$

However, deciding trace signals based on this score function is not easy as these three factors have conflicting preferences. It is desirable that RR and ED are as high as possible whereas LW (wire length required for routing particular signal to trace buffer) be as low

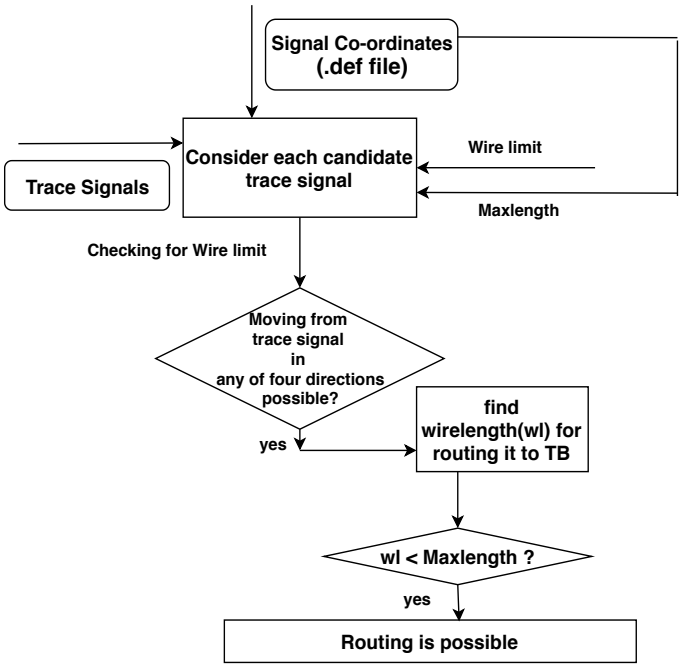

Figure 5.12: Proposed algorithm for routing of trace signals

as possible. Hence, additive combination of all these three parameters is not straightforward. Furthermore, the computation of routing congestion is possible only when successive signals are routed and checked for overhead (say, wire length constraints for a given budget). As explained before, we propose a two step signal selection methodology to overcome this problem. The proposed 3-parameter trace signal selection methodology is described below as Algorithm 12.[18]

The first step of signal selection is done by picking from three lists- S_{res}, S_{ed} and a sorted list of trace signals based on Manhattan distance (from the trace buffer) of each one of them. Thus, the third list serves as a measure of wire length for S_{ra} in Eqn. 5.3. However, actual routing length of the signals would not be same as their Manhattan distance. So, the final selection should be based on route length which can be decided only when routing of signal proceeds as the congestion resulting due to subsequent routing of each signal needs to be accounted for. We consider an initial number of signals as

[18]Algorithm 11 implements Manhattan distance-based checking + congestion-checking. Whereas, in Algorithm 12, Manhattan distance-based checking is not needed as already a sorted list of trace signals (in S_{ra}) is input to it.

Algorithm 12: *SelectFinalTrSignals*

Input: a,b,c,TBw,S_{res},$placedCoord$,tbx,tby,S_{ed}
Output: $TrSignals$

```
1  q, r ← 0;
2  for each trace signal ff_i in FFlist do
3  |   Mhd_i ← Calculate Manhattan distance (MD) from placedCoord and tbx,
   |    tby;
4  |   LW ← Mhd_i;
5  end
6  S_ra ← Sort all signals as per their LW values;
7  TrSignals' ← 2*TBw signals from S_res, S_ed and S_ra by using choices of a, b & c;
8  route 1st signal from TrSignals' by Algorithm 11;
9  compute route length (RL) for 1st signal;
10 tempi=1, index=tempi;
11 while tempi != TBw do
12 |   index=index+1;
13 |   if RL of sig[index] < MD of sig[index + 1] then
14 |   |   Keep this signal in TrSignals;
15 |   |   tempi=tempi+1;
16 |   |   if this signal is from S_res or S_ed then
17 |   |   |   increment q or r accordingly
18 |   |   end
19 |   end
20 |   else
21 |   |   route sig[index + 1] to trace buffer;
22 |   |   swap positions of these 2 signals in TrSignals' to move less profitable
   |   |    signals downwards;
23 |   end
24 end
25 if q and r are not comparable to a*TBw and b*TBw respectively then
26 |   Need to change wire length budget(maxlength);
27 |   Call Algorithm 11 with new wire length budget and start again from step 8
   |    of this algorithm;
28 end
29 TrSignals ← finally selected and routed trace signals;
```

twice the trace buffer width (TBw) in the list $TrSignals'$. In practice, a list of n*TBw signals is needed where n>1. For simplicity, we chose n as 2 here. This is because if some signals in top TBw signals can not be routed as per the routing constraints, low ranked signals can be included in the final trace list to select TBw signals. Thus, initially the

total number of signals chosen from S_{res}, S_{ed} and S_{ra} equals 2*TBw as per the respective choices of a, b and c.

One of the unfavorable scenarios occurs when all the signals from the groups S_{res} and S_{ed} can not be routed. Under such a scenario, a higher wire length budget is to be accommodated. This is a preferable approach compared to a sorting of signals of S_{res} and S_{ed} and then creating an initial group out of them. It can lead to undesirable signals from the perspective of state visibility enhancement and error localization causing severe degradation of these important factors. Hence, we introduce an approximate check condition for such cases in Algorithm 12. The algorithm begins routing for the first signal with LW parameter (in Eqn. 5.8) for it as Manhattan distance (MD). After this signal is routed, we obtain its routing length (RL) as wire length avoided due to collision (if any). This length is compared with Manhattan distance of the next trace signal. If RL of previous signal is less than MD of next signal, this signal is selected as final signal otherwise the positions of signals (flip-flops) in the list $TrSignals'$ are swapped. This swapping and incremental selection ensures that we are not missing out important signals from the point of restorability/error detection and simultaneously choose signals which are layout-friendly. This process is iterated till the trace buffer width (TBw) is reached. This is the second step of signal selection to get the final list of trace signals, $TrSignals$.

One of the minor differences between the 2-parameter signal selection described in earlier part of this section and the proposed 3-parameter signal selection algorithm described above is that fine-grained divisions of the individual signal sets are not accounted in the latter. However, a fine-grained partitioning based on input parameters (a, b and c) can be easily done on all the three signal sets involved here and then accordingly incorporated in $SelectFinalTrSignals$ methodology. Moreover, the common elements (i.e., trace signals) between various signal sets, if present in S_{final} (which is obtained from the methodology in Algorithm 12) can be removed similar to the 2-parameter trace signal selection (Algorithm 10) described earlier.

5.8 Proposed Error Localization Methodology

We analyze the restored and traced signal states for the purpose of gate-level error localization. After application of signal state restoration technique, a significant amount of the unknown signal states are discovered. The partially reconstructed (traced+restored) flip-flop signatures are compared with golden signal states which can be obtained from a high level reference design description. In the experimental section, we report error localization results for signal selected with the proposed methodology and compare with restorability maximization-based signal selection techniques. We perform a topological connection-based analysis of the injected error location ($error_i$) and identify the actual suspect flip-flops that can be infected by $error_i$. The randomly injected design error (at gate-level) falls into the logical cone of few flip-flops ($suspect_{actual}$). With the help of the traced and restored flip-flop values, we can localize to $suspect_{actual}$ either exactly or obtain a list of probable candidates. This is illustrated in detail in Section 5.9.3. If the suspect flip-flops ($suspectff$) obtained by the analysis of the debug data matches with $suspect_{actual}$, we can localize the error(s) easily to a smaller region.

Typically, silicon debug is done in multiple sessions[19] [159]. For example- consider a debug scenario with two successive runs. After the first run finishes, at T^{th} clock cycle, contents of all the flip-flops can be known through the scan chains. In the next run/session, due to an execution of 1024 clock cycles (which is the trace buffer depth), the data from $T + 1^{st}$ to $T + 1024^{th}$ clock cycles needs to be taken out of the trace buffer and analyzed for the debugging purpose. We have considered T as 1000 for all the circuits during the experiments. The full state of the design at T^{th} cycle is known through the help of scan chains. The last completely known design state (corresponding to the previous execution) is shown by lcs in Algorithm 13. As we perform off-line dumping of the complete state corresponding to T^{th} cycle for debugging analysis, we can feed this state (lcs) back into the chip through the same scan chains, to make sure that during the session from $T + 1^{st}$ to $T + 1024^{th}$ clock cycles, the chip state remains same

[19]The methodology in Chapter 7 is also based on this technique.

as that at the end of first session. However, *lcs* may not be fully correct as it has been obtained from a buggy design execution.

Algorithm 13: *SigStatesBasedDebug*

Input: $GFF_{signature}$,$TracedSignalStates$,lcs,$FFlist$
Output: $suspectff$

```
1  lcs ← last known complete design state;
2  Data_in ← TracedSignalStates;
3  Apply signal state restoration technique on Data_inc using lcs, Data_in and design
     description;
4  Data_exp ← expanded debug data;
5  GFF_signature ← flip-flop signature of D_tb cycles;
6  for each flip-flop (ff_i) of FFlist do
7  |   Compare GFF_signature and Data_exp;
8  |   for each clock cycle do
9  |   |   f ← signal state of ff_i in Data_exp;
10 |   |   f' ← signal state of ff_i in GFF_signature;
11 |   |   suspectval_ff_i=0;
12 |   |   if f == X or f == f' then
13 |   |   |   suspectval_ff_i=0;
14 |   |   end
15 |   |   else
16 |   |   |   suspectval_ff_i=suspectval_ff_i+1;
17 |   |   end
18 |   end
19 end
20 Rank all FFs by suspectval_ff_i;
21 suspectff ← Top ranked FFs;
```

With different trace signal selection techniques, expanded visibility ($Data_{exp}$) varies significantly. This variation has impacts on the efficacy of error localization process. Note that an increase in $Data_{exp}$ does not lead to a better localization if the traced signals are not effective from the point of error localization. As stated before, we assume that $GFF_{signature}$ is available from the simulation of a high level abstraction of the design which is fully verified. This is very difficult to obtain in practice. For the purpose of experiments, simulation of a non-buggy version of the design is treated as the golden circuit response. This is one of the limitations of the proposed debugging methodology since most of the subtle bugs can take days or weeks for excitation [159].

Note that the signal state restoration is applied on the buggy netlist and the traced signal states of the buggy design. This ensures that the restored signal states are indeed correct and assist in ruling out the possibility of false positives in the list of suspect candidates during the debug process. To obtain *lcs*, we utilize the scan chains as explained before. With the assumption of repeatable behavior of bugs, the scan dumping can be achieved easily and can be performed immediately after the trace data is offloaded. However, the proposed methodology requires stalling of the chip execution to facilitate the dumping of the scan and trace data. This is definitely an undesirable aspect of the proposed debug methodology as it leads to intrusiveness inside the design execution. While a complete elimination of these execution stalls is very difficult, efforts can be envisaged to minimize the number of such stalls by methods of compression of traced data or summarizer-based gathering of trace data [160]. Finally, it is worth to note that the proposed methodology presented in Algorithm 13 assumes the complete knowledge of *lcs* which can not be extracted in some of the debug scenarios even if the corresponding bug behavior is repeatable.

5.9 Experimental Results and Discussions

5.9.1 Experimental Setup

We selected a variety of benchmark circuits for experiments- ISCAS'89, ITC'99 and Opencore [143] suites. The characteristics of these benchmark circuits are noted in Table 4.4. For the purpose of routing, the benchmark circuits were first synthesized using Synopsys Design Compiler and Synopsys 32 nm educational generic standard cell library. Floorplanning and placement were performed by Synopsys IC Compiler tool with the cell libraries (in the SAED 32nm distribution) as per multiple metal layers.

In all the experiments, we have considered the trace buffer width, TBw as 32 and the trace buffer depth, D_{tb} as 1024. We define the following metric[20] to quantify the amount

[20]Because of change in inputs, $States_{resto}$ varies with each experiment targeting error localization. The results in Table 5.4 indicate the value of the metric which is observed maximum times out of experiments involving a total of 100 iterations.

of signal states reconstructed with a given set of trace signals:

$$Restored\ fraction, Rf(list) = \frac{States_{resto}(list)}{D_{tb} * F_{tot}} \tag{5.9}$$

5.9.2 Comparative Evaluation of Signal State Restoration

The signal restoration calculation of different signal selections are reported in Table 5.4. Note that F_{tot} represents the total signals (flip-flops) in the design). We consider here trace signal selected by Heuristic-1 and Heuristic-2 as $S(H1)$ and $S(H2)$ respectively. Additionally, following methods of restorability maximization from literature are considered for comparison:[21]

- greedy heuristic of region-based selection, utilizing structural relationships of the design netlist [1]-$M1$
- simulation-based selection using ILP (Integer Linear Programming) to discover most effective signals [9]-$M2$
- multi-mode hybrid signal selection, based on analysis of circuit into various modes due to control signals [141]-$M3$

The signal restoration results in terms of Rf[22] values are presented in Table 5.4 for all the five trace signal lists (which are obtained from $H1$, $H2$, [1], [9] and [141]). For the latter three cases (i.e., [1], [9] and [141]), they are shown by $M1$, $M2$ and $M3$ respectively for identical representation. Throughout the rest of this chapter, these methods are often referred to as restorability maximization-based signal selection techniques.

Note that in Equation 5.9, $States_{resto}(list)$ is inclusive of the traced signal states. For some of the benchmark circuits, there is a wide disparity between signal state restoration computation of signals obtained with the proposed Heuristic-1 or Heuristic-2 and the restorability maximization-based signal selection methods [1],[9],[141]. However, signals

[21] We specifically consider these three signal selection techniques for comparison as the respective authors have publicly released their tools.

[22] Compared to RR, Rf serves as a direct metric for accounting error localization as fraction of total signal states known to us (either through signal restoration or tracing) is computed in the latter metric.

Table 5.4: Restored fraction, $Rf(\%)$ for different signal selections

$Name$	$S(H1)$	$S(H2)$	$S(M1)$	$S(M2)$	$S(M3)$
$b17$	4.02	5.56	14.40	15.63	8.89
$b21$	13.26	10.14	11.34	24.87	25.13
$p16c5x$	12.70	13.72	19.68	17.59	17.80
$mips$	81.93	81.93	96.71	96.59	42.87
usb	7.08	5.43	45.08	32.38	35.79
s5378	48.20	41.40	59.28	51.16	62.39
s9234	23.72	42.42	28.62	66.68	82.76
s13207	50.34	52.26	42.58	29.50	79.80
s15850	13.50	12.05	25.27	65.03	85.09
$s35932$	7.39	7.22	53.35	84.67	81.19
$s38417$	2.55	2.55	5.95	16.08	20.87
$s38584$	72.15	70.69	70.07	81.76	82.97

with proposed heuristics ($H1$ and $H2$) could not achieve comparable restoration to these techniques for some of the benchmark circuits. Thus, we expect to benefit from the point of signal reconstruction when we perform a combined signal selection (i.e., mixing of signals selected for various preferences as the internal visibility can be enhanced by combination). Note that the sum of traced and restored signal states reported in Table 5.4 are obtained by applying user-defined inputs unlike random inputs as is done in [116]. As is evident for some of the benchmark circuits, two different set of trace signals may have the same magnitude of *Restored fraction* (Rf). For *mips* circuit, trace signals obtained with proposed heuristics ($H1$ and $H2$) achieved higher signal state restoration than trace signals obtained from the technique described in [141].

5.9.3 Design Error Localization

5.9.3.1 Chosen Design Error Models

For electrical bugs, their behavior can be modeled as bit-flips [24, 34, 161]. However, it is widely acknowledged that modeling design bugs at the post-silicon level is difficult and can not be generalized [55, 116]. In spite of this limitation, certain gate-level errors can reasonably represent design errors with in complex digital blocks of the design [43], [50], [162],[163],[164]. Four such error models [51, 55, 116] which are utilized throughout

this thesis are shown in Table 5.5. Their corresponding RTL equivalents can be easily derived [93]. However, the inherent assumption is that their detection at the pre-silicon verification may not be very easy because either the particular test-case is missing in the test bench or the specific sequence to excite (sensitize) this kind of error is lengthy or very complicated [24, 51, 55, 165].

Table 5.5: Gate-level design bug/error models

	Name	Meaning
$e1$	wire exchange	one wire gets exchanged by another
$e2$	extra inverter	an unintended inverter gets added
$e3$	input shorts	inputs of a gate get shorted
$e4$	random gate replacement	a gate gets replaced by another gate

Out of the four bug models mentioned in Table 5.5, we illustrate the first two cases. The *wire exchange* scenario is shown in Fig. 5.13. Suppose wires marked as **s** and **t** are exchanged. Consequently, the error propagates to F2, F3 and F4 (all shown in red). With the help of discovered (restored) and known (traced) signal states, if we are able to localize to all or one of them, in subsequent debug steps we can reach to the exchanged wires with least effort. The second bug scenario, $e2$ (extra inverter insertion) is depicted

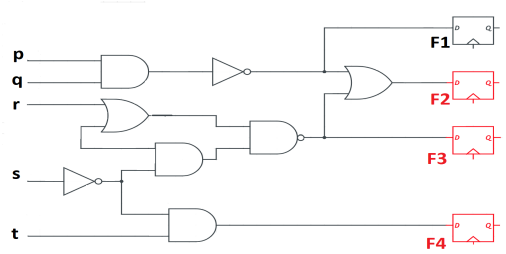

Figure 5.13: Design error-1 ($e1$) illustration

in Fig. 5.14. The inverter (circled) is added to the netlist because of a design bug and its effects reaches to F1 and F2 through combinational gates (which are shown in red in addition to the affected flip-flops). For an effective localization, the debug data should assist in arriving at F1 or F2 or both (but not F3).

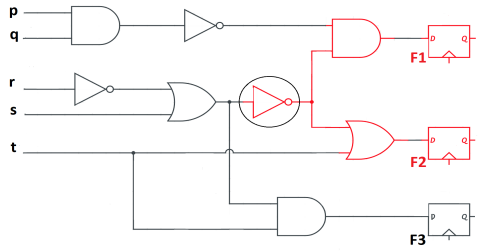

Figure 5.14: Design error-2 ($e2$) illustration

5.9.3.2 Error Localization Metric Definition

Typically, error detection-aware selection algorithms employ a metric which measures the number of erroneous signature bits when compared to golden signature bits [116, 148]. However, such a metric can not be directly related with error localization. Thus, we define a localization function (Z_{loc}) described in Table 5.6. We performed 25 iterations of each type of error injection scenario (mentioned in Table 5.5) for each benchmark circuit. In each iteration of the error injection, a set of the infected flip-flops ($suspect_{actual}$) are obtained and then this set constituents are matched with elements of $suspectff$ set. This matching leads to an exact root-cause discovery and the corresponding bugs/errors in the RTL description can be identified and some fixes or correction measures can be attempted in a relatively quicker manner.

Table 5.6: Localization metric definition

$fn.$	Value	Condition	Remark
Z_{loc}	1	$suspectff = suspect_{actual}$	effectively localized
Z_{loc}	1	$suspectff \subset suspect_{actual}$	localized
Z_{loc}	0	$suspectff \not\subset suspect_{actual}$	not localized

5.9.3.3 Comparative Evaluation of Design Error Localization

Z_{loc} values for all the four scenario ($e1$, $e2$, $e3$ and $e4$) are reported in Table 5.7 in an averaged manner. For each type of error, the error injection experiment is carried out K times; Z_{loc} has a maximum value of K and a minimum value of zero. We performed K ($= 25$) iterations of each type of error (wire exchange, extra inverter, input shorts, gate

replacement) injection for each benchmark circuit, totaling into 100.[23]

As it is seen in Table 5.7, the signals with proposed $H2$ achieve better error localization than the restorability maximization methods [1, 9, 141] except for $s35932$ and $s38417$ circuits. Additionally, trace signals corresponding to $H2$ perform better than $H1$ because of improved error propagation through gates on the combinational paths. However, this difference is not large because of inaccurate estimation with $H2$ which can be improved further. For $s15850$ circuit, signals from $H1$ perform better error localization than that of $H2$. We observed similar trends when we repeated the error injection and the subsequent localization experiments for fifty iterations of each error model corresponding to each circuit. From Table 5.7, it is clear that an increased internal visibility does not

Table 5.7: Localization results (Z_{loc} values out of 100)

$Name$	$S(H1)$	$S(H2)$	$S(M1)$	$S(M2)$	$S(M3)$
$b17$	54	59	50	49	45
$b21$	59	62	53	47	46
$p16c5x$	57	65	49	46	35
$mips$	67	57	53	49	43
usb	50	55	49	47	52
$s5378$	58	56	45	43	54
$s9234$	55	57	43	46	46
$s13207$	59	60	44	42	49
$s15850$	58	56	49	52	47
$s35932$	40	42	47	50	46
$s38417$	33	35	42	43	46
$s38584$	57	62	54	47	51

monotonically translate to an enhanced error localization.[24] Note that restorability maximization techniques are generally out-performed. One of the reasons is that structural relationships are helpful in identifying signals capable of higher error detection. The other metric to evaluate error localization is the number of common elements between $suspect_{actual}$ and obtained $suspectff$ for each iteration of error injection. We observed

[23]Thus, the averaged Z_{loc} values (corresponding to the four error scenarios, $e1$, $e2$, $e3$ and $e4$ shown in Table 5.5) also have a maximum value of K (= 25). In [33], these values were reported for a total of 80 iterations for each benchmark circuit.

[24]Increasing the number of error injection experiments to 200, we observe the same trend for the signal selections as shown in Table 5.7.

that the number of such common elements are substantially higher in the case of $S(H1)$ and $S(H2)$ as compared to the restorability maximization techniques [1, 9, 141]. Across the benchmark circuits and different error models, the least value of Z_{loc} obtained by $H1$, $H2$, [1], [9] and [141] vary in the range of six to seventeen out of the maximum value of twenty-five among the individual error models.[25][26] Some of the trace signals perform better for some of the error models with varying nature from one circuit to another. The lower values of Z_{loc} indicate the difficulty in error localization with a restricted visibility of the internal signal states of the design.

We performed the above experiments with the trace signals from topology-based selection [51] which is explained earlier in this chapter. The error localization (Z_{loc} values) are comparable to that of the trace signals obtained with $H1$ or $H2$. For the sake of completion, we also implemented the methodology in [111] where in the analysis of error propagation by conditional probability is done. We observed that error localization with the trace signals obtained from the technique in [111] provide comparable results to that of the trace signals obtained with $H1$ or $H2$ in most cases, while slight improvement is achieved in few cases.

Note that inspite of lower values of Rf, we succeed in error localization as this metric includes traced signal states along with the restored ones. Traced signal states play an important role in localizing the error. For the error injection experiments in which Z_{loc} value turns out to be zero (i.e., there is no exact match between the sets $suspect_{actual}$ and $suspectff$), it is required to closely inspect the signals obtained in the $suspectff$ set. The flip-flops in the vicinity of the elements in $suspectff$ set can lead to the signals in $suspect_{actual}$ set. Achieving exact error localization (i.e, the particular gate or wire/net)

[25]For any one error model, Z_{loc} has a maximum value of 25, in total across all four models, the maximum value of Z_{loc} is 100 for any benchmark circuit.

[26]It is definitely true that the error localization results would vary with the type and location of error injection. We observed that gate replacement type of error injection requires a larger number of gates to be replaced for obtaining a buggy netlist which shows difference in the signal state of flip-flops. Typically, to obtain 25 iterations of such netlist, we require going through 40 to 70 iterations of error injection in which few lead to difference in the signal state of flip-flops while others show the same value as the golden signal states. Such iterations are required for other error scenarios also. Additionally, we observed that a gate replacement type error requires many gates in the netlist to be replaced to reflect difference in the signal states of flip-flops.

for all the design error scenarios is our future work. Note that error localization strongly depends on the injected error location. If it lies in the vicinity of trace buffer (TB), it can be easily localized. From Z_{loc} values in Table 5.7, it is clear that higher signal state reconstruction/restoration does not necessarily assist in localizing the chosen design bugs. However, expanded internal visibility becomes useful for localizing electrical bugs (modeled like bit-flips) or non-reproducible errors. Thus, it is desirable that higher signal state restoration be also considered as a goal for the combined trace signal selection along with the objectives of efficient error localization and routing-awareness.

5.9.4 Routing and Wire Length Measurement Results

In the beginning, a floor-plan is created as per the circuit size and area budget and then only the placement of all the circuit components (including the trace buffer) is done by the commercial tool to obtain the placement co-ordinates of all the candidate trace signals. Afterwards, all the circuit components (except trace signals) are routed which gives the routing congestion scenario. Then, the proposed routing and selection algorithm comes into play. The algorithm, $SelectFinalTrSignals$ selects the trace signals and subsequently their routing (to the trace buffers) is done. For a seamless integration and implementation, the proposed algorithm can be internally incorporated in any commercial tool. The trace buffers are placed at the end of the chip boundary so that off-chip dumping is quick because of the mismatch between chip execution speed and the dumping of the internal signals (flip-flops).

We have considered a regular arrangement of the trace buffers. However, there can be distributed trace buffers also in the design. The routing methodology remains same with slight adaptation for the distributed case. Table 5.9 shows the routing wire length (with the proposed congestion-aware algorithm, $TrSignalRouting$ i.e., Algorithm 11) for signal sets with TBw as 32. As expected, there is a wide variation between wire length of various set of signals for different benchmark circuits. The minimum routing wire length does not necessarily occur for any particular set of trace signals for all the circuits. Note that the wire route length of all the benchmark circuits appear to be in

the same range. The P&R tool (Synopsys IC Compiler) automatically chooses the die area as per the size of the circuits. We observed that the actual chip boundary (in which the routing and placement occurs) varies with the size of the benchmark circuits (for ten circuits, the co-ordinates are reported in Table 5.8 which act as end-points of the grid considering starting point as (0,0)).

We performed a second set of experiments with a higher area configuration to let the P&R tool choose a larger chip boundary for each circuit according to its size. However, due to the relative scaling of the wire length needed to route all the components (and accordingly the trace signals), the trend of variation of different signal selection techniques on route wire length remains same. We observed that there is no change in the relative routing congestion with different signal selection techniques when a larger layout area (and higher grid size) is adopted. When a larger size circuit size is placed in a size suitable for smaller ISCAS'89 circuits, the minimum wire length is in range of hundreds of microns. The wire length to route different trace signal (to the trace buffers) shows a variation from 1X to 3.4X from the minimum value.

Table 5.8: Grid co-ordinates after placement

Circuit	X co-ordinate	Y co-ordinate
$s5378$	62288	61832
$s9234$	68672	68520
$s13207$	135728	135272
$s15850$	137096	136944
$b21$	128408	127040
$p16c5x$	109864	108648
$s38417$	157440	157136
$s38584$	166560	165496
$s35932$	158808	158808
usb	164128	163824

In Table 5.9, entries in braces indicate the number of trace signals which can not be routed with in a given $Wlength$ budget. Thus, for $b17$ circuit, when signals selected as per technique in [1] are routed with the proposed algorithm, one out of 32 signals can not be routed with the given chip (grid, Gr) configuration. In this case, the wire length of remaining 31 signals is found to be 7524 units. All the distances are in microns.

Table 5.9: Total wire length (of original trace signal list)

$Name$	$S(H1)$	$S(H2)$	$S(M1)$	$S(M2)$	$S(M3)$
$b17$	7569	8276	7524(1)	8230	9579
$b21$	9865	9315	9758	9105	8460
$p16c5x$	9518(1)	9612(1)	9339	9572	8360(1)
$mips$	8478	7890	7988	6987	8956
usb	8445	6430	7585	6407	8450
$s5378$	9118(2)	10269	10320	10109	9583(1)
$s9234$	10196	9878	9203(2)	9679	9704
$s13207$	9016	9495	9739	9314	9034
$s15850$	9035	9497	8940	9190	9160
$s35932$	7719	8384	6927(7)	8341(1)	9616
$s38417$	8637	8628	9675	8417	8542
$s38584$	8893	8892	8405	8402	8782

Interestingly, for the case of $s35932$ circuit, for trace signals obtained from [1], as many as 7 of them can not be routed. We then perform the routing as per the complete signal selection algorithm, $SelectFinalTrSignals$ which takes as input signals twice the trace buffer width (2*TBw). It then performs an iterative search for signals to be routed from a sorted list (as per Manhattan distance) of 2*TBw trace signals.

For analyzing variation in routing wire length with the modified approach as compared to $TrSignalRouting$ algorithm, we consider an initial list of 64 (twice the trace buffer width) signals from each of the five trace signal methodologies: $H1$, $H2$, [1], [9] and [141]. As is shown in Table 5.10, the wire route length decreases because of the iterative search (in a successive manner) to discover profitable signals for minimizing the routing overhead. For some circuits, signal selections (having trace signals which could not be routed), the corresponding values in Table 5.10 are higher as now, all the signals are being routed leading to an increase in the total wire length. This can be seen in the case of [9] for $s35932$ circuit where total wire length increases from 8341 to 8378.

We repeated the routing experiments by deploying the trace buffers in the middle of the circuit for different trace signal lists. Under this scenario, as expected, the total wire length reduces because of lesser skew on the input trace signals. However, the routing congestion worsens when the trace buffers are placed in the center. We observed that regardless of the signal selection technique, some trace signals can not be routed to the

Table 5.10: Total wire length (proposed routing & selection technique)

$Name$	$S(H1)$	$S(H2)$	$S(M1)$	$S(M2)$	$S(M3)$
$b17$	7336	8264	7415	8184	9517
$b21$	9743	9227	9552	9002	8429
$p16c5x$	9473	9560	9311	9572	8909
$mips$	8402	7877	7889	6884	8887
usb	8432	6367	7098	6399	8354
$s5378$	9675	9867	10116	9728	9690
$s9234$	9990	9711	9852	9540	9629
$s13207$	9007	9483	9729	9309	9018
$s15850$	8453	9407	8910	9076	9020
$s35932$	7668	8289	8088	8378	9094
$s38417$	8607	8597	9061	8398	8535
$s38584$	8852	8852	8294	8402	8291

trace buffer even when initially $2*TBw$ signals are selected for routing. The number of signals which can not be routed to trace buffer varies from one to eleven in the case of proposed heuristics and restorability maximization techniques. The worst case scenario occurs for $s35932$ circuit when as many as 19 signals (for trace buffer width of 32) can not be routed for the signals obtained from technique in [1]. This observation justifies placing the trace buffer at the periphery. It is worth to mention that we performed static timing analysis (STA) of the final netlist (after the insertion of trace buffer and subsequent routing) to assess the impact on circuit performance. We observed that the targeted clock frequency can be met for the final netlist. This shows that the design performance is unaffected by the proposed routing methodology. However, minimization of critical length and congestion-aware routing of trace signals to the trace buffers can be considered in future for simultaneous optimization.

5.9.5 Combined Trace Signal Selection Results and Analysis

5.9.5.1 Obtaining a Signal List by Combining Individual Signal Sets

As per $SelectFinalTrSignals$ algorithm, a combination of signals with input values of a, b and c needs to be created depending on user preferences. For the purpose of illustration of this combination, we chose a simple scenario with $a = 0.3$, $b = 0.5$ and $c =$

0.2. This particular choice of a, b and c is meant for achieving better error localization.[27] Accordingly, these signals are stored in $TrSignals'$ list from where the final trace signals, $TrSignals$ (S_{final}) are selected. Some signals which are selected with $c = 0.2$ are common with signals selected from 0.5 fraction of S_{ed} and 0.3 fraction of S_{res}.[28] Note that for the purpose of combined signal selection, we chose S_{res} which has maximum restored signal states out of signals with [1],[9] and [141]. For S_{ed}, we chose set of signals selected with Heuristic-2 as they are observed to be slightly better in localization compared to signals selected with Heuristic-1. We first perform the routing of the candidate trace signals ($TrSignals'$) and based on the routing overhead minimization, the final trace signals ($TrSignals$) are obtained. Restorability calculations are then done on signals in $TrSignals'$ (32 in number) for 1024 clock cycles (which is the trace buffer depth here). Note that we begin with a list of 64 (2*TBw) signals and end up with the routing of 32 signals only to meet the target trace buffer width ($TBw = 32$).[29]

5.9.5.2 Results of Combined Signal Selection

For the combined list of trace signals, $Restored\ fraction(\%)$ values are reported in the second column of Table 5.11 and the corresponding maximum Rf (out of the five signal lists) are reported in 3^{rd} column denoted by max. Since the final list (S_{final}) is constituted with some signals from either of [1],[9] or [141] and $S(H2)$, the $Wlength$ (sum total of actual length of wires routed to the trace buffer) lies between minimum and maximum of the corresponding values of these lists. For each circuit, the maximum $Wlength$ (out of the five signal lists) computed from values in Table 5.10 are reported in 5^{th} column of Table 5.11. For most of the circuits, we achieved substantially higher $Restored\ fraction$ values as compared to the signals as per the proposed heuristics ($S(H1)$ or $S(H2)$).

[27] Signal sets for such choices of a, b and c can be obtained by the iterative partitioning of the individual set of signals as presented in Algorithm 10.

[28] Essentially, for many circuits, this choice resembles choices of $a = 0.5$, $b = 0.5$ or $a = 0.4$, $b = 0.6$ and $a = 0.4$, $b = 0.5$, $c = 0.1$.

[29] The final results of this combined signal selection depend on the manner in which the trace signals are arranged in the combined pool of signals. Signals corresponding to the choice of a, b and c can be arranged in six different ways. We report the results when signals are arranged in the order: first signals of S_{res}, signals of S_{ed} and then signals of S_{ra}.

Further reduction in $Wlength$ can be obtained by increasing the value of c; however, this may severely impact the signal state restoration and the subsequent error localization.

Table 5.11: Results for combination, S_{final}(S) with $a = 0.3, b = 0.5, c = 0.2$

$Name$	$Rf(S)$	$Rf(max.)$	$Wl(S)$	$Wl(max.)$	$Z_{loc}(S)$
$b17$	11.04	15.63	8306	9517	68
$b21$	23.04	25.13	8640	9743	63
$p16c5x$	16.71	19.68	9552	9572	67
$mips$	86.14	96.71	8456	8887	67
usb	42.87	45.08	8255	8432	59
$s5378$	58.57	62.39	9788	10116	63
$s9234$	73.71	82.76	9924	9990	60
$s13207$	72.63	79.80	9160	9729	62
$s15850$	79.17	85.09	9356	9407	61
$s35932$	57.43	84.67	8573	9094	51
$s38417$	5.31	12.87	8652	9061	45
$s38584$	78.82	82.97	8416	8852	66

As can be seen from Table 5.11, for some circuits, we achieved quite close Rf to their maximum values (out of the five trace signal lists). This is essentially because greedy selection (like [1]) suffers from diminishing restoration ratio (RR) effect as explained earlier. Thus, the main contribution to restoration is because of some signals only. If they are included in the final list (S_{final}), signal restoration can be significantly high as compared to the signals with $H1$ or $H2$ methods. To examine the utility of the combined signal set towards error localization, we injected the same errors corresponding to $e1$, $e2$, $e3$ and $e4$ scenarios (each having twenty-five iterations totaling 100 iterations for each circuit) as reported in Table 5.7. We attempted localization of these errors with the set of signals in S_{final} as per the proposed debug methodology. Z_{loc} values are reported in the last column of Table 5.11. These values are higher than that of Table 5.7 because the inter-mixing of signals from different signal sets assists in localizing the errors which may have been missed out even by all the signals from the same set (list). Interestingly, we observed similar trend when the final signal selection is carried out by combining with signals obtained from $H1$. We also observed that the final results of error localization vary significantly with the combination of trace signals (parameters a, b and c). We could not achieve complete error localization (i.e., Z_{loc} values as < 100 in Table 5.11)

which shows that the chosen values of combination parameters (a, b and c) is definitely not the best for error localization purpose.

5.9.6 Different Perspectives on Trace Signal Selection

5.9.6.1 Varied Utility of Trace Signal Selection Methodologies

The proposed approach of signal selection can be utilized for on-line error detection in addition to post-silicon validation. The scheme outlined in [8] requires a selection of signals for on-line design error detection for processors. For error detection in SoCs, selection of signals of various models based on the proposed methodology can be adopted. Even though we attempted design bug localization only, Iwata et al. have shown that the proposed approach of error detection aware trace signal selection is helpful for localizing bit-flips to a significant extent [24].

5.9.6.2 Path Diversity in Topological Ranking of Signals

We have proposed a signal selection algorithm which analyzes common gates on the paths between various flip-flops and refines assignment of error transmission capabilities of these flip-flops in [119]. We observed that this method becomes computationally intensive without yielding much impact on error localization for design errors. Similar to the heuristic utilized in [119], we experimented with ranking of signals by considering various S-graph related factors like those of graph centrality. These refinements have very little impact on end results in the debug process.

5.9.6.3 Alternatives to Conventional State Restoration Technique

The basic principles of signal state restoration techniques need to be investigated to achieve higher restoration since mere application of *forward propagation* and *backward justification* rules are not sufficient for the maximization of signal state restoration [166]. Furthermore, except exhaustive simulations, there is no method to quantify the maximum restoration available from a given set of trace signals. Thus, the absolute profitability of a set of trace signals for the purpose of restoration maximization can not be

ascertained. We observed the dependence of ordering of trace signals in the signal list during our experiments on several circuits. Some of the previous work have considered the order in which signals are present in the trace signal list [167],[141, 165] for signal state restoration maximization. Investigating the ordering of trace signals on other aspects like error localization is a promising goal in this direction. A new approach has been proposed by Cheng et al. [168] in which groups of flip-flops (which are referred to as clusters) in the design are identified. Based on the logical relation between the cluster elements (flip-flops) and the inputs to the cluster, signal states of all the untraced cluster flip-flops can be known. However, the methodology involves the tracing of the initial values of the flip-flops in the cluster, following which the signal states can be restored for any number of clock cycles.

5.10 Conclusion

This chapter proposed a topology based signal selection which is directed towards identifying signals useful for early and maximum error detection. The proposed methodology of topology-based signal selection is refined by incorporation of two heuristics aimed at error detection. These heuristics are combined together with the metric of signal state restoration to obtain an enlarged view of internal visibility and achieving enhanced error localization. Additionally, the proposed signal selection algorithm is routing congestion-aware and hence incurs minimal routing overhead. The combined selection of trace signals from different sets improves the effectiveness of the selected signals. Since, we could not achieve error localization in all the iterations (of error injection) during our localization experiments, we explore learning-based techniques in the next chapter.

– * – * –

Chapter 6

Learning-assisted Gate-level Error Localization Techniques

6.1 Introduction

An efficient bug localization process at the post-silicon stage helps significantly in avoiding re-spins of the design. However, in the present era of designs with tremendous complexity, the limited accessibility of internal signals of design is the one of major obstacles in quick bug localization. Machine learning strategies hold promise of assisting us in revealing subtle intricacies of correlation between different components of the debug data. We explore some aspects of such learning strategies for effective design error localization inspite of restricted observability.

We present an alternative methodology for enhancement of the observability of internal signals at the post-silicon stage through the usage of a learning algorithm. Based on mock simulation of the designs, a nearest neighbor model is developed which is independent of the characteristics of the particular design. This model (i.e., learning strategy and no. of neighbors) can then be utilized for finding out nearest neighbors of flip-flops of any arbitrary design given the traced and restored data corresponding to a post-silicon test execution. This ensures full scalability of the proposed technique to industrial-sized large circuits. The basic premise of the proposed methodology lies in the fact that the

unknown signal states can be decided based on the signal state of its neighbors. To the best of our knowledge, this is the first approach which targets the discovery of complete set of signal states and achieves the expanded observability with reasonable accuracy.

We propose an error localization methodology at the gate-level to assist the debug engineer in analyzing the root-cause of failures at the post-silicon stage. We synthetically inject errors into the netlist and based on the circuit response (i.e., signal states of flip-flops of the design), a classification model is developed for analysis. The basic principle of this approach is to classify a smaller region of the netlist into buggy or non-buggy based on the developed model. One of the intended bug injection methods is random gate replacement technique [55, 163] which acts as a representative of many logic bugs. Considering the large number of gates in modern designs, replacement of each gate with a random one would turn out to be highly exhaustive. Therefore, we iteratively perform error injection into very small portions of the design (netlist) and then apply the classification analysis for building the model. The localization experiments are performed by utilizing responses (i.e., flip-flop state values) gathered from the on-chip trace buffers. We obtain a list of flip-flops (corresponding to erroneous netlist regions) infected by the error (injected during testing) which are ranked based on a pre-defined scoring mechanism.

The remainder of the chapter is organized as follows. Section 6.2 summarizes related work on post-silicon error localization techniques at different abstraction levels of design and the application of machine learning in error localization. Section 6.3 presents our proposed methodology of learning-based complete visibility expansion and different terminologies associated with it. Section 6.4 explains the proposed gate-level error localization methodology based on the expanded signal states. Section 6.5 presents the results of observability expansion and error localization by using the proposed technique. Section 6.6 describes our methodology of design response-based model building in detail. Section 6.7 explains our testing methodology for a given erroneous design based on design response model building. Section 6.8 finally concludes the chapter.

6.2 Post-silicon Observability and Error Localization with Learning Techniques

6.2.1 Maximal Post-silicon Observability Expansion

Previously proposed techniques of state restoration (i.e., the discovery of unknown signal states based on the traced signal states) attempt to solve this through analysis of the design netlist and decide values of untraced signal states with the help of structural dependencies [1, 115, 169]. However, a very important aspect of this process is the choice of trace signals as the technique of backward justification or forward propagation can reveal the untraced signal values only if appropriate hints (in the form of values of structurally related signals) are supplied. Therefore, attempts have been made to filter out best possible set of trace signal candidates through heuristics based on design/netlist analysis [22, 104, 109, 115, 170]. However, attempting only state restoration for expanding the observability is not sufficient. This is because we have observed in our experiments that almost (or more than) 60-70% of the internal signal values are always unknown for large circuits even though a highly effective signal selection technique is utilized.[1] This necessitates observability expansion after the application of signal state restoration. We noticed that the objective of design observability expansion can be regarded as a variant of data mining problem. In this regard, clustering algorithms like nearest neighbors can assist in deciding the values of unknown signals (which are either untraced or not restored) as they capture the correlation between data values. We exploit this fact to achieve *maximal* (i.e., full with reasonable accuracy) internal visibility.

6.2.2 Relevance of Learning Techniques in Post-silicon Error Localization

Techniques such as *Instruction Footprint Recording and Analysis (IFRA)* [69] or *Reversi* [65] assist in error detection in processor systems using low cost on-chip recorders as an

[1] This can be observed from the results in Table 5.4 of Chapter 5.

observability mechanism or inversion-based program instrumentation respectively. The authors in [70] perform program analysis after constructing bug localization graphs for error localization to a block-level granularity. However, typically an architectural block in a processor may contain thousands of gates. This makes it very difficult to debug at the gate-level and necessitates devising other techniques of low-level localization. In the recent years, machine learning techniques have been explored for the purpose of bug triaging or error localization at both the pre-silicon and post-silicon stage (gate-level granularity). Since during post-silicon validation, a large number of tests can be applied, the amount of logs collected can become very large. Therefore, machine learning techniques like clustering, regression analysis etc., can be suitably deployed to extract hints for bug/error localization from the obtained test response logs. Furthermore, because of the limited accessibility of internal signals at the post-silicon stage, the extracted signal dumps (even if they are corresponding to larger execution cycles) may be incomplete in nature. DeOrio et al. [90] proposed a post-silicon bug diagnosis methodology based on data collection from failing tests and then applying a clustering technique to form different signal groups. A mechanism to detect root-cause signals by identifying "anomalous" signals from post-silicon tests and iterative selection of signals to be monitored has been proposed by Bertacco et al. [91] on lines similar to that of [90]. For post-silicon bugs that manifest inconsistently over repeated executions of the same test and have non-deterministic behavior, Khudia et al. [92] have classified total internal signals into *passing groups* and *failing groups* for error localization. Using big data techniques, the author in [171] attempt error localization from post-silicon signal dumps without resorting to the requirement of golden responses. They utilize structural dependencies from RTL descriptions to group the failing post-silicon traces into different groups which are then individually analyzed for the purpose of localization. Machine learning techniques like clustering have also been applied for dynamic trace signal selection to enhance fault detection with the help of information obtained from trace buffers by Zhu et al. [172] with further optimization by using linear programming. Apart from post-silicon error localization, a lot of work has been done in the last decade for the application of learning

techniques for the analysis of yield failures during manufacturing test [173–176].

6.3 Proposed Methodology of Visibility Expansion

6.3.1 Methodology Illustration

As outlined in Section-6.1, limited observability is one of the main obstacles in the process of post-silicon error localization. This becomes more challenging at the gate-level or at the granularity of flip-flops since the usage of observability enhancement mechanisms provides the states of very few flip-flops. This large mismatch between the number of internal signal values available to us during simulation phase and post-silicon phase makes the debug process difficult. To cope up with this wide disparity of internal signal values, method of *matrix completion* (completing a partially-filled matrix) is used [177]. We formulate the observability expansion as a machine learning based clustering (particularly, *nearest neighbors*) problem. The k-nearest neighbors (kNN) technique is a supervised machine learning algorithm, where k depicts the number of neighbors. This algorithm assists in obtaining signal states similar to the known (i.e., traced or restored) flip-flop values. Once the neighbors of a particular signal are identified, the state of that particular flip-flop can be obtained. The state (either 0/1) of a particular flip-flop in any clock cycle is assigned the value most common among states (which are known either through restoration/tracing) of its k-nearest neighbors. Continuing in this manner, the state of all the flip-flops for all clock cycles can be obtained, leading to full internal observability. The *NN* algorithm decides the closely related neighbors (in this case flip-flops) based on some information pertaining to the design. It can be either related to the structure of the design or simulation values. For the sake of illustration, we consider the circuit shown in Figure 6.1 where six flip-flops are depicted as A, B, C, D, E and F. The procedure of neighbor-finding and subsequent signal prediction can be applied to netlists in which synthetic design bug(s) have been injected. Note that simulation values of bug injected netlist are not a necessity for the success of this method. However, signal values of a buggy netlist provide more useful behavior compared to the original netlist as

the logic inconsistencies are pin-pointed. Without any loss, pre-silicon signatures (i.e., signal states of flip-flops) of fully correct netlist can also be utilized for the purpose of neighbor finding (which constitutes the learning/training step).

Nearest neighbor algorithm works on discovering some kind of metric (distance) between the targets. One such metric is the Euclidean distance[2] which is the measure of dissimilarity between different data points. Consider that after simulation of the netlist illustrated in Figure 6.1, we obtain the Euclidean distance between all of these flip-flops is shown in Table 6.1. The lower the Euclidean distance, closer the neighbor is to a particular signal (flip-flop). With nearest neighbor algorithm, 4 neighbors obtained for each flip-flop are shown in Table 6.2. If the flip-flops of last column are removed from the above set, we obtain the set of 3 neighbors. Depending on the state of these neighbors, the X values can be determined. This is essentially done by a majority vote of the known values of neighbors at the particular instant. For instance- if 2 neighbors of a flip-flop have "1" and the third has value "0", the target flip-flop is assigned a value of "1". In cases of a tie-up between neighbor values, we assume "1" as the default value. Note that the signal state of neighbors of a target flip-flop is dynamically updated as the discovery of values progresses. The difference in the distances of the neighbors (Table 6.1) justifies the choice of nearest neighbors (abbreviated as "nbr") that are depicted in Table 6.2.

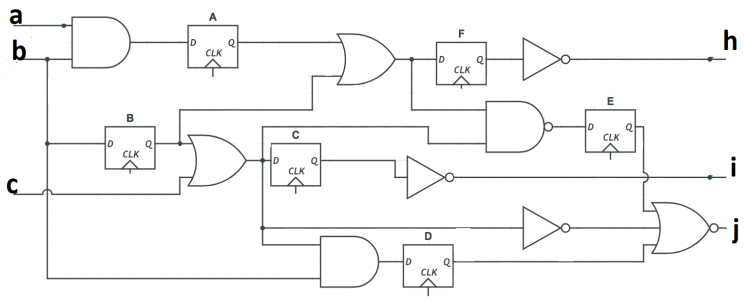

Figure 6.1: Example circuit for illustrating methodology

As stated earlier, based on the mock simulations of the netlist at the pre-silicon stage, the neighbors are to be identified. Let's assume that for a certain input sequence, we

[2]computed by $\sqrt{\sum (x-y)^2}$ where x and y are data points. In our case, x corresponds to the signal states of one flip-flop while y corresponds to the signal states of a different flip-flop.

obtain the states of all signals (flip-flops) of design (i.e., netlist) shown in Figure 6.1 for a fixed number of clock cycles.[3] Thereafter, Euclidean distance between different signals leads to the values shown in Table 6.1. These values can also be computed by averaging for a fixed number of iterations (say, 100) of mock simulations of the design (with changing the inputs in each iteration of the mock simulations).

Table 6.1: Euclidean distance between different signals for circuit in Figure 6.1

FF↓→	A	B	C	D	E	F
A	0	1.42	2.65	2.65	1.42	2.65
B	1.42	0	2.24	2.4	0	2.24
C	2.65	2.24	0	0	2.24	0
D	2.65	2.24	0	0	2.24	0
E	1.42	0	2.24	2.24	0	2.24
F	2.65	2.24	0	0	2.24	0

Table 6.2: Obtained nearest neighbors for example circuit

FF↓	nbr1	nbr2	nbr3	nbr4
A	B	E	C	D
B	A	E	C	D
C	D	F	B	E
D	C	F	B	E
E	B	A	C	D
F	C	D	B	E

We performed a simulation of this netlist (Figure 6.1) for 10 cycles under different inputs.[4] Table 6.3 shows states of six flip-flops for these ten clock-cycles.[5] Flip-flop A & C were traced and their states are known for all ten cycles. Through the usage of state restoration technique [115], some states of other flip-flops can be computed. The state of many flip-flops can not be discovered. Those states are depicted as X. Note that it

[3]For brevity, we do not show here the signal state of the flip-flops for different clock cycles.

[4]This is done for the purpose of illustration only. In real application, the tracing would be done with a buggy netlist (i.e., a design bug injected in Figure 6.1). In results shown later in this chapter (Figures 6.3,6.4,6.5), we have performed tracing on a synthetically bug injected (through wire exchange) netlist for each benchmark circuit.

[5]Note that it is difficult to fully showcase the merit of this method through a miniature example because here many signal states have already been reconstructed through signal state restoration.

is a common assumption in silicon debug to not trace the inputs because of excessive storage requirements due to run-time ranging from hours to days [178].

Table 6.3: Restored and traced states for illustration

FF→	A	B	C	D	E	F
c1	**0**	X	X	**0**	X	X
c2	**0**	X	X	**0**	X	X
c3	**1**	1	X	**1**	1	1
c4	**0**	X	X	**0**	X	1
c5	**0**	X	X	**0**	1	1
c6	**0**	1	1	**1**	1	1
c7	**0**	X	X	**0**	1	1
c8	**0**	X	X	**0**	1	1
c9	**0**	1	1	**1**	1	1
c10	**0**	X	X	**0**	1	1

The completely expanded (fully known) states are shown in Table 6.4, done with a choice of 3 nearest neighbors for each of the flip-flop (signal). However, as expected, the derived signal states are different from the actual ones in few cases. These cases may vary with the choice of number of neighbors. For instance, in 2^{nd} clock cycle, state of FF-C is different from the actual signal state when the expansion is done either with the help of 3-neighbors (or 4-neighbors). We obtained "0" (which is shown in braces) in this case whereas the actual value is "1". Note that the accuracy in predicting signal states depends significantly on the traced and restored signal states.

Table 6.4: Completely expanded internal signal states

FF→	A	B	C	D	E	F
c1	**0**	0	0	**0**	0	0
c2	**0**	0	1(0)	**0**	1(0)	1(0)
c3	**1**	1	1	**1**	1	1
c4	**0**	0	0	**0**	0	1
c5	**0**	0	1	**0**	1	1
c6	**0**	1	1	**1**	1	1
c7	**0**	0	1	**0**	1	1
c8	**0**	0	1	**0**	1	1
c9	**0**	1	1	**1**	1	1
c10	**0**	0	1	**0**	1	1

It is imperative that the inaccuracy in prediction (i.e., the number of derived signal states which are different from the actual ones) would vary from one circuit to another apart from the choice of neighbors. This is definitely a drawback of this technique of visibility expansion. However, we observed in our experiments that the expanded visibility is indeed useful for the purpose of error localization.

6.3.2 Algorithmic Description of Visibility Expansion

The terminology utilized for formally explaining the proposed methodology is presented in Table 6.5. In the proposed methodology, we develop a model, M_{nbr} by two training methods presented Algorithms 14 and 15. For the first method, we utilize the error

Table 6.5: Notations and their meaning

Term	Meaning
M_{nbr}	k-NN learning model
TBw	width of trace buffer
D_{tb}	depth of trace buffer
L_i	i^{th} Nearest Neighbor technique
K	no. of iterations of learning
ϕ_j	numbers of nearest neighbors in M_{nbr}
Cy_{iter}	number of cycles in each iteration of learning
F_{tot}	number of flip-flops in design (netlist)
FF_e	error signature of design
ξ_i	features of the design(netlist)
FF_t	flip-flop signature(s) used for testing
GFF_t	reference/golden flip-flop(s) signature

signature obtained from the chip (netlist). In the second method, we employ structural relationships to obtain nearest neighbors. Observability expansion can be achieved by either of these methods. Based on the traced signal states (which assists in obtaining restored signal values) and the obtained neighbors, states of all the flip-flops can be discovered. An very important consideration is to figure out the optimum number of nearest neighbors. Therefore, we iterate the neighbor finding exercise for certain number of times taking the minimum no. of neighbors as 10 and maximum as 1400 for different

benchmark circuits. The procedures shown in *Learning*1 and *Learning*2 expand the observability of internal states and measure the difference between the expanded signature and the reference complete signature (i.e., golden signal states).

There can be many methodologies to derive nearest neighbors based on the specific implementations. For the purpose of finding nearest neighbors, we utilize *Scikit-learn* [179] library. We have implemented 3 types of nearest neighbor algorithms:[6] (*K-D Tree*, *Ball Tree* and *Brute Force*). By normalizing values of specific parameters in these algorithms their variants are obtained. These are denoted by L_i in both these procedures. For each of the NN algorithm-variants, we perform training on 8 or 9 configuration as no. of neighbors (denoted by ϕ_j) as 10, 20, 50, 75, 100, 200, 250, 275 and 300 for some circuits while 50, 100, 200, 250, 300, 400, 500 and 600 for other circuits. We take Cy_{iter} as 1024 which is the typical trace buffer depth (D_{tb}).[7]

Algorithm 14: *Learning*1

Input: FF_e,K
Output: M_{nbr}

$M_{nbr} \leftarrow \emptyset$;
$I \leftarrow$ incomplete states (restored and traced ones) for Cy_{iter} cycles;
$FF_e \leftarrow$ state of F_{tot} flip-flops for Cy_{iter} cycles;
$score_{\phi_j,L_i} \leftarrow 0$;
for *each* L_i **do**
  for $z = NN\ (\phi_j)$ **do**
    for w =*1 to* K **do**
      $nbrs \leftarrow$ NNmodel(FF_e,L_i);
      $fullval \leftarrow$ fillvalues($nbrs$,I);
      $score \leftarrow fullval$ - FF_e;
    end
    $score_{\phi_j,L_i} \leftarrow \sum$ score;
  end
end
$nbrscore =$ minimum($score_{\phi_j,L_i}$);
$M_{nbr} \leftarrow NN(\phi_j, L_i)$;

[6] In particular, these refers to the manner in which data points are internally organized and partitioned for the purpose of distance calculation. It is apparent that with varying manner of distance computation, the computation of neighbors significantly varies.

[7] Cy_{iter} could be different from D_{tb} as well.

As it is seen in the procedure *Learning*1, finding out the nearest neighbors depends on the complete error signature of chip, FF_e (derived from a reference simulation) and the particular learning method (L_i).[8] The After particular neighbors (corresponding to specific (ϕ_j)) are identified, the incomplete signal states of the signature (restored + traced) can be predicted leading to maximal expansion of internal observability. The best configuration is the combination of ϕ_j and L_i for which the lowest difference between the expanded signature and the reference complete erroneous signature (FF_e). $nbrscore$ is the sum of variation between actual and discovered states for this particular choice of ϕ_j and L_i.

We propose a second procedure to develop the model (M_{nbr}) utilizing only the netlist and thus, independent of error signatures. Essentially, only static features of the design description are accounted for in this case. To capture the structural characteristics of any flip-flop ff_i, we define following features on the lines of [170]. For each flip-flop ff_i, a set of traits which are called as features is shown in Table 6.6.

Table 6.6: Features and their meaning

ξ_1(fan-in)	number of different flip-flops connected to ff_i in its fan-in
ξ_2(fan-out)	number of different flip-flops connected to ff_i in its fan-out
ξ_3(gate count)	number of gates in connection to ff_i in its fan-in and fan-out cone
ξ_4(2^{nd} level connectivity)	average of number of different flip-flops in second level fan-in and second level fan-out
ξ_5(ed score)	$\sum_{p=0}^{p=P}(\prod_{n=0}^{n=Total}(score\ of\ each\ gate))$

While majority of the features listed above are fairly straightforward and can be easily computed, the last feature is taken from our work[9] in [116] which approximately calculates the error propagation score of each flip-flop. An estimation of the error transmission through each gate can be given by $1/N_{inputs}$, where, N_{inputs} is the number of

[8] As stated before, the three methods of learning which we attempted are *K-D Tree*, *Ball Tree* and *Brute Force* available in *Scikit-learn*[179] library. The details of these specific implementations can be obtained from [180–182].

[9] This work of ours is a precursor to Algorithm 9 of Chapter 5.

inputs of the gate. Once the score of each gate is calculated, by counting all the incoming paths to different flip-flops, we can approximately find *ed score*. Further, we need to consider the score of each gate in a multiplicative manner (no. of gates represented by $Total$) and sum these scores along all the incoming paths (no. of paths generalized by P) to get the rough estimate of error transmission ability of each flip-flop. As it is

Algorithm 15: *Learning2*

Input: $Netlist$,K,F_{tot}
Output: M_{nbr}

$M_{nbr} \leftarrow \emptyset$;
$Feature\ space \leftarrow \emptyset$;
$I \leftarrow$ incomplete states (restored and traced ones) for Cy_{iter} cycles;
$FF_e \leftarrow$ state of F_{tot} flip-flops for Cy_{iter} cycles;
$score_{\phi_j,L_i} \leftarrow 0$;
for *flip-flop* 1^{st} *to* F_{tot} **do**
    Calculate all features (ξ_1,ξ_2,ξ_3,ξ_4,ξ_5);
    $Feature\ space \leftarrow$ features of flip-flop;
end
for *each* L_i **do**
    for z = *NN* (ϕ_j) **do**
        for w = *1 to* K **do**
            nbrs $\leftarrow$ NNmodel($Feature\ space$,L_i);
            $fullval \leftarrow$ fillvalues($nbrs$,I);
            score $\leftarrow$ $fullval$ - FF_e;
        end
        $score_{\phi_j,L_i} \leftarrow \sum$ score;
    end
end
$nbrscore$ = minimum($score_{\phi_j,L_i}$);
$M_{nbr} \leftarrow NN(\phi_j, L_i)$;

seen in *Learning2*, finding out the nearest neighbors depends on the computed features (depicted by $Feature\ space$) from the netlist and the particular learning method (L_i). Thus, unlike *Learning1* procedure, closely related neighbors are discovered statically without resorting to the analysis of data (i.e., error signatures) from simulation. Intuitively, it appears that *Learning2* would provide better accuracy in the prediction of unknown signal states. However, we observed in our experiments that *Learning2* fails to achieve so because of the inaccurate learning in this approach.

In our experiments, we reached 100% internal visibility expansion with varying accuracy. This allows us to have better visibility of internal signals and thus better debugging of errors. Assuming that we have a "golden" reference model constructed out of a completely verified higher level abstraction of the design-under-validation, we can exactly localize the design bug/error. The overall methodology for error localization with the complete signal visibility shown in Figure 6.2.

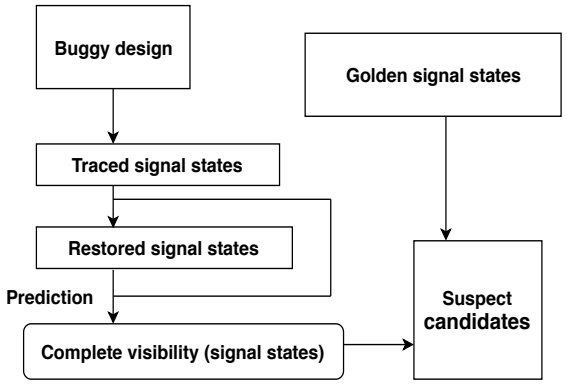

Figure 6.2: Error localization with expanded visibility

6.4 Coarse-grained Error Localization Methodology

Localizing at gate-level is difficult due to the restricted visibility of internal states of the chip. Therefore, with the expanded observability by the proposed method, we aim to achieve error localization at the granularity of flip-flops. The debug methodology is formally expressed[10] as Algorithm 16.[11] The basic principle of the debug methodology is to catch the difference between the reference signature and the expanded signature. Since, the unknown signal values (either untraced or non-restored) have been resolved, the differences can be more clearly pin-pointed now. For localizing to smaller portion of the design, we divide the complete error signature (consisting of F_{tot} flip-flops and D_{tb} cycles) into certain number of blocks (which are decided by a factor termed as $bfactor$). Based on the dissimilarity between chunks of flip-flop values (denoted by *data block*), we

[10]This is similar to Algorithm 13 of Chapter 5.
[11]Here, $diff$ function simply calculates the difference in signal states.

Algorithm 16: *CompleteVisibDebug*

Input: FF_t,GFF_t,$bfactor$
Output: *Error blocks*

$FF_t \leftarrow$ flip-flop signature used for testing;
$FF'_t \leftarrow$ completed FF_t by $Learning1$ or $Learning2$;
$N_t \leftarrow$ length of FF_t;
$nffblocks \leftarrow N_t/bfactor$;
$\{block\ data1, block\ data2\} \leftarrow \emptyset$;
$GFF_t \leftarrow$ state of F_{tot} flip-flops for Cy_{iter} cycles;
for $ffblock\ 1^{st}$ *to* $nffblocks$ **do**
    $block\ data1 \leftarrow$ portion of FF_t;
    $block\ data2 \leftarrow$ portion of FF'_t;
    $score_{ffblock} =$ diff($block\ data1$,$block\ data2$);
    $\{block\ data1, block\ data2\} \leftarrow \emptyset$;
end
Rank $ffblocks$ in decreasing order of $score_{ffblock}$;
Error blocks $\leftarrow$ Top ranked $ffblock$;

can localize to a smaller region of the netlist. This comparison leads to a ranking of all the $ffblocks$ out of which we choose the top ranked $ffblock$. Because of the inaccurate signal state prediction, some flip-flops which are not among the actual suspects may also appear as candidate suspects. This elongates the debug process and degrades the quality of error localization.

6.5 Observability Expansion Formulation & Results

6.5.1 Experimental Setup

As mentioned earlier, *Scikit-learn* [179] library has been used for implementing the proposed methodology. We chose circuits from ISCAS'89, ITC'99 and *Opencore* benchmark suite shown in Table 4.4 of Chapter 4. We performed 100 iterations of learning on the circuits to select the appropriate no. of neighbors (ϕ_j) and type of learning (L_i).[12] We

[12]The training (i.e., building the model, M_{nbr}) was done with the simulation of circuits for inputs which are not used in the error localization stage. Additionally, we performed the neighbor finding exercise with the help of correct netlists and netlists with synthetically injected errors (wire exchange). The latter is intended to bring out more differentiating circuit responses which can assist better in neighbor discovery.

chose our configuration as TBw as 32 and D_{tb} as 1024.

6.5.2 Internal Observability Expansion Results

After achieving observability expansion, we validate the obtained signal states by comparing with the reference complete signature in each iteration of the learning experiment. We report the dissimilarity in the actual and expanded signal states obtained by the method *Learning*1 in Figures 6.3 and 6.4. The maximum accuracy occurs for a particular number of neighbors. When we choose very less number of neighbors like 5 or 10 in M_{nbr}, accuracy is very poor. Similarly, if very large number of neighbors like 500 is chosen, the accuracy again falls.[13] Therefore, for most circuits, there is a non-monotonic variation in accuracy with the number of neighbors. As expected, there is a continual increase in the time (measured in seconds) spent in unknown signal state discovery as the number of neighbors increases. However, the increase in time spent in signal state prediction is linear with the increase in the number of neighbors in M_{nbr}.

Let us consider that the traced and restored signal states are represented by T and R respectively. Even after restoration, the signal states which are left unknown are represented by U. It is obvious that total signal states (S) are given by $T + R + U$. Also, suppose that signal states incorrectly predicted (or, mispredicted) by the discovery process are represented by I (representing incorrect). With this terminology, the inaccuracy can be given by $\frac{I}{S}$. However, the inaccuracy can be more suitably calculated as $\frac{I}{U}$ which can tell us the fraction of unknown signal states which are wrongly predicted by the proposed technique of visibility expansion. The results of this metric are shown in Figures 6.3 and 6.4. It is obvious that lower the values of this parameter ($\frac{I}{U}$), better is the prediction of unknown signal states. For smaller circuits, we consider variation of number of neighbors ($nbrs$) from 10 to 300 in certain random steps. Similarly, for large circuits, we consider variation of number of neighbors ($nbrs$) from 50 to 600. To further confirm non-monotonicity with the increase in number of neighbors, we computed $\frac{I}{U}$ (in %) taking $nbrs$ as 750, 800, 900, 1200 and 1250 for $s38584$, $s38417$ and usb circuits. This

[13] In *Scikit-learn* implementation, the best accuracy is achieved with *K-D Tree* learning choice.

variation is shown in Table 6.7 for the three large benchmark circuits. The inaccuracy in these cases are lower than that of the respective percentages shown in Figure 6.4.

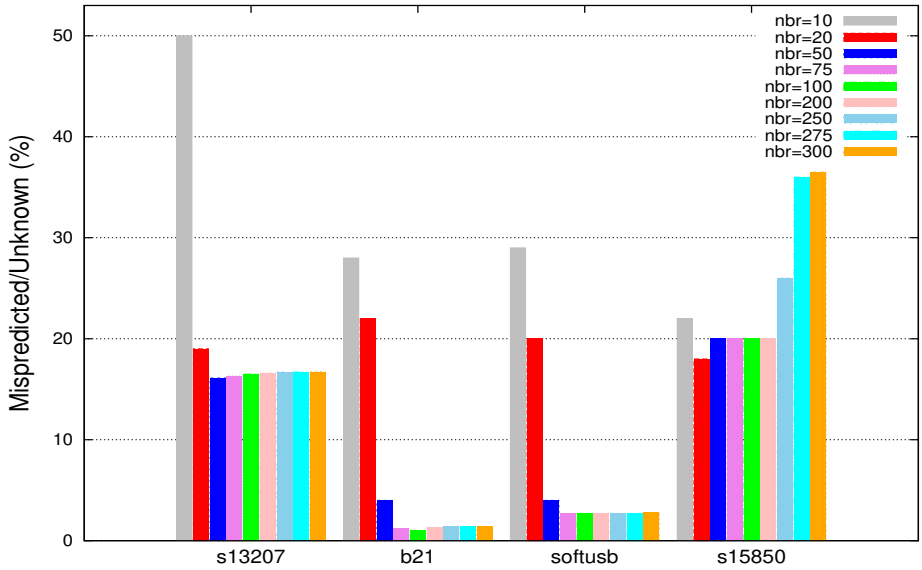

Figure 6.3: Inaccuracy ($\frac{I}{U}$ in %) with *Learning*1 (*nbrs*: 10 to 300)

Table 6.7: $\frac{I}{U}$ (in %) with increased neighbors ($nbrs > 600$)

circuit	750	800	900	1200	1250
*s*38417	21.58	22.21	21.61	23.15	23.18
*s*38584	36.85	37.51	38.78	41.04	41.10
usb	17.88	17.93	17.88	17.88	17.94

We observed a poorer accuracy with *Learning*2 as compared to *Learning*1. From the variation of inaccuracy with neighbors (*nbrs*) shown in Figure 6.5, a non-monotonic relationship can be observed, however with a higher inaccuracy as compared to *Learning*1. For *s*38417, we evaluated the inaccuracy for neighbors up to 1400. The metric $\frac{I}{U}$ (in %) for 750, 800, 860, 900, 1200, 1250, 1300 and 1400 evaluates to 27.9, 27.8, 27.7, 27.4, 26.2, 26.3, 26.3 and 26.7 respectively. However, the lowest value of 26.2% inaccuracy is lower than 20% which is obtained with *Learning*1. For *s*38584 circuit, the metric $\frac{I}{U}$ (in %) for 950, 1200, 1250, 1300 and 1400 evaluates to 41.37, 41.22, 41.22, 41.20 and 41.19 respectively. These values are lower than that of the respective percentages shown in Figure 6.4 for *Learning*1 showing continual decrease with increase in *nbrs*.

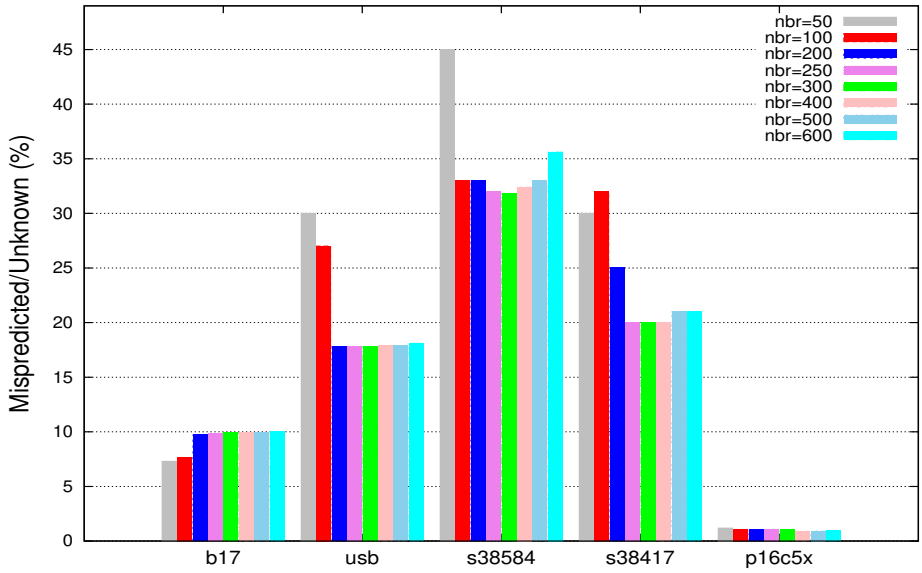

Figure 6.4: Inaccuracy ($\frac{I}{U}$ in %) with *Learning*1 (*nbrs*: 50 to 600)

The higher inaccuracy with *Learning*2 means that this method leads to poorer estimation of neighbor proximity through structural feature computation. It can be understood that many of these features can have similar values for different pair(s) of flip-flops even though there is no correlation between them. For the sake of completion, we attempted random filling of the unknown signal states (after traced + restored signal states). This essentially means that we attempt prediction of the unknown signal states (which are denoted as X in Table 6.3) through filling by either 1 or 0 in a random manner. Table 6.8 shows the best (i.e., lowest) values of the parameter, $\frac{I}{U}$ (in %) out of a total of 20 iterations of random filling for seven of the benchmark circuits. For the purpose of comparison, we also report the minimum $\frac{I}{U}$ in % (i.e., the best possible case) for *Learning*1 and *Learning*2 in Table 6.8.

Note that except for *b*17 circuit, the above values of $\frac{I}{U}$ are comparatively higher than that of *Learning*2 technique (Figure 6.5) meaning that the latter (*Learning*2) provides good accuracy in prediction of unknown signal states compared to random filing. However, for all circuits, the values obtained with random filling are higher than $\frac{I}{U}$ (in %) obtained for *Learning*1 (Figure 6.4 and 6.3). This reinforces our claim that *Learning*1 technique is quite useful in prediction of the unknown signal states (and

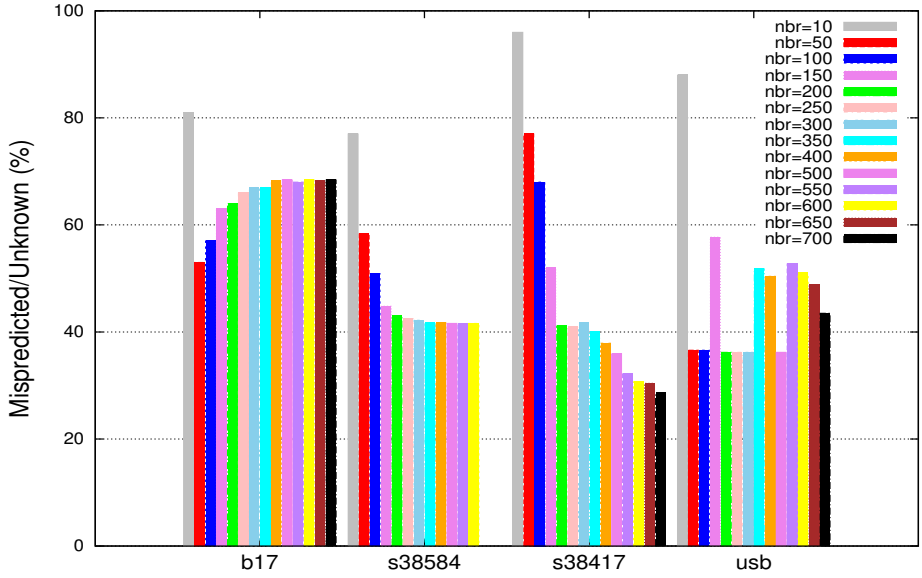

Figure 6.5: Inaccuracy ($\frac{I}{U}$ in %) with *Learning*2 (*nbrs*: 10 to 700)

Table 6.8: Comparative results ($\frac{I}{U}$ in %) with random filling

circuit	min. $\frac{I}{U}$ (random)	min. $\frac{I}{U}$ (*Learning*1)	min. $\frac{I}{U}$ (*Learning*2)
*s*38417	33.49	20.00	28.60
*s*38584	45.68	33.10	41.56
usb	44.21	17.80	36.20
*b*21	21.67	0.99	18.67
*b*17	38.14	7.30	53.10
*s*15850	37.07	18.21	28.61
*s*13207	46.71	16.23	34.27

thereby it can assist in completely expanding the internal visibility).

6.5.3 Defining Error Localization Metric

We perform a topological connection based analysis of the injected error location. The errors injected at granularity level of nets/gates in the netlist can be localized if the error is detected by flip-flops in the immediate vicinity. So, it is meaningful to analyze the efficacy in the following terms:

- The infected flip-flops obtained ($f_{obtained}$) are among the suspect candidates (f_{actual}).
- The proposed method provides lesser but the important suspect candidates. This

can reduce the debug effort significantly.

Note that flip-flops in $f_{obtained}$ are derived from *Error blocks* obtained from Algorithm 16. We define a localization function (g_{loc}) in Table 6.9 to quantify the efficacy of gate-level error localization with the proposed debug methodology. If the error injection and subsequent localization experiment is carried out K times, the function g_{loc} has the maximum value of K. We report g_{loc} values for the two training methods (*Learning*1 and *Learning*2) and different error injection scenarios (*design error*1 and *design error*2 are depicted by *Random-Wire* and *Random-Invert* respectively) in Figure 6.2. The number of *Error blocks* are fixed as top 10% of the total.[14] Because of the incorrect prediction of signal states, we expect some false positives. However, during our experiments, we observed that they are quite less in number.

Table 6.9: Localization metric definition

$fn.$	Value	Condition
g_{loc}	1	$f_{obtained} = f_{actual}$
g_{loc}	1	$f_{obtained} \subset f_{actual}$
g_{loc}	0	$f_{obtained} \not\subset f_{actual}$

6.5.4 Error Localization Results with Complete Visibility

The error injection experiments were iterated for 100 times for both *design error*1 ($e1$) and *design error*2 ($e2$) cases (totaling into 200 iterations for each circuit). Table 6.9 shows the values of the localization function,[15] g_{loc} obtained in the experiments. From the observability expansions, we find out the number of neighbors which provides the best possible accuracy (i.e., lowest value of $\frac{I}{U}$) and then utilize it to obtain the completely expanded visibility and carry out the error localization procedure. Note that because of coarse-grained error localization (i.e., in terms of a group of flip-flops), the accuracy is not explicitly related to the success in error localization. However, the better accuracy

[14] Compared to Algorithm 16, Algorithm 13 of Chapter 5 chooses top ranked certain flip-flops typically, 10, 20 or 30 in number, depending on the size of circuit.

[15] This is identical to Z_{loc} metric introduced in Chapter 5.

helps in giving the least number of suspects (i.e., $f_{obtained} = f_{actual}$). Additionally, we believe that better accuracy would be very useful in localizing bit-flip kind of errors with the proposed methodology.

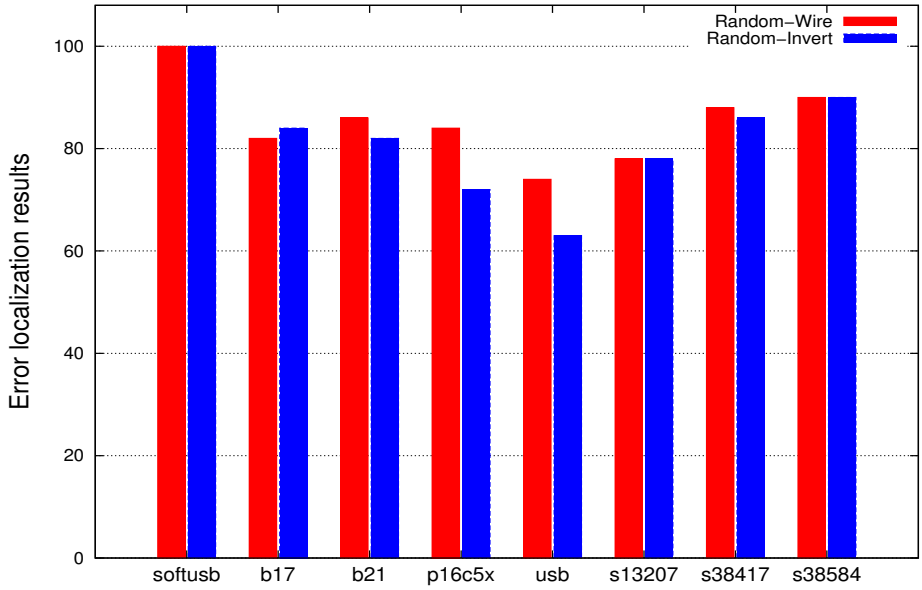

Figure 6.6: Error localization results (out of 100) with M_{nbr} from $Learning1$

It is worth to note that the above results are significantly better compared to the localization results shown in Table 5.7 of Chapter 5. Although the specific netlist errors considered above are different[16] from that of the evaluation in Chapter 5, the nature of errors remain similar. Therefore, the learning methods assist in better error localization. Similar inference can be deduced from the results (Figures 6.10 and 6.11) obtained by the second methodology proposed in this chapter.

Lesser flip-flops in $f_{obtained}$ is expected to assist in quick debug as the intersection of logic-cones of each flip-flop provides the exchanged nets/wires. We obtained maximum value of g_{loc} for one circuit (i.e., $softusb$). There is minor variation in the individual localization results for different error models ($e1$ and $e2$). Similarly, we observed minor differences in g_{loc} values when the number of neighbors or the type of learning method in the built nearest neighbor model, M_{nbr} are changed. On similar lines, we also attempted

[16]In each experiment, we inject random errors so that bias towards error injection is avoided.

bug localization by signal states discovered with the help of *Learning*2 technique. Bug localization results are slightly lower than that of Figure 6.6 with a larger number of unwanted suspects during the debug analysis. The proposed methodology of observability expansion and localization can also be applied in a similar fashion for any large design. The work most closely related to ours is that of [183] where the authors have identified clusters (of flip-flops) in the design and based on the logical relation between the cluster elements and the inputs to the cluster, signal states of flip-flops can be known. This also requires the knowledge of the initial values of the elements of the cluster which is available through tracing. They achieve significant improvement in state restoration ratio; however, our goal is to obtain all the unknown signal states which can lead to a completely enhanced internal visibility of the design. In the next section, we propose a different approach which does not require complete expansion of internal visibility. This technique utilizes a model building approach [176] to obtain a list of suspects with ranking of their suitability of being the actual culprit.

6.6 Error localization with Design Response Model Building Approach

We develop a methodology which assists in localizing the infected flip-flop(s), which in turn hint towards buggy gate(s) in a erroneous design. The proposed methodology includes two phases: *training* and *testing*. The *training* phase includes building a model[17] which consists of a relationship between smaller region of circuit (in which bugs are injected synthetically) and the corresponding circuit responses which, in turn are to be utilized for identifying the buggy gates given an erroneous netlist. During the testing phase, given a certain circuit response (which essentially means the signal states of all

[17]In the conference version[184] of this work, the methodology was named as regression analysis. The notion of regression is not perfectly suitable to the proposed technique as output variables take continuous values in the regression analysis. Contrary to this, in the proposed technique, we predict a discrete (and unordered) class output. Hence, this usage of model building can be more aptly referred to as a classification problem.

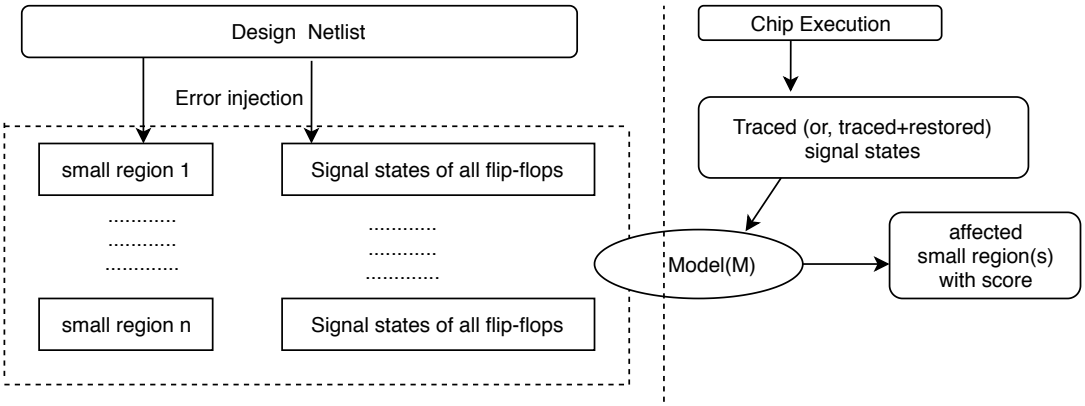

Figure 6.7: Design response model (M) building methodology

flip-flops), the proposed methodology classifies it as one of the labels (which are the smaller regions in the netlist of the design) present in the design response model. In this manner, the localization upto a small region (each consisting of only one flip-flop and few gates) can be achieved.

The design response model (M) is developed through a large number of iterations, where in each iteration one error is injected into the design description and the complete error signature (state of all the flip-flops of the design for a fixed number of cycles) are used in the *training* phase. During the *testing* phase, for localizing the error location in a buggy netlist based on the partial bug signature (state of certain number of flip-flops of the design) obtained through on-chip trace buffers. The overall flow of the methodology is shown in Figure 6.7.[18]

While the localization ability of the proposed model depends on the training of the model, error localization to a small portion of the design depends on the post-silicon observability available through the trace buffers. The proposed methodology makes use of the advantage of indirect expansion of visibility through state restoration technique [1, 9, 102]. Since restoration of untraced signal states varies with the trace signal selection methodology, variation in localization results is expected with each technique up to some

[18]In practice, building this model is quite difficult if the correct netlist (i.e., which corresponds to the reference/golden implementation may not be available.) However, even if a correct netlist is available, the process of localization tends to become tedious. The proposed approach aims to ease the process of localization to a smaller portion of the design netlist in those debug scenarios.

extent. Table 6.10 introduces the terminology for various parameters to be used.

Table 6.10: Notations and their meaning

Term	Meaning
M	training design response model
TBw	width of trace buffer
D_{tb}	depth of trace buffer
ffZ_k	any *flip-flop zone* of the circuit
K	no. of iterations of training
Sle_i	signature used for training
Cy_{iter}	no. of cycles in each iteration of training
FF_j	error signature of j^{th} flip-flop for training
Ste	signature used for testing
$rank_j$	*rank* of j^{th} flip-flop
ff_j	error signature of j^{th} flip-flop for testing

6.6.1 Finding Smaller Zones in Circuit

The first step in the *training* phase is the introduction of bug/error in smaller portion of the netlist for the purpose of training. The circuit description is divided into a fixed partitions, termed as flip-flop zones (ffZ) meaning a collection of gates connected to one particular flip-flop. Under this step, a particular ffZ can have gates connected in certain levels (generally one to six). If the circuit has F_{tot} flip-flops, there can be a maximum of F_{tot} flip-flop zones. The idea behind ffZ finding approach is that once we are able to locate any ffZ, the individual erroneous gate(s) can be identified with the help of that information. Figure 6.8 illustrates ffZ identified from a design netlist. The portion outlined (with solid line) comprising of one flip-flop and six gates is ffZ corresponding to *FF1*. The other region (outlined with dashed line) depicts ffZ corresponding to second flip-flop (FF2).

The error propagation to a flip-flop depends on the conditional probabilities of signal transitions at different inputs of gates following on the path leading to it. Therefore, for developing M, we take one ffZ and inject error into it and then obtain a bug(error) signature. During the next iteration, we select (in random fashion) another ffZ and

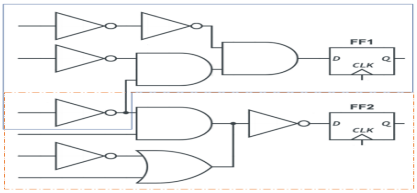

Figure 6.8: Circuit for illustrating ffZ identification

repeat the same procedure. Note that it is possible that some gate(s) may be common between two flip-flop zones of the design. During testing phase of the proposed methodology, error localization up to either one ffZ or a small list of ffZ is achieved with the help of statistical analysis.

6.6.2 Error Injection in Smaller Zones

At each iteration of training, we inject error by the method of random gate replacement, where one gate is randomly replaced by another [55, 163]. This model resembles manifestations of many kinds of mutations at RTL after the synthesis step. For some cases, this model can serve as a representative of electrical errors also. For example, the *OR* gate from which a flip-flop in design derives its input gets replaced by *XOR* gate and in some cycle the content of this flip-flop is "1". When both inputs to the new gate become "1", its outputs become "0" causing a bit-flip at the concerned flip-flop.

Note that any other error model can also be utilized for the purpose of generating buggy signatures to develop the classification model (M). We introduce random error to remove any bias towards any specific region in the circuit during *training phase*. Figure 6.9 shows an example ffZ where one *AND* gate (shown in the left side) is replaced by one *OR* gate (shown in red color in the right side).

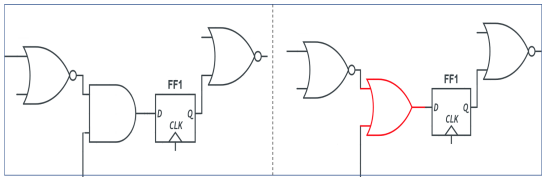

Figure 6.9: Circuit portion for illustrating error injection

It is likely that during any iteration of the *training* phase, error is not injected due

to nature of the selected ffZ in this iteration. For instance - if ffZ contains only "NOT" gates, error injection can not proceed. Another likely scenario is that during some iterations (out of total K) of training, same ffZ is selected for error injection since this selection is done in random manner. However if during $(i+1)^{th}$ iteration, selected ffZ is same as that of i^{th} iteration, the error injected is likely to be different giving a different error signature. The error signature generated during each iteration of this phase is related with the infected flip-flop zone(ffZ) and over a large number of iterations, a classification model is developed between the two parameters. Thus, error signature, Sle_i (in each iteration) leads to a training vector, Trv_i in the model M which can be related with the corresponding ffZ.

6.6.3 Building the Classification Model

For analyzing the relationship between the injected error (in ffZ) and respective FF_i (error signature), the steps described above is iterated K times (depending on number of flip-flops of the netlist and is presented here as Algorithm 17). The *error signature* consists of states of all flips of the design for Cy_{iter} cycles. Two possible methods to compute error signature (FF_i) are given as follows:

- *T1*: Recording only the erroneous states of all the flip-flops, assuming no golden (error-free) signature.
- *T2*: Assuming that a golden signature is available, computing XOR values of the golden and erroneous signature.

Corresponding to these two training methods, difference is expected between error localization ability of the resultant model. The steps of developing the classification model (M) are summarized as Algorithm 17.

Each training vector (Trv_i) comprises of a matrix of F_{tot} x Cy_{iter} state bits (flip-flop responses). For K iterations of training, M contains K labels where label ffZ_1 correspond to Trv_1, label ffZ_2 correspond to Trv_2 and so on.

Algorithm 17: *buildModel*

Input: $Netlist$,K,$choice$,F_{tot}
Output: M

```
1  M ← ∅;
2  goldensignature ← state of F_tot flip-flops for Cy_iter cycles;
3  for i=1 to K do
4      find flip-flop zones from Netlist;
5      S(ffZ) ← flip-flop zones from Netlist;
6      ffZ_k ← randomly selected one ffZ from S(ffZ);
7      inject random error in ffZ_k;
8      for each of F_tot flip-flops for Cy_iter cycles do
9          FF_jp ← state of j^th flip-flop in p^th cycle;
10         errorsignature ← state of all FF_jp's;
11         if choice = T1 then
12             I ← {ffZ_k,errorsignature};
13         end
14         if choice = T2 then
15             XORsignature ← XOR(goldensignature,errorsignature);
16             I ← {ffZ_k,XORsignature};
17         end
18     end
19     M ← I ∪ M;
20     S(ffZ)← ∅, I ← ∅;
21 end
```

6.6.4 Evaluating the Classification Model

6.6.4.1 Defining *Objective Function*

During testing phase, signature (Ste) generated from trace buffers is the testing vector, Ttv. From this testing vector, an *Objective function*($Objfn$) is computed for each label (ffZ) which denotes the deviation of each flip-flop , ff_j (of Ttv) from M. For $T1$ training method, $Objfn$ is calculated as per Equation 6.1, where FF_j denotes corresponding flip-flops in the training vector(s). For each ffZ, *Objective function* is depicted by $Objfn_{T1}(ffZ)$ when $T1$ training method is adopted.

For the second training method, *Objective function* is given by Equation 6.2, where the only difference is that FF_j and ff_j are replaced by XOR values of their golden and erroneous signature values. In both the equations, p denotes the cycle of j^{th} flip-flop.

This factor measures distance as the cycle-by-cycle computation($Objfn$) of all flip-flops (beginning from the first flip-flop) proceeds.

$$Objfn_{T1}(ffZ) = \sum_{j=1}^{j=n} \sum_{p=1}^{p=Cy_{iter}} (FF_{jp} - ff_{jp})^2 \tag{6.1}$$

$$Objfn_{T2}(ffZ) = \sum_{j=1}^{j=n} \sum_{p=1}^{p=Cy_{iter}} (FF'_{jp} - ff'_{jp})^2 \tag{6.2}$$

where, we have

- FF'_j given by XOR(golden FF_j, erroneous FF_j)
- ff'_j given by XOR(golden ff_j, erroneous ff_j)

For an effective localization, $Objfn$ must be minimized justifying that localization happens based on the model (M) if Ttv is similar to any one of the Trv_i. Thus, $Objfn$ captures the dissimilarity between all Trv and Ttv for each ffZ. In above Equation, FF'_j (training vector components) and ff'_j (testing vector components) are compared only in those clock cycles where there is a difference between the two.

It is worth to note that the model M contains K labels corresponding to K iterations, where each label correspond to a ffZ (which consists of one flip-flop with some gates). From the perspective of a classification model, the label which corresponds to best error localization in M is represented by Equation 6.3 where a_1, a_2 ..... a_K are constants and ffZ_1, ffZ_2 ..... ffZ_K are the corresponding labels (ffZ).

$$ffZ_k = a_1 ffZ_1 + ..a_i ffZ_i + .. + a_K ffZ_K \tag{6.3}$$

The actual erroneous ffZ_k is any one ffZ_i (the constant a_i corresponding to that is 1, while all others are 0). This ffZ_k corresponds to the minimum value of either $Objfn_{T1}$ or $Objfn_{T2}$ as per the respective training method ($T1$ or $T2$).

6.6.4.2 Analysis of Profitability of Training Methods

For $T1$ method, flip-flop state values in testing vector are to be compared with those in training vectors of M, while in $T2$ only XOR values of golden and erroneous in testing vector are compared with XOR values in training vector. Thus, in $T2$ we have a refined data ($Objfn$ values) for the purpose of comparison. This is because we have comparison points ($Objfn$ values) in $T2$ case only when bit values of XOR between golden and erroneous signature are "1".

The computation of $Objfn$ (in Equations 6.1 and 6.2) proceeds differently with the post-silicon observability available through on-chip trace buffers. This is because the amount of known bits (0 or 1) and don't care bits (X) varies whether restoration technique is applied (or not). After utilizing restoration, $ObservationWindow(ObsW)$ which denotes the number of flip-flop states visible/observable changes. To account for this variation, a factor ($1/ObsW$) may be introduced in $Objfn$ computation. Note that for restoration, we can estimate $ObsW$ as *restored and traced states* (rts), calculated by sum of *traced states* ($= TBw{*}D_{tb}$) and *restored states* whereas for non-restoration, we can estimate $ObsW$ as *traced states* (ts) only. It is obvious that $ts < rts$ whenever we achieve non-zero restoration using the traced signal states.

For the same injected error (of random nature, either wire exchange or gate replacement), if we do not apply restoration, number of state bits in the test vector error signature are less. However, in case of restoration (obtained from trace signals selected by any restorability maximization technique [1, 9, 102]) number of unknown signal bits are reduced. Because of this, we obtain lesser values of $Objfn$ with restorability technique as compared to non-restoration. However, note that the classification of ffZ and the suspect identification (by ranking based on $Objfn$ values) is to be done among the probable candidates separately in the case of restoration or non-restoration.

6.7 Experimental Formulation and Results of Localization with Model-based Classification

6.7.1 Experimental Setup

We have chosen seven benchmark circuits (ISCAS-89, ITC-99 and OpenCore) for testing the localization efficiency of the proposed analysis model. For building the classification model, we selected value of K (which can be variable for each benchmark circuit) between 1000 to 2000.[19] Note that random input test vectors have been used during the *testing* phase. For the *training* phase also, random inputs were applied. So, the input vectors for both phases (of training and testing) are different. For experiments, TBw is chosen as 32 and D_{tb} as 1024 for trace buffers. Thus, for testing an erroneous design netlist, we have the *error signature* (comprising of these 32 flip-flops for 1024 cycles (Cy)). We inject error into the design netlist by 2 methods [55, 163] separately (during each experiment): random gate replacement and wire exchange.

6.7.2 Formulation for Identifying *False Positives*

During *testing* phase, based on the *error signature*, we obtain the suspect ffZ (i.e., corresponding flip-flops in ffZ). One important issue is to verify the suspect ffZ obtained from M. As is possible with any statistical process, there are chances of *false positives* i.e., cases when some ffZ are not affected but during testing, the model M gives these *flip-flop zones* as suspects(S). To verify that no *false positives* are provided during *testing phase*, we perform a topological connection based analysis of the injected error location($error_i$) and identify the flip-flops (i.e., few ffZ) that can be infected by $error_i$. Through this topological analysis, we obtain a parameter $n(S_{con})$, whereas from the classification model M, we obtain another parameter $n(S)$. Let $n(S_{con})$ represent number of flip-flops (or, ffZ regions) directly connected to the location of $error_i$ (in

[19] We observed that higher the value of K, better is the error localization. For circuits of larger sizes, value of K should be set up in between 2000-5000.

the forward logical-cone) and $n(S)$ represent number of suspects (ffZ) obtained during *testing*. For efficient localization, we want $n(S)$ to have the least possible value (i.e., 1). However during testing, when we have multiple suspect ffZ, these probable ffZ are ordered by $rank_j$ as per their respective $Objfn$ values. The model, M must abide by the following two conditions while being evaluated on any testing vector for error localization:

- $n(S_{con}) \geq n(S)$
- $n(S) \subseteq n(S_{con})$, for occurrence of no *false positives*

During the *testing* phase, with M, we can classify every ffZ as buggy region/non-buggy region with a score (based on $Objfn$ values). Taking top scores[20] out of them, we can obtain the error localization results. It is worth to note that in every iteration of *testing* phase, a separate error is introduced for the purpose of testing. The error injection is purely random in nature. We choose $N_{testing}$ as 100 in our experiments. During our experiments we observed $n(S)$ as 1 in many iterations for all the benchmark circuits, signifying that the proposed methodology is able to identify the actual suspect ffZ. It is worth to note that the values of $n(S_{con})$ vary from a range of 1X-6X to that of $n(S)$. For all the testing experiments, we obtained that $n(S) \subseteq n(S_{con})$ depicting that the proposed analysis avoids the occurrence of *false positives*.

6.7.3 Results on Error Localization with Classification Model

We performed 100 iterations (denoted by $N_{testing} = 100$) of wire exchange error and the results[21][22] with restoration and non-restoration is shown in Figure 6.10 for $T2$ method-

[20] This can vary as 10 or 20 similar to Algorithm 13 of Chapter 5.

[21] Alternatively, the metric, g_{loc} mentioned earlier in this chapter can also be used here.

[22] In the conference publication of this work [184], we defined the following parameter to measure the quality of error localization. Lower the value of *elmetric*, better is the error localization.

$$elmetric = \frac{\sum_{j=1}^{j=N_{testing}} n(S)}{N_{testing}} \tag{6.4}$$

However, this metric fails to differentiate between the number of iterations in which the error localization succeeds and iterations in which localization fails. So, in many cases, this metric can be misleading as those iterations are not accounted in which $n(S) = 0$.

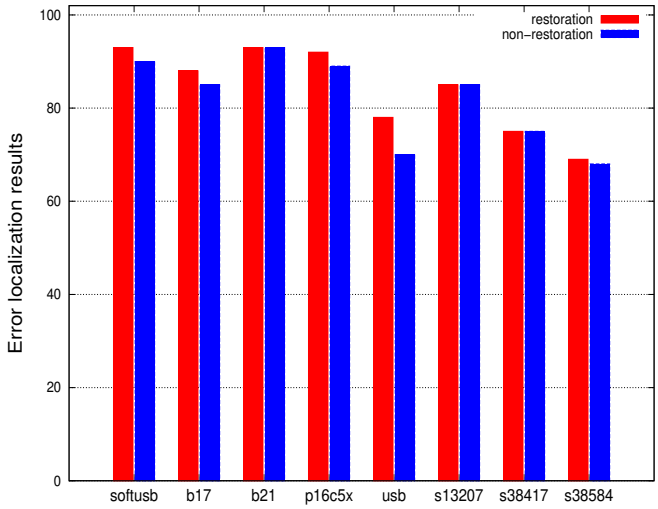

Figure 6.10: Error localization success (out of 100) for wire exchange error

ology. We consider the metric Z_{loc} defined in Section 5.9.3.2 of Chapter 5 to obtain the error localization results.

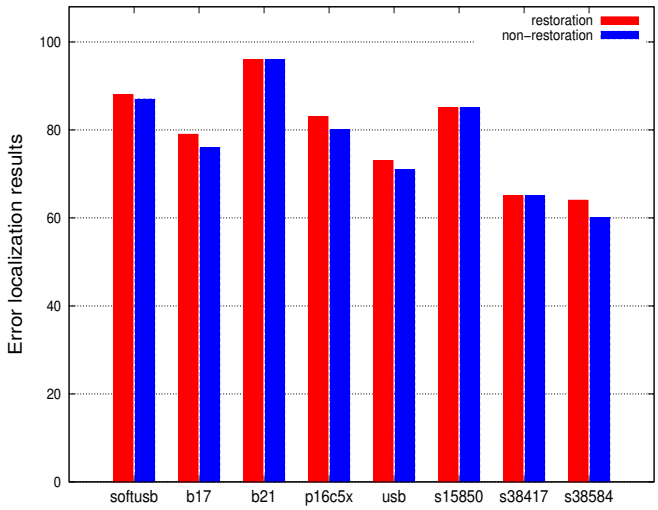

Figure 6.11: Error localization success (out of 100) for gate replacement error

We also performed experiments for gate replacement error with the design response model, M for $T2$ methodology. The Z_{loc} values are shown in Figure 6.11. These values show similar trend as observed in Figure 6.10. For few circuits, the localization values are lower than that observed for wire exchange error. An increase in ease of localization is obtained by using state restoration technique as we observe lesser $n(S)$ values with this technique. In 10-20% of the iterations, we obtained $n(S)$ as only one. There is minor

variation in Z_{loc} values when the trace signal selection technique is varied.

As expected, when M is build with $T1$ methodology, the error localization is inferior (i.e., Z_{loc} values are lower) than that of Figures 6.10 and 6.11. However, the major takeaway from these results is that model building approach succeeds in gate-level localization (in relative comparison of localizing with only traced and restored signal states as was done in Chapter 5). Additionally, we observed in our experiments that this approach is less sensitive to the trace signal selection technique.

6.8 Conclusion

This chapter proposed a methodology of gate-level error localization based on the observability expansion of the limited debug data available through on-chip trace buffers. By discovering nearest neighbors, unknown signal states can be discovered with the help of decision making based on the traced and restored signal states. The proposed methodology reconstructs signal states with reasonable accuracy in comparison to the actual signal states. Two different methodologies were proposed for finding the neighbors for any particular signal of the design. The first methodology involves neighbor finding by distance calculations from simulation values. The second methodology involves netlist structure-based analysis for computing a set of features which decide the neighbors of any particular signal (flip-flop). With the expanded visibility, design error localization improves significantly. Using learning analysis, a model of design responses (signal states of all flip-flops) was developed by training for a large number of iterations. This trained model was used to test an erroneous design to obtain probable suspects with a ranking scheme. With the model building approach, better gate-level design error localization can be achieved. We believe that standard outlier detection algorithms from machine learning field need to be employed to achieve finer error localization.

— * — * —

Chapter 7

Debug Architectures with On-chip Compression

7.1 Introduction

Due to the limited tracing options available and irreproducible behavior of failures, the difficulty of the post-silicon validation problem increases and is a severe hindrance in achieving time-to-market (TTM) targets [33, 185, 185–189]. Even if the errors are of reproducible nature, localizing them takes a large amount of time because of the restricted internal visibility. Typically, silicon debug needs to be performed via a mix of on-chip debug data collection and off-line processing methodologies. To facilitate this debug process, some amount of on-chip storage (in the form of trace buffers) is present [185, 190]. However, because of different overheads and subtle performance issues, the amount of on-chip debug data storage is limited which must be judiciously utilized. This can be done via on-chip data compression mechanisms and then subsequent off-chip analysis [123, 124, 160]. The type of compression employed has implications on the granularity of error localization and the overall debug time[122]. Under this scenario, we need to deploy lossless compression so as to avoid the aliasing of debug data. The process of post-silicon validation and debug is typically carried out in multiple sessions[160, 187, 190–192] for repeatable error scenarios. The number of sessions and the time spent in each session

are indicators of the total debug time. Techniques proposed by Yang et al. [187] and others[188, 191, 193] involve multi-session based debug approaches to efficiently utilize the on-chip storage and minimize the subsequent debug time. The motive is to utilize the on-chip storage only for collecting the useful portion from the debug data and discarding the remaining. In this chapter, we propose an improved version of two-session data capture methodology that employs a modified debug infrastructure aimed at the efficient usage of the on-chip storage. We also propose a methodology for the segregation of debug data to achieve finer temporal visibility expansion leading to quick error localization. The proposed technique achieves a significant improvement upon the visibility window expansion results reported in [187] and the proposed debug architecture is complementary to that of [185, 188, 189, 191].

In this chapter, we also present a reconfigurability-directed solution to counter the limited observability during post-silicon debug. The approach is to reuse the existing DFT infrastructure in a similar direction as in the case of serial scan architecture [194]. Our approach uses progressive random access scan architecture [14], a variant of random access scan. The key contributions of our approach of reusing random access scan-based architecture are as: (a) a debug methodology is proposed which utilizes progressive random access scan (PRAS), and (b) a mechanism of reconfigurable observability enhancement without any steering logic so as to enable debugging in an effective manner is proposed.

The rest of the chapter is organized as follows. Section 7.2 discusses the preliminaries involved in the two session-based debug approach. Section 7.3 presents our proposed debug architecture and explains its operation in detail. Section 7.4 elaborates the proposed methodology through an illustration. The methodology for generation of tag bits and fine-grained spatial data collection is explained in this section. Section 7.5 elaborately analyses the debug performance of the proposed architecture on various circuits for different error types. Section 7.6 explains the proposed PRAS-based scheme with details of the architecture and the methodology. Experimental result is discussed in Section 7.7. Section 7.8 presents an elaborate discussion on the merits of the proposed session-based

debug architecture and puts it in contrast to other silicon debug techniques reported in the recent literature. Finally, the chapter is concluded in Section 7.9.

7.2 Background behind Session-based Silicon Debug

As stated previously, the amount of debug data to be offloaded needs to be restricted owing to the limited off-chip dumping speed. Additionally, due to area overhead, we can only accommodate limited on-chip storage (commonly referred to as trace buffers) [160]. Further, because of faster execution speed of the circuit (or, core)-under-debug (CUD), the debug data in trace buffers can be over-written unless a successive run-and-halt mechanism is followed. One way of mitigating this is on-chip debug data compression so that the amount of required storage is minimized [123, 124, 127]. On-chip compression allows the debug procedure to be performed in more than one session [124, 187, 188, 191]. The overall flow of the proposed two session debug methodology is shown in Figure 7.1. The off-chip steps are shown in the left hand side and the on-chip steps are outlined on right hand side of Figure 7.1 (details of each step are explained in the later sections of this chapter).

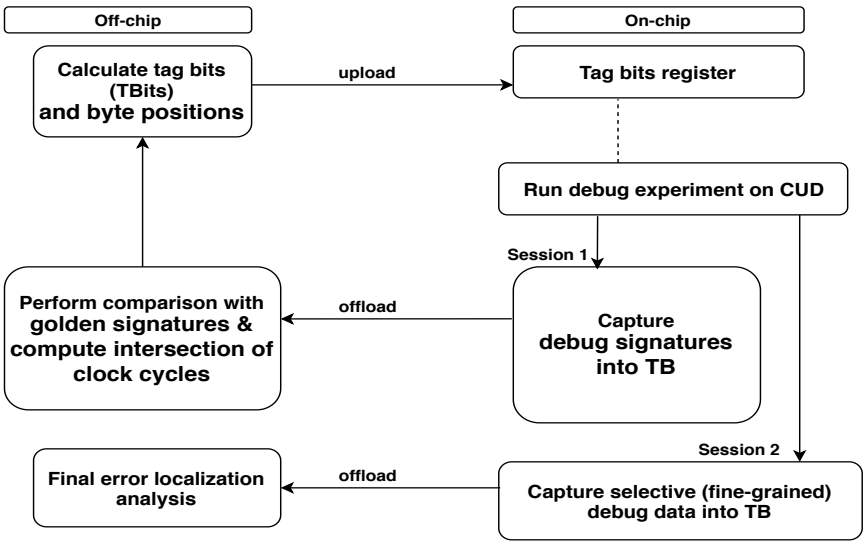

Figure 7.1: Overall flow of 2 session-based debug

To explain the multi-session debug problem in more detail, we adopt the terminology in Table 7.1.

Table 7.1: Terms and their meaning

Term	Meaning
CUD	Circuit (or, core)-under-debug
L	Number of clock cycles of CUD execution
W	Data-word (signature) width
TBw	Trace buffer (TB) width
D_{tb}	Trace buffer (TB) depth
S_i	i^{th} debug session
$Bits$	CUD data to be stored in TB
$TBits$	Tag bits (for clock cycles) to be stored
Cy	Set of debug data cycles to be stored in TB
$\|Cy\|$	Number of debug data cycles to be stored in TB
ER	Error rate in CUD execution

Let's consider that CUD data[1] is to be observed for L clock cycles, the total storage required is L data-words. With compression by a MISR of length $W1$, this reduces to $\lfloor L/W1 \rfloor$ signal dumps (or, signatures) which should be stored in trace buffer (TB) and then offloaded with a fixed JTAG frequency (f_{JTAG}). Clearly, the dimensions (TBw*D_{tb}) of TB determine how much of L can be accommodated in both the S_i sessions (i.e, S_1 and S_2). Further, all of the $\lfloor L/W1 \rfloor$ signal dumps may not necessarily differ from the golden data (i.e., each one of them may not capture the error). Under the multi-session debug scheme, in first session, the compressed MISR signatures are offloaded for analysis. By comparison with the respective golden signatures, the erroneous clock-cycles can be computed. These clock cycles are then utilized for the identification of the respective MISR signal dumps which need to be actually stored in TB. Therefore, three requirements emerge for effective multi-session debug:

- the number of identified suspect clock cycles ($|Cy|$), at end of first session should be as small as possible
- the amount of debug data ($Bits$) to be stored in TB should be least, allowing TB

[1]We assume that internal signals to be observed are given. As is clear from previous chapters, there is a large body of work in the recent literature which addresses the signal selection problem. Specifically, the methodology of multi-session silicon debug discussed in this chapter can be applied to any set of trace signals. These signals can be obtained either through methods introduced in Chapter 5 or the set of signals provided by the design engineers based on their expertise.

to be effectively used

- the amount of tag bits ($Bits$) to be stored for facilitating selective data capture should be also minimum

If the first requirement is fulfilled, then the temporal observation window ($L1$) can be enlarged leading to a reduction in the total number of debug iterations. Henceforth, if earlier $\lfloor L/L1 \rfloor$ iterations were to be done, with the effective usage of on-chip storage, this number becomes significantly lesser than $\lfloor L/L1 \rfloor$ because of temporal expansion in the individual iteration of sessions in the debug experiment. The maximum time taken to offload the contents from TB is given by:

$$T_{offload} = TBw * D_{tb} * \frac{1}{f_{JTAG}} \tag{7.1}$$

Since we can not optimize $T_{offload}$ much because of its dependence on f_{JTAG}, we can attempt reducing the number of times offloading needs to be done through better utilization of TB. In an uncompressed debugging scheme, the observation window is limited to D_{tb} (which may be equal to a fixed CUD execution length, L). With the compression enabled debugging, the achievable temporal expansion can be given by

$$TExpansion = \frac{L}{|Cy|} \tag{7.2}$$

Note that if $|Cy| = L$, $TExpansion$ is one which is its minimum value. The lower the value of $|Cy|$, the higher $TExpansion$ can be obtained. Because of this higher temporal expansion, the overall debug time can be reduced significantly. The proposed debug scheme assumes that the correct CUD response signatures (i.e., golden data) can be computed off-chip by a sufficiently verified behavioral reference model. This assumption has also been considered in similar methodologies [123, 185, 187, 188, 191] in the literature. Nevertheless, obtaining a fully verified reference model is itself an equally challenging problem.

7.3 Proposed Multi-session Debug Architecture

7.3.1 Brief Overview

The proposed debug architecture is shown in Figure 7.2. The same on-chip storage (i.e., trace buffer) is utilized for storing compressed debug data signatures (which are to be offloaded for Cy identification) in the first session and storing tag bits along with the debug data corresponding to suspect clock cycles (Cy) in the second session (S_2).

The total time (T_{uo}) spent in uploading and offloading of different contents in a debug experiment can be given as follows:

$$T_{uo} = S1.T_{offload} + S2.T_{upload} + S2.T_{offload} \tag{7.3}$$

We target reducing the amount of debug information needed to be offloaded and uploaded so that the individual components in the right hand side of above equation are reduced. $S1.T_{offload}$ denotes the offloading of compressed signatures, $S2.T_{upload}$ depicts the time spent in uploading of tag bits for facilitating the selective data capture in S_2. $S2.T_{offload}$ represents the offloading of fine-grained debug data for the final error localization analysis.

7.3.2 Debug Architecture Operation

The techniques in [187, 193] employ a combination of MISR and cycling register (which is basically an arrangement of XOR gates and is abbreviated as CR) for the purpose of debug data compression. These methodologies compute suspect erroneous clock cycles (Cy) from the intersection of MISR and CR failing signatures ($Sign_{failing}$) as below:

$$Cy \ = MISR\ Sign_{failing} \ \cap \ CR\ Sign_{failing} \tag{7.4}$$

It is obvious from above that as the amount of intersection of clock cycles (corresponding to failing signatures) increase, Cy size decreases and we move closer to the actual

erroneous cycles (Cy_e). For exact error localization, we desire Cy to be equal to Cy_e.

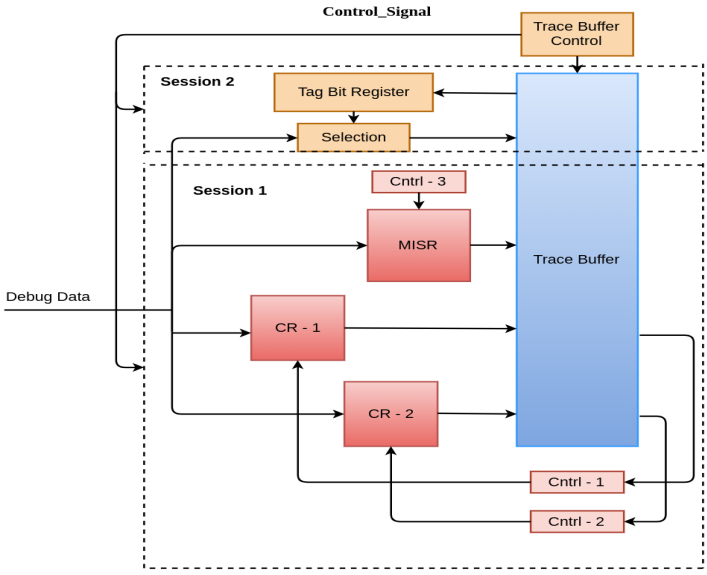

Figure 7.2: Proposed multi-session debug architecture

However, Cy may be significantly larger than Cy_e. For a successful debug experiment, we require that $Cy_e \subseteq Cy$. The difference (Cy-Cy_e) serves as the overhead for which data needs to be stored in TB in addition to data corresponding to Cy_e cycles. One of the methods to obtain higher data compaction is to employ a larger number of cycling registers. However, apart from additional area overhead, chances of aliasing of cycling register signatures also increase. Therefore, we fix the number of cycling registers as two and utilize other techniques to minimize the difference (Cy-Cy_e).

In the first debug session (shown in Figures 7.2 and 7.3), signatures of the CUD execution are computed by a MISR and two cycling registers ($CR1$ and $CR2$) and then these are stored into the trace buffer. For the MISR, each signature compacts the CUD execution data depth by k. A cycling register of m_1 length compacts the CUD data at every m_1^{th} data-word.[2] In total, m_1 signatures are produced by the first cycling register ($CR1$) while m_2 signatures are generated by the second CR (i.e., $CR2$). Under the proposed methodology, Cy is calculated by taking the intersection of the debug data

[2]A data-word essentially means a CUD execution data of length W bits.

signatures as follows:

$$Cy = (Cy.MS \cap Cy.CR1) \cap (Cy.MS \cap Cy.CR2) \tag{7.5}$$

In the above, $Cy.MS$ denotes the clock cycles for which MISR signatures mismatch with the corresponding golden signatures. Similarly, $Cy.CR1$ and $Cy.CR2$ denote the clock cycles for which their debug signatures are different from the respective golden data. Intersection of these clock cycles provides us the suspect erroneous cycles (Cy) for which data must be stored in TB. The detailed operation of the debug architecture[3] is shown in Figure 7.3. After the intersection of the mismatched signatures of the first

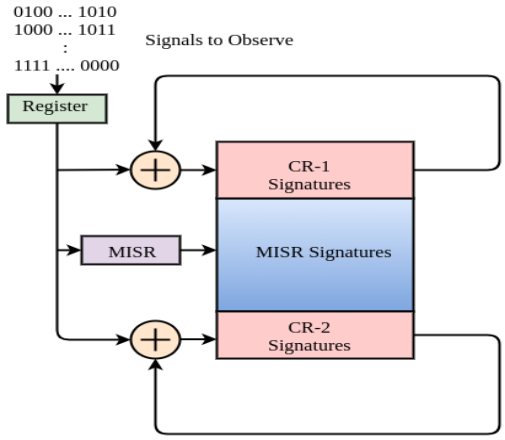

Figure 7.3: Session-1 operation in multi-session debug

session is calculated, tag bits are generated for facilitating the capture of debug data (in the form of signatures) corresponding to suspect clock cycles. This data is stored (i.e., uploaded onto the chip via JTAG mechanism) on the trace buffers before starting the second session. The authors in [187] have proposed the usage of a parity scheme in their first session to estimate error rate (ER) in the debug data. This facilitates the computation of the temporal observation window length for the subsequent second and third sessions. In the proposed architecture, the compressed debug data is loaded into the on-chip storage (TB) in the first session for the off-line comparison with the golden

[3]The register in Fig. 7.2 is merely for facilitating transmission of the same debug data to the MISR and the two CRs for data compression.

signatures. The final debug data capturing occurs in the second session. One of the major differences between [187] and the proposed scheme is that the former requires three sessions while the latter requires only two sessions. The argument behind this is that in the proposed scheme, the amount of debug data compression is relatively higher which allows a serialized observation window in S_2. However, the number of iterations of S_1 may be higher as the observation window is to be determined based on the dimensions of the trace buffer.[4] Note that if the golden debug data can be stored on-chip as suggested in [188, 191], the comparison and debug data capturing can be done simultaneously in a single session itself.

The proposed tag bit generation method is different from the techniques described in [187, 193] and is elaborately described in Section 7.4.2. The proposed approach gives assurance that minimum tag bits are required which lead to storage of debug data corresponding to lesser number of additional clock cycles (i.e., the suspect clock cycles which are not erroneous and not present in Cy set). Storage of debug data corresponding to these extra clock cycles acts as overhead and the number of these additional clock cycles (for which unnecessary data gets dumped in TB) must be minimized.

In the first session, the control signals enable the 2-D (i.e., both temporal and spatial) compaction by allowing debug data compression through MISR and Cycling registers and generated signatures are stored in the trace buffer as previously stated. Generally, MISR signatures require more space, so trace buffer size is accordingly divided. If we choose cycling register ($CR1$, $CR2$) sizes as prime numbers, then there is an appreciable chance of getting lesser amount of the intersection with MISR signatures leading to a reduced number of suspect clock cycles (Cy). Essentially, $CR1$ and $CR2$ consist of these individual registers and the respective XOR gates as shown in Fig. 7.2. In the first session, three control blocks (shown as Cntrl-1, Cntrl-2 and Cntrl-3 in Fig. 7.2) are active. The first control block (Cntrl-1) is associated with MISR and it ensures the

[4]The combined time for parity-based session and then intersection finding session in [187] would be comparable to the time taken for multiple iterations of S_1 (because of serialization-based partition of the total observation window) in the proposed methodology. However, with the removal of parity-based session, we get rid of the requirement of off-line dumping at the end of it.

resetting of MISR after the MISR signatures are stored in the trace buffer. The second and third control blocks are associated with the cycling registers. Once the trace buffer is full, signatures will be offloaded for further analysis. The scheme for the second session is shown in Fig. 7.4 (where tag bits are stored in the location shown as Tag Bits part of TB). Typically, the selection hardware consists of a tag bit register with a down counter

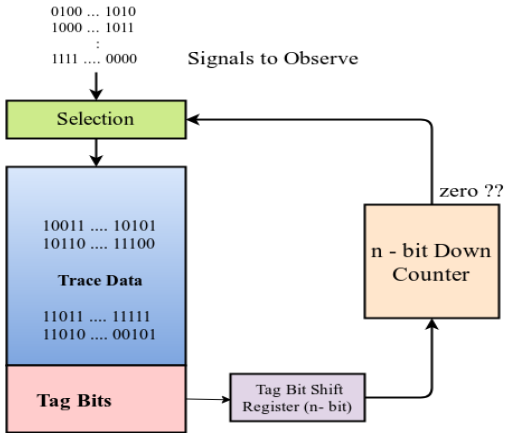

Figure 7.4: Session-2 operation in multi-session debug

aimed at capturing the data in the suspect clock cycles. The down counter is loaded with the tag bit register value (i.e., suspect clock cycle number) and starts counting in the downward direction. When zero (in the down counter) is reached, one flag is generated, and the debug data gets stored on the trace buffer, and a new value from tag bit register will be loaded for the down counter. As the tag data is read out of the trace buffer, it can be overwritten in the trace buffer by the captured data. During design of the debug infrastructure, enough slack has to be provided such that the captured data never tries to overwrite any unread tag bit data.

7.3.3 Design Choices in Proposed Architecture

As mentioned in previous section, there are three important parameters which decide the amount of on-chip compression: k, m_1, m_2. To avoid aliasing, the length of MISR should be kept high and preferably matching with W. If k is kept high, we can obtain spatial compression along with the temporal compression (i.e., in a number of clock cycles) provided by MISR. However, to avert the subsequent complex decoding of the

signatures from the CUD execution data, we prefer to avoid spatial compression of data, thereby fixing k as W. On-chip cycling registers assist us in achieving significant spatial compression provided their lengths are chosen reasonably. From Equation 7.5, it can be easily deduced that larger the length of cycling register (i.e., m_1 or m_2 value), the number of intersections would be less because of higher compaction by it. However, since the number of signatures generated by cycling register is dependent on its length, the trace buffer depth also needs to be considered. Note that the amount of trace buffer slots (of a fixed width) required by a cycling register is equal to the number of signatures generated by it.

7.4 Details of Two Session-based Debug

7.4.1 Suspect Clock Cycle Determination in 1^{st} Session

Two-dimensional (2-D) compaction is performed on the incoming CUD execution data in the first session with the help of MISR and CR. As stated before, k locations of the trace buffer are allocated to store the MISR signatures, thereby MISR signatures are stored at every k cycles. Similar to the previous notation, m_1 and m_2 locations of the trace buffer are allocated for cycling register. The cycling register signatures are generated by XOR operation of the incoming data-word (i.e., debug data) with the data already present (at one of the places in the trace buffer, marked as cycling register locations in Fig. 7.3) pointed to by mod-m_1 and mod-m_2 address counters respectively. In this manner, the cycling registers generate signatures which consist of the XOR of every m_1^{th} and m_2^{th} debug data-word respectively. We consider a scenario for 35 clock cycles ($=L$) of debug data words with and $k = 5$, $m_1 = 7$ and $m_2 = 5$. Under this scenario, $CR1$ (or, CR-1) compacts every 7^{th} data-word and $CR2$ (or, CR-2) compacts every 5^{th} data-word. An example of 2-D compaction is shown in Figures 7.5 and 7.6. Each MISR signature compacts a consecutive sequence of debug data words by k. $MS1$ represents the MISR signature and C1 denotes the data in 1^{st} clock cycle. MISR compacts data through 1^{st} to 5^{th} clock cycles, represented by {C1, C2, C3, C4, C5}, then 6^{th} to 10^{th} clock cycles,

represented by {C6, C7, C8, C9, C10} and so on. So, $MS1$ denotes the compaction of 1^{st} to 5^{th} cycles. The first signature in the first cycling register (i.e., $CR1$) is represented by CR1_1 which represents the debug data corresponding to clock cycles {C1, C8, C15, C22, C29}. Similarly, CR2_1 represents the debug data corresponding to clock cycles {C1, C6, C11, C16, C21, C26, C31} of first signature of the second cycling register (i.e., $CR2$). For $CR1$ and the MISR, we have seven signatures while for $CR2$, we have five signatures. The failing signatures of $MISR$ and $CR1$ are shown as shaded in Figure 7.5[5]

MS1 : {C1,C2,C3,C4,C5}	CR1_1 : {C1,C8,C15,C22,C29}
MS2 : {C6,C7,C8,C9,C10}	CR1_2 : {C2,C9,C16,C23,C30}
MS3 : {C11,C12,C13,C14,C15}	CR1_3 : {C3,C10,C17,C24,C31}
MS4 : {C16,C17,C18,C19,C20}	CR1_4 : {C4,C11,C18,C25,C32}
MS5 : {C21,C22,C23,C24,C25}	CR1_5 : {C5,C12,C19,C26,C33}
MS6 : {C26,C27,C28,C29,C30}	CR1_6 : {C6,C13,C20,C27,C34}
MS7 : {C31,C32,C33,C34,C35}	CR1_7 : {C7,C14,C21,C28,C35}

Figure 7.5: Intersection of debug signatures of MISR & CR1

Let's assume that there is an error in the debug data for C6, C11 and C23 clock cycles. After the comparison of the MISR signatures and that of cycling register (failing MSi with failing $CR1\_j$ and $CR2\_k$ i, $j \in$ {1,2....7} and $k \in$ {1,2....5}) with the respective golden response signatures, we obtain the clock cycles corresponding to the mismatch of these signatures. The clock cycles for the mismatch signatures are highlighted (in

MS1 : {C1,C2,C3,C4,C5}	CR2_1 : {C1,C6,C11,C16,C21,C26,C31}
MS2 : {C6,C7,C8,C9,C10}	CR2_2 : {C2,C7,C12,C17,C22,C27,C32}
MS3 : {C11,C12,C13,C14,C15}	CR2_3 : {C3,C8,C13,C18,C23,C28,C33}
MS4 : {C16,C17,C18,C19,C20}	CR2_4 : {C4,C9,C14,C19,C24,C29,C34}
MS5 : {C21,C22,C23,C24,C25}	CR2_5 : {C5,C10,C15,C20,C25,C30,C35}
MS6 : {C26,C27,C28,C29,C30}	
MS7 : {C31,C32,C33,C34,C35}	

Figure 7.6: Intersection of debug signatures of MISR & CR2

red color) in Figures 7.5 and 7.6. For the MISR, failing signatures are MS2, MS3 and

[5]Similarly, the failing signatures of $MISR$ and $CR2$ are shown as shaded in Figure 7.6

MS5. For cycling registers ($CR1$ and $CR2$), the failing signatures are CR1_2, CR1_4, CR1_6 and CR2_1, CR2_3 respectively. The intersection between debug data signatures proceeds as given below:

- Intersection of MISR and CR1 signatures provide suspect candidates as {C6, C9, C11, C13, C23, C25}
- Intersection of MISR and CR2 signatures provide suspect candidates as {C6, C8, C11, C13, C21, C23}
- Intersection of the above clock cycles yield suspect clock cycles (Cy) as **{C6, C11, C13, C23}**.

As per the approach of [187], when only one cycling register (i.e., $CR1$) is considered, the intersection of mismatching signatures of MISR and CR1 yields suspected clock cycles (Cy) as {C6, C9, C11, C13, C23, C25}. Note that with the proposed approach, we obtain length of Cy as only 4.

The probability of aliasing for MISR is very low, i.e., 2^{-K} where K is the size of the MISR. Typically, we utilize 32 or 64-bit MISR, so aliasing probability is negligible for MISR. Aliasing in a cycling register (CR) signature occurs when errors occur for an even number of bit in the same bit position in CUD execution data. The probability of aliasing in a cycling register is also depends on the error rate[187]. During our detailed debug experiments on various designs, we observed that aliasing in either of $CR1$ and $CR2$ does not have any significant impact on the error localization process.

7.4.2 Tag Bits (TBits) Generation for 2^{nd} Session

As stated earlier, in the second session (S_2), the trace buffer is used to store the suspect clock cycles (Cy). The triggering action for storing the debug data corresponding to these cycles from a stream of CUD execution is performed using tag bits ($TBits$). Before the onset of S_2, we store $TBits$ on the trace buffer to capture the selective debug data. Let's consider that without any compression, L clock cycles can be observed and traced in S_1.

Hence, the temporal observation window for S_2 is also given as L. With compression, this observation window is expanded by a factor (say, f) to $f*L$. This factor, f also depends on $TBits$ because the latter decides the start-points and end-points for the selective debug data capture in S_2.

One of the simplest ways to generate tag bit (or, storage triggers) is to have either a single bit for each cycle in L. However, the tag bit storage in this manner can incur significant overhead. For instance, Let's consider that Cy (which consists of {C6, C9, C11, C13,C23, C25} as per the technique of [187]) length is 6 and hence the tag bits are 35 in number (00000100101010000000001010000000000). The technique in [187] suggests putting a single tag bit for every two bits of the actual debug data. Hence, for L to be 35, we need a maximum of 18 tag bits. For the Cy set considered above, the tag bits would be 001011100001100000. Using this approach, debug data corresponding to 12 clock cycles is needed to be stored in trace buffer. Since the suspect cycle candidates are only six, an overhead of storing another 6 (i.e., {C5, C10, C12, C14, C24, C26}) cycles is incurred. We propose a difference-based $TBits$ generation methodology (shown in Algorithm 18). We compute the difference between the successive entries of Cy and store these differences instead of the original clock cycles. With this, the amount of tag bits to be stored is altered and the overhead for storing the unnecessary clock cycles (i.e., clock cycles absent in suspect cycle set, Cy) is reduced.

From the discussion in the previous sub-section, with the proposed technique, we have Cy as {C6, C11, C13, C23}. The difference (considering the first entry to be same as the original) between successive entries is {6, 5, 2, 10} i.e., {0110, 0101, 0010, 1010}. If we use 4 bit for every position, then we need 16 tag bits without any overhead of extra cycles. This is because if the position 6 is known, based on the successive differences all other clock cycles can be exactly identified. For instance- adding 5 to position 6 leads to 11^{th} clock cycle. However, if we consider 3 bits for storing the successive differences, the scenario changes slightly. Under this case, we need to store the differences as {6, 5, 2, 7, 3} i.e., {110, 101, 010, 111, 011}. Here, we need to store clock cycle C20 as an overhead (since it is not present in Cy set). Thus, with reduced length for storing each successive

difference, there can be overhead of storing additional cycles (debug data corresponding to which are not erroneous). However, these additional cycles are significantly lower than the tag bit compression technique proposed in [187].

Algorithm 18: *TBitsGeneration*

Input: Cy, $TagRegister\ Size$
Output: $TBits$

```
1  diffset ← Cy[0];
2  t = 0;
3  trs = TagRegister Size;
4  while t < length(Cy) do
5  |   diff = Cy[t+1] - Cy[t];
6  |   diffset ← diffset ∪ diff;
7  end
8  mdiff = maximum of diffset;
9  if mdiff < (2^trs-1) then
10 |   TBits ← Boolean encoding of all u in diffset;
11 |   else
12 |   |   exdiff = mdiff - maxb;
13 |   |   diffset ← diffset ∪ exdiff;
14 |   |   TBits ← Boolean encoding of all u in diffset;
15 |   end
16 end
```

7.4.3 Fine-grained Spatial Visibility in 2^{nd} Session

7.4.3.1 Illustration and Description

We propose a mechanism to achieve fine-grained spatial error localization through effective debug data segregation (i.e., the segregation of the individual debug data-word/debug signature). Note that since data-word width is same as signature width, the segregation of both the signature and actual data-word follows the same pattern. Once suspect clock cycles (Cy) are identified, across each of these clock cycles, complete data-word (consisting of W length) needs to be stored in the trace buffer. However, the error may be only in certain positions of that data-word. The proposed methodology (shown in Algorithm 19) segregates the debug data into groups (i.e., bytes).

With segregation, the signatures consisting of data-words of W length translates to W/q chunks of debug data where q is the length of each chunk. We assume W as 32 and fix q as 8 leading to 4 (= $W/8$) bytes of debug data termed as W_0, W_1, W_2 and W_3. This allows selective data capture among these 8 bytes instead of the complete data-word of length W during S_2 session. Along with storing successive difference of Cy entries, we need to store these byte positions also. The following example provides an illustration of the data segregation. For the error positions (in terms of clock cycles) shown in Fig. 7.5 and 7.6, we assume that clock cycle C6 has error in the 1st byte in the respective data-word (of length 32 bits); C11 has error in the 3rd byte and C23 has error in the 2nd byte. The intersection between different segregated debug data signatures proceeds as follows: (format adopted here is {v: {Cq, Cr,..}} meaning that for clock cycles Cq and Cr, v^{th} byte of the signature data-word is erroneous and so on.)

- Intersection between MISR and CR1 signatures (specifically[6] MS2 and CR1_6, MS2 and CR2_1) leads to: **{1: {C6, C13}}; {2: {C9, C23}}; {3: {C11, C25}}**
- Intersection between MISR and CR2 signatures (MS3 and CR1_4, MS3 and CR2_3) leads to: **{{1: {C6, C11, C21}; {2: {C8, C13, C23}; {3: {C6, C11, C21}}**
- Final intersection signatures (MS5 and CR1_2, MS5 and CR2_1) leads to: **{1 : C6} {2 : C23} {3 : C11}**

Since Cy is {C6, C11, C23}, we obtain the successive difference as (6, 5, 12). Therefore, $TBits$ (assuming 4 bits for each position) required are 0110 **0001** 0101 **0100** 1100 **0010**. Here, the bold part represents byte selection (with one-hot encoding[7] leading to 3^{rd} byte position for $C11$ encoded as **0100**) resulting into 24 bits storage in TB in total. With data segregation, Cy has only 3 entries and as their byte-positions are also known, we require only 3*8=24 bits to be stored in TB during second session (S_2). These 24 bits involve data-word byte corresponding to the exact erroneous clock cycles only, thereby

[6]Since these signatures are the failing ones, data-words of these signatures for each cycle should be intersected byte-wise among each other.

[7]Any other encoding is also possible like the way we encode cycle positions. The corresponding decoder can be similar to a counter.

avoiding additional (i.e., debug data corresponding to the non-erroneous clock cycles) storage in TB. This procedure is shown in Algorithm 19.

Algorithm 19: *DataSegregation*

Input: *CUD Data Sig*, *Golden Data Sig*, *Cy*
Output: *DByte*

```
1  Dbyte_Cy ← φ;
2  DByte ← φ;
3  w = W/8 (Since 8bits=1byte);
4  for r in Cy do
5  |  D ← CUD Data Sig for Cy[r] ;
6  |  DG ← Golden Data Sig for Cy[r] ;
7  |  for z in 0 to w-1 do
8  |  |  if D[z] ≠ DG[z] then
9  |  |  |  Dbyte_Cy[r] ← z;
10 |  |  end
11 |  end
12 |  DByte ← DByte ∪ Dbyte_Cy[r] ;
13 end
```

For the particular debug scenario considered above (i.e., Figures 7.5 and 7.6 where Cy length is 6 with 1 MISR and 1 CR as elaborated in Section 7.4.1), Table 7.2 shows the comparison of the required tag bits ($TBits$) and the number of bits ($Bits$) to be stored in the trace buffer (TB) for the proposed methods and the technique proposed in [187]. $M1$ represents our methodology based on two-step signature intersection (via 1 MISR & 2 CR) elaborated in Section 7.4.1 and 7.4.2. $M2$ represents the fine-grained spatial error localization explained in this section. Corresponding to all these methods, tag bits ($TBits$) form varies and the amount of $TBits$ also varies. From hereafter, Cy length is denoted by $|Cy|$. Note that $TBits$ increases in $M2$ because of the additional encoding of byte-positions along with the cycle (Cy) positions. The counter (4-bit) is similar to that shown in Fig. 7.4. As illustrated before, the length of this counter may vary from one debug experiment to another. However, its optimal size can be empirically decided based on the initial observation window (L).

Table 7.2: Tag bit size & on-chip storage calculation

	Compression	$\|Cy\|$	$TBits$ form	$TBits$	$Bits$
[187]	MISR, 1 CR	6	1 bit for 1 cycle	35	6*32 = 192
[187]	MISR, 1 CR	6	1 bit for 2 cycles	18	12*32 = 384
$M1$	MISR, 2 CR	4	4-bit counter, diff.-based	16	4*32 = 128
$M2$	MISR, 2 CR, byte-wise	3	4-bit counter, diff.-based + byte-tags	24	3*8 = 24

7.4.3.2 Implementation of Segregation-based Storage in TB

Due to the partially specified data-words in $M2$ method, we need to ensure the storage of $Bits$ in TB in a manner so that the byte-positions of the debug data bits do not overlap. To facilitate this, we propose a minor modification in the scheme shown in Figure 7.2. This arrangement ensures that the writing of partially specified data-words into TB is

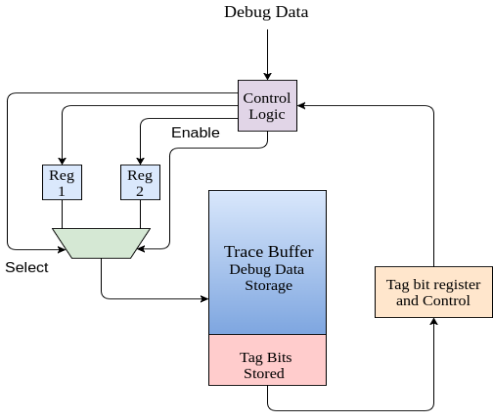

Figure 7.7: Storage of fine-grained debug data

serialized via two registers i.e., Reg1 and Reg2 (Reg1 dumps its contents first into TB, then Reg2), each having a length of W controlled by a multiplexer as shown in Figure 7.7. Therefore, the incoming debug data in the segregated form is stored into TB via Reg1 and Reg2. Note that the control logic in Fig. 7.7 ensures that different debug bits are dumped into Reg1 and Reg2, one at a time. The multiplexer ensures the selection of Reg1 for writing into TB while the incoming debug data arriving at that time, is stored into Reg2. Table 7.3 shows these control values for different conditions. Once the

Table 7.3: Different conditions of fine-grained storage

Reg1 condition	Reg2 condition	Select	Enable
partially empty	partially empty	X	0
full	partially empty	0	1
partially empty	full	1	1

contents of Reg1 is completely written into TB, the multiplexer in Fig. 7.7 selects Reg2 for writing into TB. The enable signal in the multiplexer ensures dumping of register contents into TB only when it is completely full (i.e., 4 bytes of data are stored either from single clock cycle or multiple clock cycle of the Cy set).

7.4.3.3 Decoding TB Contents during Off-chip Analysis

Consider that L is eight clock cycles numbered A_0 to A_7. With this set-up, we have Cy as 4 clock cycles (A_2, A_3, A_5 and A_7) and their suspect byte positions are W_2, W_0, W_2 and W_3 respectively. Since TBw is considered as 32, its one location gets filled up with these byte positions. While offloading TB contents, as we know the suspect clock cycles (through $TBits$), we can easily understand the debug data during detailed off-line error localization process after the second session. Note that we store the byte positions (i.e., W_0, W_1, W_2, W_3) also in $TBits$ which assist in determining the correspondence of a particular byte to a suspect clock cycle.

7.4.3.4 Fine-grained Error Localization

With the method in $M2$, we store the byte-wise partitioned debug data signatures in TB in S_1 session. After the end of S_2 session, once these segregated signatures are offloaded, in the second session, the errors can be localized to a resolution of the particular byte (4 bits) in data-words (of total length 32 bits). Considering the debug scenario in previous subsection for 4 clock cycles (A_2, A_3, A_5 and A_7). With byte-wise segregation, we can conclude that the error belongs to either of the W_2, W_0, W_2 and W_3 portion of the data-word for the respective clock cycles instead of the complete data words (i.e., all of W_i, $i \in \{0,1,2,3\}$) of these four suspect clock cycles.

7.5 Experimental Setup, Results and Analysis

7.5.1 Experimental Setup

We performed error injection experiments on a wide range of circuits- 2 from ISCAS'89 suite [145] (s38417, s38584), three are processor cores ($p16c5x$, $or1200$, $msp430$) from Opencores [143]. We injected errors randomly (i.e., bit-flipping in the traced signals) at different error rates (ER) and a varying L, W across these designs. Either constrained-random testbenches or applications written in C language (after compilation through respective tool-chains) are utilized for the design simulation. We injected errors in the data-words (of total length as 32, 40 or 64 bits) in 4 different ways for all the designs: a) single cycle, single bit error in a randomly chosen data-word (**SCSB**), b) single cycle, multiple bits (**SCMB**) generally 2 to 5 bits of the data-word, c) multiple cycle, multiple bits (**MCMB**), and d) multiple cycle, single bit (**MCSB**) ranging from 2 to 10 clock cycles. All error injection experiments (i.e., a single row in the Tables 7.4 to 7.8) are repeated for 200 iterations resulting into a total of 3200 iterations for s38417 circuit and 4000 iterations for each of the other four circuits ($p16c5x$, $msp430$, $s38584$, $or1200$). Therefore, across all the four different error types, the reported results in Tables 7.4 to 7.8 have been obtained after averaging for 200 iterations for each ER. In accordance with the terminology adopted in [187],[188],[189] error rate (ER) is defined by:

$$ER = \frac{|Cye|}{L} \tag{7.6}$$

where $|Cye|$ is denotes the number of clock cycles in which the actual error is present. In Tables 7.4 to 7.8, for 3 methods (i.e., $M1$, $M2$ and [187]), we report the averaged debug data storage in S_2 (i.e., $Bits$) and the averaged number of suspect clock cycles (i.e., $|Cy|$) for various designs.[8]

Across the designs, L and W are varied to assess the impact of aliasing and the

[8]From hereafter, $|Cy|$ refers to the averaged number of suspect clock cycles computed across multiple iterations of debug experiments.

debug data storage under different error types. Consider the case of uncompressed on-chip debug data collection, then we need to store L*W bits. For instance- in results of s38417 circuit (Table 7.4) with L as 512 and W as 32, we need to store 512*32 bits during uncompressed debug data acquisition. Under an on-chip debug data compression scheme, if the suspect clock cycle(s) can identified (out of total 512 cycles), we need to store only 32 bits (corresponding to the signature of only that clock cycle). This illustration shows that with debug data compression, a significant reduction in the on-chip storage can be achieved (which is shown as *Bits* in the three columns of Tables 7.4 to 7.8). Aliasing situations generally occur for MCSB error type. Although these occurrences are low in number, efforts should be made to decrease them to zero. We believe that the length of cycling registers is important for containing the aliasing in signatures.

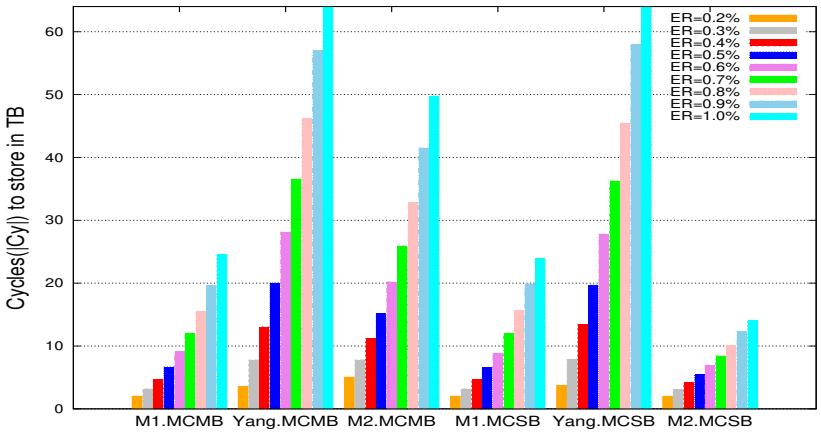

Figure 7.8: Variation of Cy with ER of $p16c5x$ circuit

The results from Table 7.5 are shown in Figure 7.8 for better visualization. As it can be observed, with the proposed methodologies ($M1$ and $M2$) provide lower values of $|Cy|$ than that of Yang et al. [187]. We define on-chip storage utilization metric as below (for the sake of brevity, we do not show this metric in results of Tables 7.4 to 7.8):

$$Util = \frac{Bits}{TBw * TBd} \tag{7.7}$$

Given the capacity of TB as TBw*D_{tb}, if $Bits$ are to be stored in S_2 in the successive debug experiments, available TB space decreases as $Util$ in the second session increases. Increase in $Util$ happens if $Bits$ increases. However, $Bits$ should be low as possible.

Therefore, better utilization of TB essentially means a low value of the metric in Equation 7.7. Here, we assume that a total of p debug experiments[9] can be done when no debug data compression has been employed. The first experiment runs from 1^{st} to L^{th} clock cycle, the second one runs from $(L+1)^{th}$ to $(2L)^{th}$ cycle[10] and so on.

During the second session (S_2) of this debug experiment, the observation window can be effectively elongated to $TExpansion^*L1$. Consider TBw as 32 and D_{tb} as 1024 and the total length of CUD execution is 32768 ($= 2^{15}$) cycles, in the first session, observation window is 1024 only. After the intersection of signatures, if $|Cy|$ is obtained as 28, we obtain an $TExpansion$ of 36.57 (putting 1024 in numerator of the expression for $TExpansion$ in Equation 7.2. Consequently, in the second session, the temporal observation window is expanded to 37449 cycles which is more than 32768 cycles allowing the complete CUD execution to be stored in only one iteration of S_2 in the debug experiment compared to the scenario of five such sessions (i.e., $p = 5$). Let's assume that in this scenario we obtain $Bits$ as 1794 which leads to $Util$ as 0.054 meaning that approximately 0.946 of TB space is available for other S_2 sessions in the debug experiment. In other words, when no compression is performed, we must have 5 S_2 sessions (each covering 1024 clock cycles) for capturing the debug data and TB gets filled up in each of these iterations. With compression, we succeed in finishing this in a single S_2 iteration.

7.5.2 Metrics for Comparative Evaluation

For comparative estimation of the usage of on-chip storage (which reflects the temporal visibility expansion) with the proposed methodology, we define two metrics ($Obsv_{Co}$ and $Obsv_{CoBy}$) in Equations 7.8 and 7.9.

$$Obsv_{Co} = \frac{Bits.(MISR,\ One\ CR)}{Bits.(MISR,\ Two\ CR)} \tag{7.8}$$

[9]So, we can have maximum of p number of first and second sessions.

[10]We assume that the second temporal observation window remains same as initial observation window, $L1$. In reality, this may also vary.

The denominator in Equation 7.8 indicates $Bits$ (i.e., the compressed debug data) to be stored in TB in S_2, obtained by compression with the proposed method which involves MISR and two CR (i.e., $M1$). The numerator denotes amount of $Bits$ to be stored in TB, achieved with MISR and one CR (the denominator shows $Bits$ obtained with the technique proposed in [187]). As stated previously, the proposed method requires lower amount of on-chip storage. Thus, the denominator attains lower values than the numerator yielding always a value of $Obsv_{Co}$ greater than unity. It is obvious that higher the values of $Obsv_{Co}$ metric, better is the utilization of on-chip storage.

$$Obsv_{CoBy} = \frac{Bits.(MISR,\ One\ CR)}{Bits.(MISR,\ Two\ CR,\ segbyte.)} \tag{7.9}$$

A similar terminology is valid for the second metric, $Obsv_{CoBy}$ with the only difference that the denominator encompasses the proposed methodology with data segregation (i.e., $M2$) represented here as segbyte. (segregation of bytes). It is obvious that for $Obsv_{CoBy}$ metric, the denominator has a lower value than that of $Obsv_{Co}$ metric. Because of this, $Obsv_{CoBy}$ attains a higher value than that of $Obsv_{Co}$.

7.5.3 Comparative Evaluation Results

In Tables 7.4 to 7.8, 10^{th} column ($Exact$) represents the exact number of bits in which errors have been injected and 11^{th} column ($Alia.$) shows the number of iterations in which we encountered aliasing of debug data signatures. Since we observed lower number of such iterations, it can be concluded that aliasing of signatures (due to either MISR or CR) is tolerable under the proposed scheme. Note that the authors in [187] compute the efficiency of their method through temporal observability expansion (in terms of Cy), however, our metrics ($Obsv_{Co}$ and $Obsv_{CoBy}$) are aimed at the maximization of on-chip storage (i.e., TB) utility. From Tables 7.4 to 7.8, it can be observed that $|Cy|$ for $M1$ is lower than $|Cy|$ for [187] for all error rates and different error configurations. For SCMB and MCMB configuration, $|Cy|$ for $M2$ is slightly higher than that of [187] and $M1$. In cases of these errors, because of segregation, the same $|Cy|$ is counted

more than once (as multiple data-words may contain the error) resulting in the overall increase in values of $|Cy|$. At other error rates, there is significant decrease in Cy of $M2$ than that of [187], resulting in fine-grained visibility. Subsequently, the number of debug experiments is also brought down because with compression, a larger observation window (L) is accommodated within a single session iteration.

For s38417 circuit (Table 7.4), under MCMB error type configuration, when experiments are performed with ER of 0.78% i.e., error is in 4 cycles, the technique in [187] provides an averaged Cy as 12.6, whereas we obtain averaged Cy as 4.285 and 10.44 for $M1$ and $M2$ respectively. Therefore, for L of 512 clock cycles, we need to store debug data corresponding to 4.285, 12.6 and 10.44 clock cycles for the technique in $M1$, [187] and $M2$ respectively. Note that with data segregation feature (i.e., $M2$) we get lower $Bits$ (= 83.52) as compared to $Bits$ (= 137) for ER of 0.78% in MCMB error type configuration of s38417 circuit (Table 7.4). Typically, the realistic design/electrical error scenarios (i.e., gate replacement, wire exchange, bit-flip etc.) can correspond to any of the four error scenarios considered above. Figure 7.9 shows the average number of clock cycles ($|Cy|$) for which data is to be stored in TB under different approaches ($M1$, $M2$ and [187]) for MCSB type error injection. Hereafter, the technique in [187] is denoted by Yang. It can be observed from this figure that the proposed methodologies ($M1$, $M2$) require storage of significantly lower number of clock cycles (Cy). A similar trend can be observed for MCMB type error injection from Tables 7.4 to 7.8. The maximum values

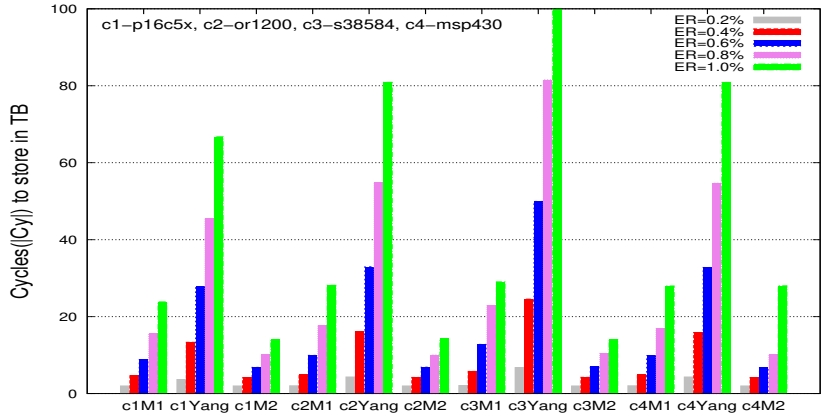

Figure 7.9: Variation of $|Cy|$ with ER of MCSB type for different circuits

of $Obsv_{Co}$ and $Obsv_{CoBy}$ typically belong to MCSB error type configuration. Further,

we observed aliasing in the compressed debug signatures during a maximum of 6 iterations out of a total of 200 error injection iterations. The maximum aliasing occurs for s38417 and p16c5x for respective ER of 1.17% and 1.04% in MCSB error type configuration. At lower values of ER and for configurations of SCSB, SCMB and MCMB, we observed zero aliasing. For msp430 circuit, we observed the maximum value of $Obsv_{CoBy}$ as 44.20 (at 0.9% in MCSB configuration) which means that compared to technique in [187], the proposed method, $M2$ needs 44.20 times lower on-chip storage (i.e., storage in TB). Thus, accordingly a much larger L (i.e., the temporal observation window) can be accommodated with the proposed methodology in a debug experiment.

We show the variation in $Obsv_{CoBy}$ values with different error rates (ER) in Fig. 7.10. There is a gradual increase in the $Obsv_{CoBy}$ values when ER is increased, which means that the proposed method ($M2$) provides effective visibility expansion and utilization of TB as compared to [187] even at higher ER.

Table 7.4: s38417 results for L=512, W=32

Type	*Error*(%)	$\|Cy\|$ (M1)	$\|Cy\|$[187]	$\|Cy\|$ (M2)	Bits (M1)	Bits[187]	Bits (M2)	*Exact*	*Alia.*	$Obsv_{Co}$	$Obsv_{CoBy}$
SCSB	0.20	1	1	1	32	32	8	1	0	1.00	4.00
SCMB	0.20	1	1	2.49	32	32	19.92	1	0	1.00	1.61
MCMB	0.39	2	3.545	5.2211	64	113.4	41.77	2	0	1.77	2.72
	0.59	3.12	7.42	7.77	99.8	237.4	62.16	3	0	2.38	3.82
	0.78	4.285	12.6	10.44	137	403.2	83.52	4	0	2.94	4.83
	0.98	5.885	18.49	14.305	188	591.7	114.4	5	0	3.14	5.17
	1.17	7.425	25.21	17.51	238	806.7	140.1	6	0	3.40	5.76
	1.37	9.6	33.025	21.81	307	1057	174.5	7	0	3.44	6.06
	1.56	11.875	41.345	26.735	380	1323	213.9	8	0	3.48	6.19
	1.76	14.69	50.25	32.45	470	1608	259.6	9	0	3.42	6.19
	1.95	17.75	59.75	38.34	568	1912	306.7	10	1	3.37	6.23
MCSB	0.39	2	3.6	2	64	115.2	16	2	0	1.80	7.20
	0.59	3.06	7.57	3.022	97.9	242.2	24.18	3	1	2.47	10.02
	0.78	4.35	12.63	4.09	139	404.2	32.72	4	3	2.90	12.35
	0.98	5.86	18.85	5.27	188	603.2	42.16	5	4	3.22	14.31
	1.17	7.52	25.54	6.39	241	817.3	51.12	6	6	3.40	15.99

7.5.4 Experiments with Burst Errors

We performed burst error experiments (i.e., error occurs in bursts of 10 cycles) for different designs with 200 iterations done for each design. Fig. 7.11 shows the averaged number of cycles (Cy) for which debug data is to be stored in TB under burst error scenarios. It is obvious that for this error type, $|CYe|$ equals 10. To maximally utilize the on-chip storage, $|Cy|$ should be as low as possible. $M1$ achieves lower $|Cy|$ values

Table 7.5: p16c5x results for L=960, W=32

Type	*Error*(%)	$\|Cy\|$ (M1)	$\|Cy\|$[187]	$\|Cy\|$ (M2)	Bits (M1)	Bits[187]	Bits (M2)	*Exact*	*Alia.*	$Obsv_{Co}$	$Obsv_{CoBy}$
SCSB	0.104	1	1	1	32	32	8	1	0	1.00	4.00
SCMB	0.104	1	1	2.51	32	32	20.1	1	0	1.00	1.59
MCMB	0.208	2.015	3.565	5.03	64.5	114.1	40.2	2	0	1.77	2.83
	0.312	3.175	7.77	7.78	102	248.6	62.2	3	0	2.45	3.99
	0.416	4.625	13.02	11.215	148	416.6	89.7	4	0	2.82	4.64
	0.52	6.655	20.02	15.13	213	640.6	121	5	1	3.01	5.29
	0.624	9.07	28.12	20.18	290	899.8	161	6	0	3.10	5.57
	0.728	11.92	36.575	25.92	381	1170	207	7	2	3.07	5.64
	0.832	15.455	46.23	32.85	495	1479	263	8	0	2.99	5.63
	0.936	19.625	57.015	41.485	628	1824	332	9	0	2.91	5.50
	1.04	24.59	67.27	49.78	787	2153	398	10	0	2.74	5.41
MCSB	0.208	2.01	3.685	2.005	64.3	117.9	16	2	0	1.83	7.35
	0.312	3.16	7.8592	3.03	101	251.5	24.2	3	1	2.49	10.38
	0.416	4.67	13.35	4.17	149	427.2	33.4	4	1	2.86	12.81
	0.52	6.59	19.69	5.45	211	630.1	43.6	5	1-2	2.99	14.45
	0.624	8.83	27.79	6.85	283	889.3	54.8	6	1-2	3.15	16.23
	0.728	11.97	36.25	8.4	383	1160	67.2	7	2-3	3.03	17.26
	0.832	15.6	45.47	10.13	499	1455	81	8	2-3	2.91	17.95
	0.936	19.86	57.93	12.31	636	1854	98.5	9	0	2.92	18.82
	1.04	23.86	66.76	14.09	764	2136	113	10	2-6	2.80	18.95

Table 7.6: msp430 results for L=1000, W=40

Type	*Error*(%)	$\|Cy\|$ (M1)	$\|Cy\|$[187]	$\|Cy\|$ (M2)	Bits (M1)	Bits[187]	Bits (M2)	*Exact*	*Alia.*	$Obsv_{Co}$	$Obsv_{CoBy}$
SCSB	0.1	1	1.17	1	40	46.8	8	1	0	1.17	5.85
SCMB	0.1	1	1.13	2.885	64	72.32	23.08	1	0	1.13	3.13
MCMB	0.2	2.06	4.33	5.53	131.5	277.1	44.24	2	0	2.11	6.26
	0.3	3.36	9.28	8.645	215	593.9	69.16	3	0	2.76	8.59
	0.4	4.94	16.1	12	316.2	1030	96	4	0	3.26	10.73
	0.5	7.09	24.1	16.315	453.8	1542	130.5	5	0	3.40	11.81
	0.6	10.2	33.6	22.355	655.4	2149	178.8	6	0	3.28	12.02
	0.7	13.4	43.5	29.415	857	2787	235.3	7	0	3.25	11.84
	0.8	17.5	56	36.625	1118	3586	293	8	0	3.21	12.24
	0.9	22.3	67.5	44.592	1428	4322	356.7	9	0	3.03	12.12
	1	28.2	82.2	55.805	1805	5263	446.4	10	0	2.91	11.79
MCSB	0.2	2.04	4.33	2.005	130.6	276.8	16.04	2	0	2.12	17.26
	0.3	3.29	9.36	3.06	210.6	598.7	24.48	3	0	2.84	24.46
	0.4	4.95	15.9	4.21	316.8	1017	33.68	4	1-2	3.21	30.19
	0.5	7.21	24.1	5.39	461.5	1544	43.12	5	2-3	3.35	35.82
	0.6	9.86	32.8	6.76	631	2100	54.08	6	3	3.33	38.84
	0.7	13.2	44.4	8.38	844.2	2838	67.04	7	3	3.36	42.34
	0.8	16.9	54.7	10.13	1082	3500	81.04	8	2	3.23	43.18
	0.9	22.9	70.5	12.76	1463	4512	102.1	9	2	3.08	44.20
	1	28	80.9	28.03	1794	5178	224.2	10	1	2.89	23.09

than compression with 1 CR, 1 MISR techniques (Yang's approach [187]). Note that because of data segregation, $M2$ has significantly higher Cy values. However, the method $M2$ achieves significantly lower $Bits$ required to be stored in TB because of the byte segregation from the incoming debug data. Figure 7.12 shows the averaged amount of $Bits$ required to be stored in TB for $M1$, Yang [187] and $M2$. For almost all the designs, $M1$ method achieves lower amount of $Bits$ than Yang [187], while method $M2$ provides the lowest $Bits$ for all the designs. Therefore, the increase in $|Cy|$ values with $M2$ is overshadowed by data segregation resulting in lower $Bits$ values.

Table 7.7: s38584 results for L=960, W=64

Type	*Error*(%)	$\|Cy\|$ (M1)	$\|Cy\|$[187]	$\|Cy\|$ (M2)	Bits (M1)	Bits[187]	Bits (M2)	*Exact*	*Alia.*	$Obsv_{Co}$	$Obsv_{CoBy}$
SCSB	0.104	1	1.875	1	64	120	8	1	0	1.88	15.00
SCMB	0.104	1	1.845	2.89	64	118.1	23.1	1	0	1.84	5.11
MCMB	0.208	2.08	6.87	6.035	133.1	439.7	48.3	2	0	3.30	9.11
	0.312	3.585	14.6	9.205	229.4	934.4	73.6	3	0	4.07	12.69
	0.416	5.89	24.28	14.115	377	1554	113	4	0	4.12	13.76
	0.52	8.775	36.105	19.49	561.6	2311	156	5	0	4.11	14.82
	0.624	12.46	49.6	25.45	797.4	3174	204	6	0	3.98	15.59
	0.728	17.65	65.31	33.32	1130	4180	267	7	0	3.70	15.68
	0.832	23.73	82.8	42.245	1519	5299	338	8	0	3.49	15.68
	0.936	29.88	97.96	52.27	1912	6269	418	9	0	3.28	14.99
	1.04	38.38	116.14	65.585	2456	7433	525	10	0	3.03	14.17
MCSB	0.208	2.12	6.78	2.01	135.7	433.9	16.1	2	0	3.20	26.99
	0.312	3.67	14.64	3.15	234.9	937.0	25.2	3	1	3.99	37.18
	0.416	5.8	24.56	4.25	371.2	1572	34	4	1	4.23	46.23
	0.52	8.89	37.46	5.53	569	2397	44.2	5	2	4.21	54.19
	0.624	12.81	49.97	7.05	819.8	3198	56.4	6	2	3.90	56.70
	0.728	17.47	64.25	8.15	1118	4112	65.2	7	0	3.68	63.07
	0.832	22.92	81.54	10.46	1467	5219	83.7	8	3-4	3.56	62.36
	0.936	29.05	96.85	12.21	1859	6198	97.7	9	2-3	3.33	63.46
	1.04	37.22	115.14	14.06	2382	7369	112	10	3	3.09	65.52

Table 7.8: or1200 results for L=1000, W=40

Type	*Error*(%)	$\|Cy\|$ (M1)	$\|Cy\|$[187]	$\|Cy\|$ (M2)	Bits (M1)	Bits[187]	Bits (M2)	*Exact*	*Alia.*	$Obsv_{Co}$	$Obsv_{CoBy}$
SCSB	0.1	1	1.095	1	40	43.8	8	1	0	1.09	5.47
SCMB	0.1	1	1.14	2.66	64	72.96	21.3	1	0	1.14	3.43
MCMB	0.2	2.03	4.24	5.49	129.9	271.4	43.9	2	0	2.09	6.18
	0.3	3.32	9.195	8.525	212.5	588.5	68.2	3	0	2.77	8.63
	0.4	4.95	15.96	12.07	316.8	1021	96.6	4	0	3.22	10.58
	0.5	7.145	24.4	16.82	457.3	1562	135	5	0	3.41	11.61
	0.6	9.885	34.23	21.775	632.6	2191	174	6	0	3.46	12.58
	0.7	13.52	43.75	28.825	865.3	2800	231	7	0	3.24	12.14
	0.8	17.79	55.36	36.66	1139	3543	293	8	0	3.11	12.08
	0.9	23.01	68.12	46.305	1473	4360	370	9	0	2.96	11.77
	1	28.57	81.11	56.24	1828	5191	450	10	0	2.84	11.54
MCSB	0.2	2.05	4.34	2	131.2	277.8	16	2	0	2.12	17.36
	0.3	3.33	9.115	3.055	213.1	583.4	24.4	3	0	2.74	23.87
	0.4	4.97	16.15	4.22	318.1	1034	33.8	4	0	3.25	30.62
	0.5	7.2	24.33	5.41	460.8	1557	43.3	5	1	3.38	35.98
	0.6	9.95	32.97	6.89	636.8	2110	55.1	6	2	3.31	38.28
	0.7	13.77	44.44	8.48	881.3	2844	67.8	7	2	3.23	41.92
	0.8	17.68	54.82	10.04	1132	3508	80.3	8	3	3.10	43.68
	0.9	21.67	66.45	12.13	1387	4253	97	9	3	3.07	43.83
	1	28.16	80.93	14.34	1802	5180	115	10	4	2.87	45.15

7.5.5 Variation of $|Cy|$ and $TBits$ with ER and Tag Sizes

In the proposed methodology, tag bit compression is an important feature and it depends on the size of the tag bit register. To find the optimal size of tag bit register, we performed a different set of error injection experiments with different tag register sizes (100 error injection iterations for each size). The register sizes (trs) range from 2 to 13 bits across all the designs. It is imperative that with change in trs, the amount of $TBits$ varies. As elaborated in previous illustration, with change in $TBits$, the number of cycles ($|Cy|$) for which data is to be stored varies significantly. We show the variation in averaged $|Cy|$ and avg. tag bit ($TBits$) storage requirement with different tag sizes in Fig. 7.13 for

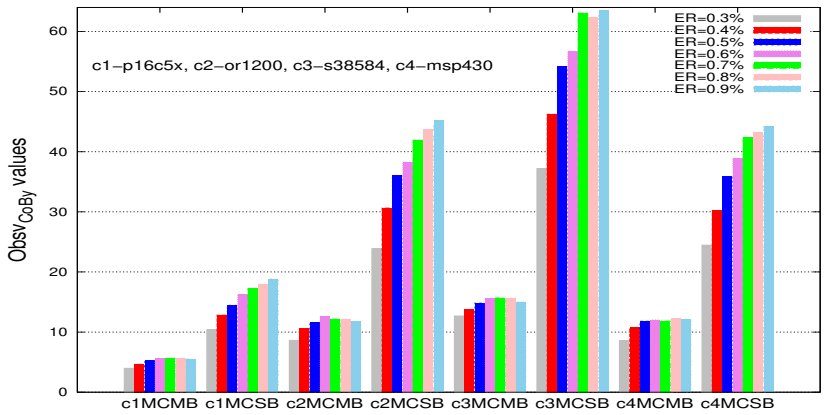

Figure 7.10: Variation of $Obsv_{CoBy}$ with ER for different error types

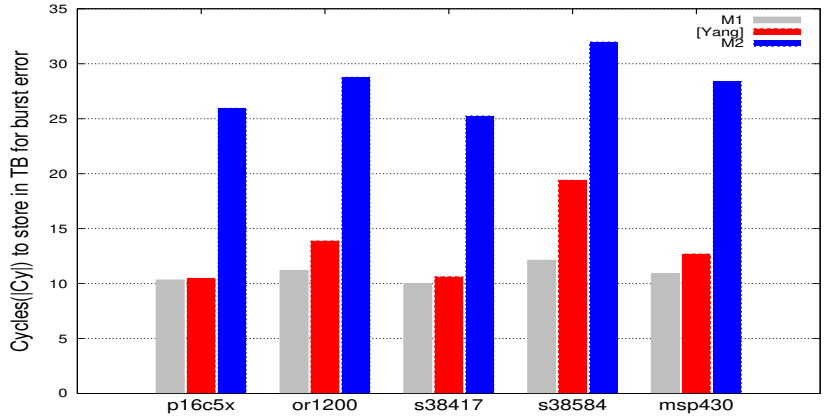

Figure 7.11: Comparison of $|Cy|$ for burst error for different circuits

s38417 circuit. For the technique in [187] where in for every two clock cycles, 1 tag bit is required, in we need 256 tag bits (for $L = 512$) in total. Therefore, upon increasing tag register sizes (trs), Cy remains same for [187] as shown in Fig. 7.13 and 7.15. With an increase in tag-register size (trs) upto a certain extent, $|Cy|$ decreases with the proposed method (shown here as Prop meaning 2 CR, 1 MISR for compression and successive difference-based tag bit generation i.e., $M1$ with our tag generation technique). When tag-register of length ten is chosen, we get cycle ($|Cy|$) reduction of 79.30%, 80% and 80% for p16c5x, msp430 and or1200 respectively over the technique proposed in [187]. For s38417 and s38584 designs, we get cycle ($|Cy|$) reduction of 80% and 79.50% respectively over the technique proposed in [187] with tag-register lengths (trs) of six and nine. Ideally, with increase in trs, $|Cy|$ of Yang's approach [187] should not vary for a design because $TBits$ generation in this approach is fixed as one bit for every two clock cycles

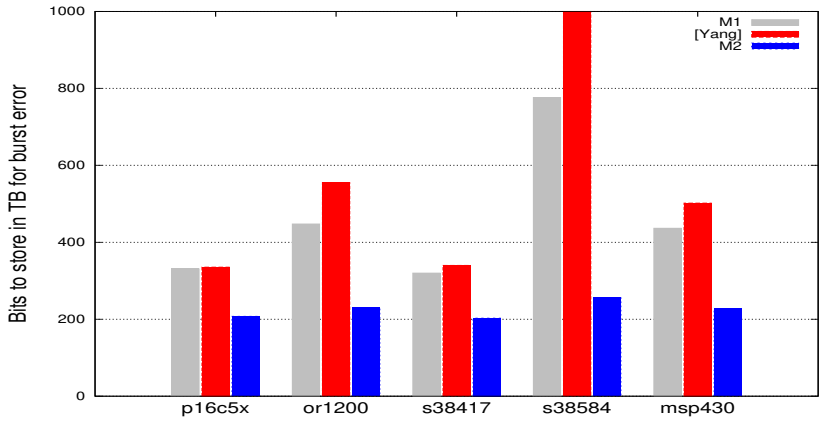

Figure 7.12: Comparison of $Bits$ for burst error for different circuits

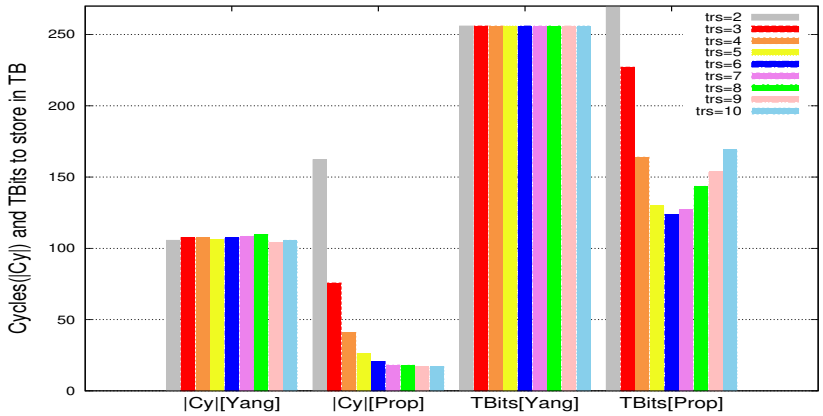

Figure 7.13: Variation of $|Cy|$, $TBits$ with trs for s38417 (L=512, W=32)

in L. However, since we perform random error injection experiments (100 in number) for each value of trs, there is a slight difference in Cy values of Yang [187] in Figures 7.13 and 7.14 when trs is changed. For different circuits, we show the variation in averaged $|Cy|$ with variation in trs under our methodology in Fig. 7.14. For L of 512, a tag size of 5 or 6 is optimal, while for L as 1000 or 980, tag size of 9 or 10 is the most suitable one. The variation in averaged $TBits$ with variation in trs is shown in Fig. 7.15. When (trs) is chosen as ten, we get tag bits ($TBits$) reduction of 49%, 43% and 44% for p16c5x, msp430 and or1200 respectively over the tag generation technique proposed in [187].

For s38417 and s38584 designs, we get avg. tag bits ($TBits$) reduction of 51% and 21% respectively over the technique proposed in [187] with tag-register lengths of six and nine. This variation in reduction percentage and tag-register length for these two circuits is because we have chosen different L for both of them (960 for s38584, 512

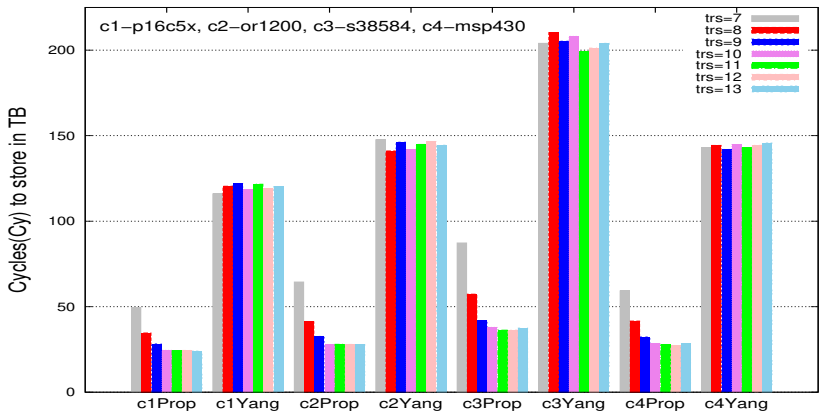

Figure 7.14: Variation of $|Cy|$ with different trs for different circuits

for s38417). As stated before, Figures 7.14 and 7.15 show that the tag bits ($TBits$) reduction (which in turn affects $|Cy|$ reduction) is a function of the tag-register size (trs) in the proposed methodology. Therefore, trs plays an important role in the expansion of temporal visibility window in the proposed methodology.

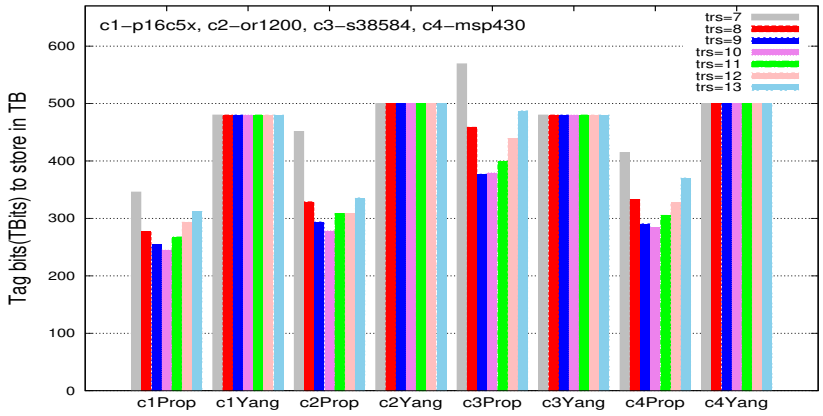

Figure 7.15: Variation of $TBits$ with different trs for different circuits

7.5.6 Overhead analysis

We implemented the proposed debug infrastructure (scheme shown in Fig. 7.2) and the architecture proposed in [187] in RTL and synthesized using Synopsys Design Compiler with 32 nm educational standard cell library (provided by Synopsys). Across all the designs, we observed a respective increase of 0.6% and 1.025% in area overhead for $M1$ and $M2$ over the scheme of [187]. The relative increase in power overhead was 1.6% and 3.4% respectively for $M1$ and $M2$ over [187]. The relatively high overheads for $M2$

can be attributed to the serialized writing mechanism for the segregated data. However, the benefits in on-chip storage utilization for $M2$ are definitely much important for the additional 1.8% overhead compared to $M1$. The proposed method ($M2$) has a possible demerit of including minuscule routing overhead in a local region in the vicinity of the trace buffer. However, there is no impact on global routing because the control logic for segregation works on registers (similar to the register shown in Figure 7.3) with help of the tag bits without causing any changes in routing of signals inside CUD.

In the next section, we explore the reuse of Design-for-Test (DfT) architectures for the purpose of silicon debug. As explained in the earlier chapters, scan chains (which are the most popular and widely used DfT feature in the designs) are not sufficient for silicon debug. We analyze their limitations in detail and present arguments for utilizing a DfT feature for the purpose of debug data acquisition.

7.6 Proposed Progressive Random Access Scan (PRAS)-based Debug Architecture

The usage of scan chains suffers from a major obstacle as the process of dumping out the scan values tends to be destructive in nature. This inherent drawback arises out of the operation of scan chains as the functional clock needs to be disabled so as to facilitate read-out of the scan values. One way to deal with this is to ensure that the chip can be started from the particular internally scanned out state. The scan chains have to be loaded with the given state (which was previously scanned out) to ensure the continuity in state transitions while execution of the chip for debugging. Hold-scan flip-flop offers an alternative solution to the problem of destruction of scan values during dumping. However, this technique has large area overhead as the number of flip-flops in the design gets doubled. For modern designs with large number of flip-flops, serial scan chain based debug method is not a viable option as debug time gets increased dramatically. Thus, alternative scan chain architectures need to be explored so that quick visibility of the internal states can be obtained for debug purpose.

Instead of stitching all flip-flops into a single chain, Random Access Scan (RAS) provides accessibility to the particular flip-flop for the purpose of writing the test vector bit or for the task of reading out the response bit. The scan cells are arranged in a SRAM grid like arrangement. Each scan cell is addressed by an unique combination of row address and column address resulting into routing issues. Progressive Random Access Scan (PRAS) proposed by Baik et al. [14] is a modification of the conventional RAS architecture which has less area overhead and is suited for practical implementation of large circuits. Typically, RAS architectures have the problem of routing congestion and area overhead due to flip-flop addressing mechanism. However, the authors in [14] have reported that the progressive random access scan implementation is capable of solving problems of test power, test volume and test application time at the expense of around 2-5% overhead over the overhead of serial scan (which is with in practical limits). For the purpose of non-destructive observability during post-silicon debug, due to individual accessibility of each scan cell, PRAS can be a suitable option. This is because debug time gets minimized as compared to serial scan chain based techniques as individual flip-flops can be accessed. In the light of above, progressive random access scan needs to be revisited for the purpose of silicon debug in addition to its applicability to manufacturing test.

7.6.1 Observability Enhancement Based on PRAS

We explore the *progressive access* scheme of PRAS [14] architecture for observability enhancement to measure the efficacy of the proposed signal selection methodology. Figure 7.16 presents the proposed scheme for observing flip-flop values non-destructively. Flip-flops are arranged in a SRAM-like memory grid structure. The geometry of the architecture depends on a number of factors like circuit structure and number of flip-flops in it. Flip-flops can be arranged in a rectangular fashion. However, we have chosen a square arrangement which is optimized for address calculations of scan cells as it reduces number of bits required for addressing. Thus, if F_{tot} is the number of flip-flops, there are $\sqrt{F_{tot}}$ rows and columns. An individual FF can be accessed by an unique combination

of column and row address bits.

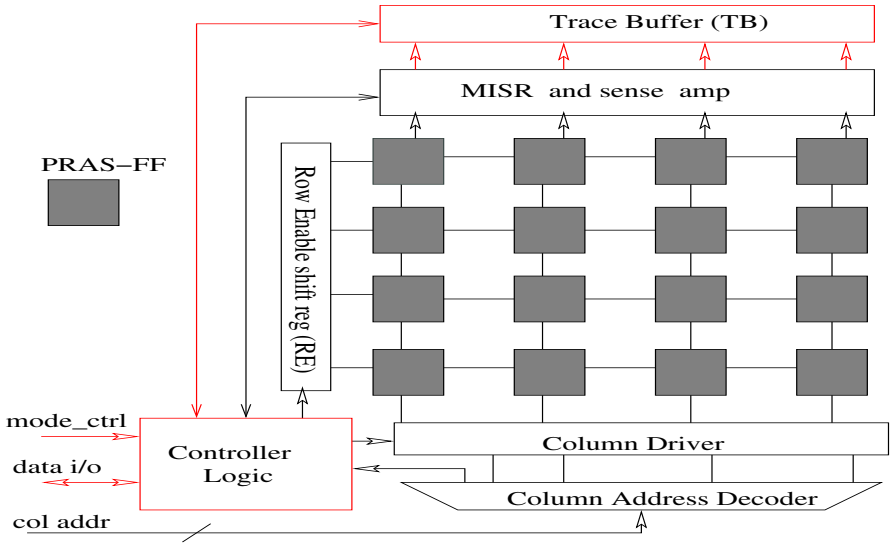

Figure 7.16: PRAS based observability enhancement scheme

For the square arrangement outlined in the Figure 7.16, each column has a particular address which is generated by column address decoder based on the column address input. The row address are generated by row address shift register which is nothing but a ring counter enabling one row at a time. This progressive access of rows avoids the necessity of a row address decoder, reducing the area overhead.

Response Compaction: Column drivers are responsible for writing into the scan cells and sense-amplifiers read the values from scan cells. As is practice with test architectures, a MISR is used to calculate the signature of scan (or response) values captured in the flip-flops through time compaction. A trace buffer (TB) is employed for storing the MISR signatures and offloading them for debug analysis. Note that the depth of TB utilized here is equal to one while the width is equal to the number of flip-flops (N) in a row, so the area overhead reduces as compared to other approaches [1, 115]. This is because techniques in [1, 115] employ trace buffer with a depth of 1024 clock cycles. Note that in Figure 7.16, outputs (Q) of MISR flip-flops are directly fed to inputs (D) of flip-flops of trace buffer.

Operation Modes: The architecture is to be operated in 3 modes: functional, test and debug mode. *Controller Logic* block is responsible for switching between the

modes. It provides necessary control signals and handling scan input/output. *mode_ctrl* pin decides the current mode of operation of the architecture. *data i/o* pin is used for offloading trace/scan output values and for receiving scan input (SI) values.

7.6.1.1 Normal (Functional) Mode Operation

In functional mode, inputs are to be applied at the primary inputs (PI) and the responses are observed at the primary outputs. There is no role of scan feature of flip-flops in this mode. Thus, they operate as normal storage elements. The operation of the architecture in this case is same as that of the functional behavior of the circuit.

7.6.1.2 Test Mode Operation

In test mode, two key operations are to be performed: a) writing the test vector bits into the scan cells and b) reading the states of scan cells. It is worth to note that in the architecture presented in Figure 7.16, reading is performed in parallel from all the flip-flops in a row while the write operation is performed on one scan cell at a time. To write into a single scan cell, column address pins provide the required column address. The *data i/o* pin provides the scan value to be written. The corresponding column enable pin is decoded and activated for writing. The row enable (RE) of that row is set to logic '1' and the column address uniquely identifies the particular flip-flop.

Test application: The MISR is activated and the row enable signal is shifted after all the write operations for the currently selected row are completed. The test application is quite similar to the approach of Baik et al. [14] with the only difference that in the current framework, test responses can be stored first in the trace buffer and then offloaded for analysis.

7.6.1.3 Debug Mode Operation

In debug mode, the inputs are applied at the primary inputs and the responses captured in the flip-flops are to be read-out via *data i/o* pin. As explained in Section 7.6.1.2, a read

operation is performed for all scan cells in a selected row. The sequence of operations in debug mode is explained with the help of a flow-chart in Section 7.6.3.

One key difference between test mode and debug mode is that the clock signal operates in a functional manner in the latter. *Thus, the circuit functionality is not stopped for writing the responses into flip-flops and reading the responses from them.* Depending on the debug requirement, the contents in row enable shift register can be altered to select the desired row for observing the corresponding scan cells (flip-flops).

7.6.2 Scan FF Operation in Proposed Scheme

There is a significant difference in the operation of scan flip-flop used in PRAS and the scan flip-flop utilized in serial scan methodology. While the latter requires functioning in a chain for test purposes, the former does not require so and is individually accessible through the addressing mechanism. Figure 7.17 is a schematic representation of PRAS FF. Its operation in different modes (*test-write*, *test-read*, *debug* and *normal*) are depicted in Table 7.9. Note that this FF does not incur much area overhead as compared to the conventional scan FF according to the results reported in [14].

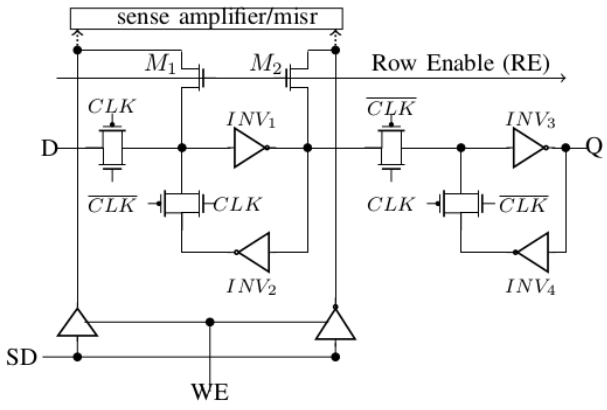

Figure 7.17: PRAS FF [14] for observability enhancement

Table 7.9: Operation of PRAS FF for observability enhancement scheme

operation mode	*test-write*	*test-read*	*debug*	*normal*
Clock(CLK)	1	1	functional	functional
Write Enable(WE)	1	0	0	0
Row Enable(RE)	1	1	1	1
Column driver	activated	deactivated	deactivated	deactivated
Sense amplifier	deactivated	activated	activated	activated

7.6.3 Methodology for Observing Internal States

The sequence of events to be followed during debug mode to enable observing internal states of the circuit is represented in Figure 7.18. The proposed observability enhancement architecture provides reconfigurability which is missing in serial scan chain based techniques. Also, a fixed visibility like restoration based trace signal selection methods [1, 102, 115] may not be useful under different kind of functional errors or bug scenarios. The proposed approach allows selective visibility of rows of flip-flops which is fully reconfigurable in nature without any usage of multiplexers [118]. As noted in Section 7.6.1, to avoid overwriting of data, only one row can be read in one clock cycle. As is shown in Figure 7.18 the same row can be observed for p cycles or reading of few selected rows may be distributed among p cycles. Note that as per the architecture of Figure 7.16, p equals F_{tot} here. If the aliasing of the responses of the buggy designs with the golden response exceeds tolerable limits, then MISR needs to be permanently deactivated.

As an alternate to second last step of the above methodology (Fig. 7.18), TB can be configured for data capture from MISR at occurrence of particular trigger events through Embedded Logic Analyzer (ELA) [1, 102, 115]. Thus, MISR signatures can be sent off-line for error localization and other analysis. In the architecture of Figure 7.16, if the row enable shift register progresses serially, first row can be accessed in first clock cycle, second row in second cycle and so on. The arrangement of flip-flops in rows is thus an important factor for quick read-out of flip-flop values. This hints at the need of assignment of priority to flip-flops for debugging.

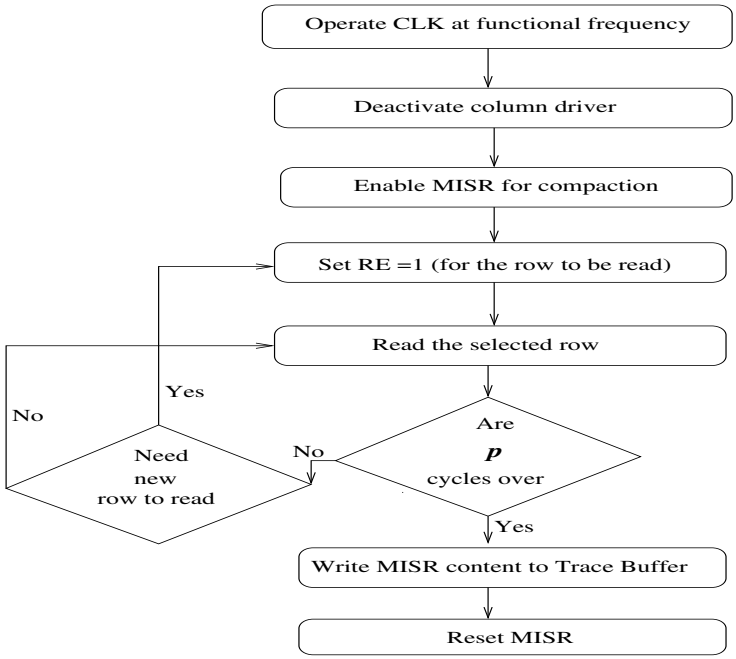

Figure 7.18: Flow chart for *debug* mode operation

7.6.4 Arrangement of Flip-flops in Debug Architecture

To establish the importance of flip-flops for their arrangement in the upper rows of the architecture presented in Figure 7.16, signal ranking methodology in Algorithm 9 of Chapter 5 can be utilized. The signal selection methodology for ranking of candidate trace signals through topological analysis of the circuit (gate-level design description) is based on [116]. During the logical synthesis of the design, the grouping of flip-flops in each of the rows in Figure 7.16 can be based on the rank of signals obtained by the signal selection methodology. The arrangement of rows based on Algorithm 9 (of Chapter 5) can aim at reducing the error detection latency. This is because read-out of flip-flop values from lower rows (in Figure 7.16) requires large number of clock cycles. The highly ranked (i.e., important) r (which is equal to $rowlength$) signals are placed in the first row, the second highly ranked r FFs in the second row and so on. The following procedure ($FFArrange$) outlines the arrangement of FFs of the design given a ranked arrangement ($RankedSignals$) of all signals. Note that the prioritized arrangement of flip-flops in the architecture of Figure 7.16 can have impact on the layout process. The routing considerations in a typical arrangement of flip-flops in RAS fashion (as in [14])

Algorithm 20: *FFArrange*

Input: *RankedSignals*, r, F_{tot}
Output: *Arrangement*

```
1  Candidates ← RankedSignals;
2  q=0;
3  rownum=1;
4  while q < F_tot do
5  |   put r Candidates in rownum^th row;
6  |   q = q + r;
7  |   rownum = rownum + 1;
8  |   Remove r FFs from Candidates;
9  end
10 Arrangement ← position of all flip-flops;
```

may have conflicting requirements as suggested by Algorithm 20. A joint optimization of error detection priority and layout ease capable of providing better selection is left as a future exercise.

7.7 Experimental Result on PRAS-based Debug

7.7.1 Experimental Formulation

The proposed observability enhancement scheme helps to obtain debug data quicker than serial scan as compared to the scenario when the serial scan chain DfT solution is used for reading and offloading scan values. The number of cycles (*cycle*) needed to offload and read n rows consecutively is given by the following relation:

$$cycle = r * (n + 1) \tag{7.10}$$

In the above equation, r is *rowlength* of one row of flip-flops in Figure 7.16 (assuming a square arrangement). Note that the maximum value of n is give by $\sqrt{F_{tot}}$, F_{tot} being complete serial scan chain length. As we increase number of rows to be observed, we move towards F_{tot}. Serial scan based methods require scanning out states of all the flip-flops even for observing a few of them. However, with the proposed architecture,

selective reading of rows can be performed quickly for the validation purpose. Equation 7.10 can also be applied in case the same row is read r times. Since serial scan chain-based methodology requires at least F_{tot} clock cycles for offloading its contents, in order to compare the number of clock cycles required with the presented methodology as compared to serial scan chain-based methods, we define a ratio, *reduction* based on Equation 7.10.

$$reduction = \frac{F_{tot} - \{1 + n * (r + 1)\}}{F_{tot}} \tag{7.11}$$

Substituting $n = \sqrt{F}_{tot}$ and simplifying,

$$reduction \approx 1 - \frac{r + 1}{\sqrt{F}_{tot}} \tag{7.12}$$

The metric, *reduction* can have a minimum value of zero. As this metric approaches zero, the *cycle* by PRAS-based mechanism achieves minimum value. So, as $\sqrt{F}_{tot}$ decreases, *reduction* metric tends to obtain lower values.

As far as applicability of PRAS to large designs are concerned, it serves as an attractive alternative considering the benefits it offers for manufacturing test as well as debug. Although implementation of PRAS requires certain overheads regarding layout, the original proposal in [14] showed that these overheads are tolerable. Recent proposals in [195, 196] provide results on implementation of Joint Scan which is a hybrid debug architecture of serial and random-access scan.

7.7.2 Experimental Metrics and Results

7.7.2.1 Ranked Arrangement of Flip-flops

To measure the effectiveness of the ranking of flip-flops from viewpoint of error localization, a metric is proposed for the proposed scheme with the arrangement in Algorithm 20 utilizing the ranking methodology (Algorithm 9 of Chapter 5). It is based on comparing the MISR signatures (MS) and golden signatures (GS). Note that MISR signatures

which are referred to here, are essentially those which are stored in the trace buffer. The following equation defines the metric:

$$\phi(n) = \frac{erroneous\ MISR\ signature\ bits\ for\ n\ rows}{total\ MISR\ signature\ bits\ for\ n\ rows} \tag{7.13}$$

In contrast to "Restoration Ratio" [1, 102, 115], the metric $\phi(n)$ serves as a better metric to evaluate the effectiveness of the data collected through trace buffer since restoration merely considers the enlargement of the debug data without consider the effectiveness of the stored information [55]. We performed 100 iterations of gate replacement error injections and the obtained results are shown in Figure 7.19. As expected, we observed an increase in $\phi(n)$ values when the number of rows is increased by one or two. For one circuit, with increase in value of n, the metric ϕ decreases. This indicates the need of dynamic trace signal selection in the silicon debug scheme.

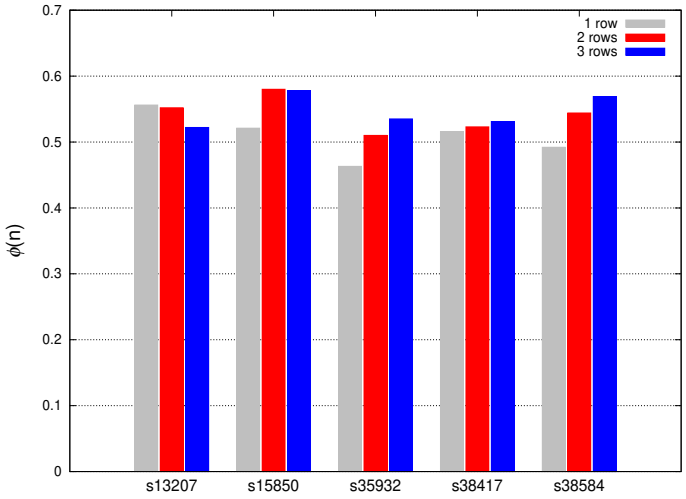

Figure 7.19: Change in $\phi(n)$ with increase in no. of rows

Trace buffer techniques [1, 102, 115] can not achieve the feature of dynamic signal selection and require dedicated multiplexers to achieve this. However, with proposed PRAS based observability enhancement scheme, this can be achieved easily by loading the appropriate bits in "row enable shift register".

7.7.2.2 Trivial Arrangement of Flip-flops

To avoid overheads arising out of ranked arrangement of flip-flops, we analyzed a normal arrangement (i.e., without any ranking of signals of the design) which is suitable to the synthesis process. We performed 100 iterations of wire exchange error scenario and then computed the following metric ($n_2 > n_1$):

$$inc = \frac{erroneous\ MISR\ signature\ bits\ for\ n_2\ rows}{erroneous\ MISR\ signature\ bits\ for\ n_1\ rows} \tag{7.14}$$

The results for computation of the above metric is shown in Figure 7.20. Note that in this scheme, we have total flip-flops observed can be given by n_2*$rowlength$ and n_1*$rowlength$. These flip-flops are significantly higher than that of typical trace buffer width (which is 32). As expected, there is an increment in inc metric when we move from

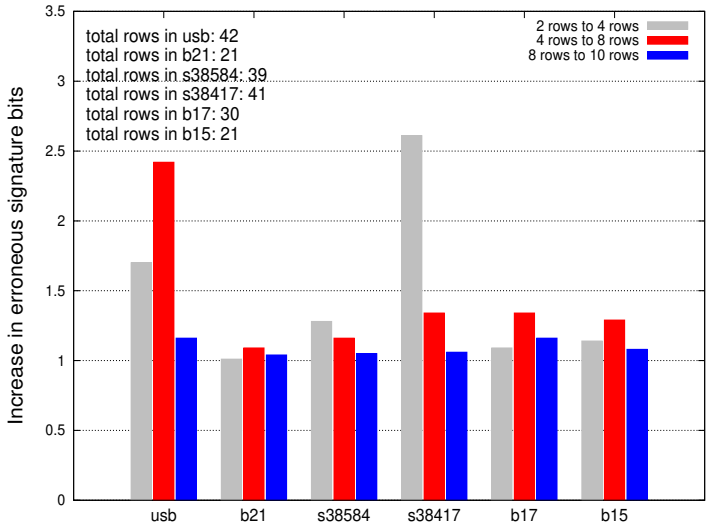

Figure 7.20: Change in inc values with increase in no. of rows

n_1 to n_2 rows in the architecture in Figure 7.16. Therefore, even with normal arrangement of flip-flops, larger number of flip-flops can be observed. Observing larger number of flip-flops can help in the silicon debug process. However, exact error localization with the help of signatures which are dumped off-line, is not trivial and therefore require considerations of other factors like length of MISR (deciding the amount of compression).

7.8 Discussions on Multi-session Silicon Debug

7.8.1 Reproducibility of Failures in Post-silicon Environment

Non-reproducible errors are extremely difficult to debug [159]. However, with mechanisms for the deterministic replay of these errors (such as those proposed in [197, 198]) the errors can be reproduced to a certain extent allowing further debug (i.e., in the second session in the proposed methodology). Dedicated mechanisms such as error tracing buses [199] or some other hardware-assisted mechanisms [200] can allow the debug data acquisition in the second session.

7.8.2 Availability of Golden Responses of Designs

As mentioned earlier, lack of golden response is one of the major impediments of quick silicon debug [33, 159, 201]. However, for low-level debug, based on expected behavior at an intermediate level, the expected signal values at RTL model can be computed. This intermediate level behavior can be obtained concerning a broad range of user applications targeting the particular design. In this work, we have not put any condition on the internal signals being traced. For instance, if a subset of the flip-flops of the design is traced, it can be possible to obtain the correct flip-flops from a higher level reference model which has the same number of sequential elements as that of the RTL.

7.8.3 Relevance with Respect to Similar Work reported in Literature

With the proposed on-chip compression scheme and other features, the debug experiments can be effectively carried out in a single session instead of two sessions. This requires additional on-chip storage of the golden data signatures to facilitate comparison with CUD signatures and suspect clock cycle/cycles (Cy) determination. The authors in [191][189] utilize DRAM for storing the golden data signatures at the cost of additional area overhead. For a multi-core design, the proposed scheme can be applied for

each core. The scheme of sharing trace buffer for multiple identical cores can be used in conjunction with the proposed approach in [188] and [185]. The proposed scheme can also be readily utilized in level-based debugging[123] to achieve significant reduction in the CUD debug data to be compared with the corresponding golden data during each level. Such an iterative debug scheme [123, 185, 191] typically employs segmented trace buffer and multiple debug sessions at each debug level. The TB segmentation proceeds till the last debug level is reached.

Consider that the CUD execution length is $L1$ (i.e., initial observation window), the number of samples (ns) per signature at the first debug level is given by $\frac{L1}{TBd}$. The parameter ns decreases at each successive debug level. With the proposed tag bit generation and the suspect determination procedure, ns can be increased from second debug level onwards. The debug time can also be brought down when the proposed technique is incorporated with in the iterative methodologies of [123, 185, 191]. For achieving on-chip compression, a setup comprising of hierarchical MISRs can also be adopted. However, with such a setup aliasing of debug signatures may increase to a significant extent. By utilizing on-chip summarizers [160], ineffective debug data can be removed and only selective data needs to be stored into TB. This technique significantly reduces the number of stalls during the debug data gathering. However, the success of these summarizers is highly dependent on the design functionality. For regular designs like processors, incorporation of effective summarizers can be achieved as selection of a certain number of important internal signals turn out to be easier compared to general sequential designs [33]. We believe that with the proposed on-chip data compaction and subsequent segregation, a larger number of trace signals can be traced in second session (S_2) as the actual debug data to be stored (corresponding to a fixed observation window) is significantly reduced. Thus, effective utilization of the on-chip storage and data segregation can together provide some sort of summarizing feature in the second session of multi session-based debug methodology.

7.9 Conclusion

During post-silicon validation, data compaction techniques are typically employed to efficiently utilize the on-chip storage during debug data acquisition. We proposed several improvements in compaction-based selective debug data capturing. Using a two session-based debugging methodology, we achieved fine-grained error localization with significant expansion in temporal visibility expansion. With the proposed scheme, the amount of on-chip storage required for tag bits (which act as markers for the selective data capture) is also reduced. We revisited the progressive random access scan architecture for realizing an observability scheme with the motive of integrating testing and post-silicon debug in a single infrastructure. The PRAS scan cell allows non-intrusive dumping of scan values which is a key requirement for debug purposes. A methodology has been proposed to extend this architecture for reconfigurable observability enhancement of internal signals. Flip-flops of the design can be ranked for their arrangement in the proposed observability enhancement architecture so as to obtain effective debug data in lesser time.

– ∗ – ∗ –

Chapter 8

Conclusion and Future Scope

Demand for higher levels of integration coupled with the shrinking time-to-market window has drastically increased the chances of errors escaping to the first released silicon [21], [151],[159]. Accurate modeling of physical effects such as process variations, thermal effects and signal integrity is usually very difficult during pre-silicon design verification resulting in occurrence of electrical errors. Thus, post-silicon validation and debug form an essential step in the design implementation cycle of modern integrated circuits. Post-silicon validation is performed at-speed in the gigahertz range in contrast to pre-silicon validation which runs in tens of hertz range. This characteristic enables application of a very large number of input sequences at this stage which can potentially capture the target behavior of the chip. However, due to the limited controllability and observability, this process becomes very time-consuming and challenging. This thesis presented effective techniques to solve some of these issues. Self-checking methods were utilized to validate and localize design and electrical errors in complex systems. Heuristics were developed to tackle the restricted visibility problem and learning methods were employed to obtain success in automatic gate-level error localization. Compression feature was extensively utilized for reducing the quantity of debug data and the possibilities of reusing an alternative DfT scheme for silicon debug was also explored. The detailed summary of the thesis and conclusions are described next.

8.1 Thesis Summary and Conclusions

- Chapter 3 described the proposed methodology for validation of cache coherence mechanism in multi-processors. An on-chip signal logging method is proposed which helps in bug detection in case of design errors and soft-errors arising out of reliability issues. The logged contents can then be further dumped off-line for fine-grained bug localization. The proposed methodology utilizes cache coherence protocol specifications to obtain the signal states of coherence transactions and the detector module flags an error once a mismatch is found between observed signal states and correct signal states. The proposed logging mechanism decreases the error detection latency at minimal area and power overheads. Experiments on a four core multiprocessor having a 7-stage MIPS pipeline implementing the widely utilized directory-based MESI protocol indicate that the proposed methodology succeeds in detecting design errors. Analysis of soft errors has also been performed and shorter error detection latency is achieved compared to a previously proposed technique in the literature.

- Chapter 4 presented the proposed methodology of satisfiability (SAT)-based error localization of bit-flips (which can model electrical errors). This chapter revisited methodologies to debug electrical errors through satisfiability(SAT) solving under a limited visibility environment. We proposed various SAT formulations and analyze their efficacy in error localization for a variety of benchmark circuits. The selection of debugging instrumentation is an important issue in post-silicon validation. We analyze different graph-based signal tracing techniques and propose a methodology that utilizes clustering of the nodes of the circuit graph. We aimed at minimizing the overhead associated with signal tracing while maintaining the error localization efficacy. We address scalability concerns in SAT solving through partitioning of large error traces. We analyzed localization results on two different error models (bit-flip and stuck-at) and evaluate its efficiency through a set of different metrics.

- Chapter 5 presented the proposed methodology for effective trace signal selection.

Since the selection of trace signals is done at the design stage itself, it becomes a deciding factor for the success of trace buffer-based DfD (Design for Debug) schemes. For processor-based systems, some of the general-purpose registers or architectural event registers and performance counters can be used as trace signals, which can capture functionally important events [202]. However, trace signal selection for digital blocks within a complex system-on-chip is very difficult and needs an algorithmic solution [34, 115]. The proposed methodology of trace signal selection aimed at detecting design errors. As a better alternative to state restoration based signal selection, two heuristics were proposed for signal selection which can account for the error propagation ability of the traced flip-flops. Topology-based selection achieved lower error detection latency compared to the signal selection techniques proposed in the literature.

- Chapter 6 elaborated on the proposed method of complete internal visibility expansion and the subsequent error localization with this approach. Using Nearest neighbors algorithm, some neighbors are identified and then utilizing majority voting, unknown (after the application of restoration) signal states are predicted albeit with variable accuracy. Using the expanded internal visibility, the design error localization succeeds to a higher extent than the mere application of signal tracing and restoration. A methodology for error localization to a smaller region of the circuit was proposed via building a design response model that works on the principle of classifying the smaller region into buggy or non-buggy. Using a training analysis, the model was developed after the training for a large number of iterations which was used to test an erroneous design to obtain probable suspects with a ranking scheme. With the design response model, we were able to achieve design error localization without any false positives during the debug process.

- Chapter 7 explained the proposed method of on-chip compression of debug data in a multi-session silicon debug procedure. Data Compaction techniques are typically employed to efficiently utilize on-chip storage. We proposed several improvements

in 2D-compaction-based selective debug data capturing. The proposed debug architecture consists of on-chip MISR, two cycling registers and trace buffer for on-chip storage. Using a 2-session based debugging methodology, we achieved fine-grained error localization with significant expansion in temporal visibility expansion. With the proposed scheme, the amount of on-chip storage required for tag bits (which act as markers for selective data capture) is reduced. This chapter also presents the reuse scheme for random access scan (RAS) in silicon debug. With the reuse of progressive random scan design-for-testability scheme, we were able to achieve reconfigurability in internal visibility of the design. This debug scheme has the potential of developing into an integrated solution of manufacturing testing and silicon debug.

8.2 Future Scope

8.2.1 Unified Validation of Memory Consistency and Coherence in Complex Processor Designs

Modern processors employ a large number of architectural features such as speculative execution and reordering of memory operations to achieve higher performance. Such memory ordering operations are governed by Memory Consistency Models (MCM) which act as an agreement between the programmer and the hardware designer. However, due to a large number of valid execution results, simulation-based verification of MCM implementation is a difficult task that necessitates its validation after manufacturing. In the first silicon, due to restricted observability, the order of execution of memory operations must be logged on-chip. These logged contents can then be dumped off-line for further constraint-graph based analysis [77, 78] and checking for any MCM violation(s). The validation of both consistency and coherence can be done in a unified manner as some of the logged contents are similar in both the approaches. The proposed CCP validation technique needs to be investigated for other cache coherence protocols

such as MOSI, MOESI etc. and shared unified memory access (UMA) configuration. Additionally, the proposed methodology can be improved by incorporating mechanisms for adjusting system performance degradation because of validation tasks.

8.2.2 Enhancements of SAT-based Error Localization

We identified several avenues to improve the SAT-based error localization for some of the circuits: (a) assigning proper primary inputs from the design knowledge (i.e., hierarchical analysis of RTL), (b) obtaining initial state of the design netlist (i.e., signal states of all flip-flops corresponding to a cycle before the beginning of error trace) from design simulation to assist in SAT solving, and (c) setting intermediate controls signals to their appropriate values to obtain proper UNSAT cores(s). We believe that these enhancements would play a more important role in the error localization for circuits of opencore and ITC'99 suites than a bit-flip detection driven trace signal selection (like the one proposed in [34]). This is primarily because of our observation that attempting the SAT-based methodology with a wide pool of trace signal selections (clustering-based, [9, 23, 51], [1, 141] led to minuscule variations in the exact error localization metrics for these circuits. With the above enhancements, stuck-at error localization would improve significantly is compared to the bit-flip error localization because the former type of error shows more affinity towards sensitization by inputs. The analysis of the proposed methodology for other kinds of error models (such as stuck-open or bridging effects) is also left as a future exercise. The investigation of the impact of false paths on the clustering-based signal selection is also left as a future exercise.

8.2.3 Targeted Trace Signal Selection

The proposed work can be extended in many directions. First, an automatic signal selection algorithm (based on learning techniques) can be worked out to obtain the most profitable combination of partitioning parameters, a, b and c. The choice of these parameters can also be formulated as an ILP (integer linear programming) problem for

concurrent optimization of all the three signal selection preferences. Second, the efficacy of the proposed heuristic needs to be investigated for other kinds of design error models and electrical bugs (which can be modeled as bit-flips/stuck-at) and blocking bugs [203]. Adapting the proposed methodology in case of bugs with non-repeatable behavior would require significant improvements in maintaining the state continuity across multiple debug sessions. Third, a better error transmission heuristic is needed to properly deal with re-convergent nodes and paths. Improvement in the accuracy of the proposed heuristics would enhance the error localization by a significant margin. Another method of improvement in accuracy can be consideration of false paths for the path based trace signal selection heuristic. On similar lines, we can even account for sequential paths (e.g. 2-cycle sequential) instead of combinational ones to capture controllability over several cycles and analyze the error detection/localization. As a future research direction, several modifications can be applied to the topology-based selection like assignment of priority to nodes of the S-graph. This priority can be linked with error-proneness of few portions of design based on pre-silicon verification information. The priority assignment is intended to ensure selection of trace signals regardless of their topological score.

Assertions generated during pre-silicon verification of the RTL design can be reused for the purpose of selecting the trace signals. Based on the mining of good quality assertions from design description and its simulation, effective trace signals can be selected based on the frequency of the occurrence of particular signal variables in the complete list of assertions [63]. This can enable us to perform a signal selection at the RTL stage itself instead of the selection with design netlist. The selection of trace signals can also assist in calculating the coverage in post-silicon environment as based on traced and restored data, we can check for the coverage of the assertions from pre-silicon verification stage. However, the major challenge in applying this technique for a large and complex design is to obtain a correct set of assertions. For designs in development stage, such assertions suite is very rare to obtain. We have proposed a preliminary approach [63] in this direction based on the assertions generated by the methodology described in [204–206]. A similar technique has been also proposed in [207] targeting coverage estimation

by assertion decomposition.

8.2.4 Automatic Error Localization for Wide Range of Error Models

We believe that localizing electrical and design errors could be possible using regression analysis and relating with observed system-level variables (i.e., obtained traces via recording/visibility mechanisms). Hierarchical dependability analysis (from RTL description) can also prove to be very useful in this regard. With the help of sufficient design knowledge, the requirement of golden responses for debug can be eliminated. This can be easily facilitated by data mining techniques as in [171]. Learning-based approaches have been used for trojan detection at gate-level abstraction without the requirement of golden responses [208–212]. The proposed methodology of visibility expansion can be improved by various means like employing a suitable technique for training a large number of features during the training phase. Additionally, Bayesian analysis (similar to the methodology adopted in [213]) can be utilized for enhancing the accuracy of neighbor-based maximal internal visibility expansion. Instead of a plain majority voting, weight assignments to the traced and restored signal states can also assist in improving the correctness in signal state prediction. Additionally, a proper theoretic formulation is needed to obtain the optimum number of neighbors in M_{nbr} to achieve the highest accuracy in unknown signal state prediction. With a highly accurate completely expanded internal visibility, the offloaded debug data can be utilized for property checking without the incorporation of on-chip checkers. Unsupervised machine learning algorithms like k-means can also be attempted for the task of predicting unknown signal states.

Increasing number of features during training can help localize to the particular erroneous gate instead of a smaller region of the circuit. Identifying such training features is expected to play an important role in error localization. The definition of the objective function while building the design response model (M) may also be refined to achieve a better ranking of suspects during the error localization process. Different types of learning algorithms have more potential to be useful for the error localization process in

the way they have been applied for yield and defect analysis in manufacturing testing [173–175].

8.2.5 Improving Debug Architectures with Compression

It would be interesting to extend the proposed methodology for non-reproducible errors as the same debug data is not available in the second session. The dimensions of on-chip MISR can be investigated for achieving better utilization of the trace buffer with the amount of on-chip compression. The selective debug data capture can be further improved with the incorporation of periodic monitoring or on-chip assertion-based trigger mechanism. Other directions for future work are devising efficient compression mechanisms along with self checking-based trigger schemes for deciding the temporal duration of storing internals signals in the trace buffers. Investigating the arrangement of flip-flops with a joint purpose of error localization and routing overhead minimization is needed for the PRAS-based debug architecture. In addition to that, evaluation of the aliasing due to MISR and its impact on the debug process would be an interesting topic. Despite the overhead issues with PRAS, the ease in post-silicon validation and debug with the proposed PRAS-based observability enhancement architecture can widen its acceptability. The mapping of failing MISR signatures (from the offloaded trace data) with the individual flip-flops (signals) can also be considered as a future extension of the proposed PRAS-based debug methodology.

8.2.6 Validating Diversified Processor Components

The methodologies presented in the thesis can be be utilized in conjunction for validating the different components available in modern processors. For example- coherence mechanism can be validated in on-chip fashion through the technique described in Chapter 3. Logic components can be validated for electrical and design errors through the techniques described in Chapter 4 and 6 respectively. Nevertheless, obtaining a methodology that encompasses the effective validation of all components in a synergistic way is

an important direction in which further work needs to be carried out. Specifically, we believe that by utilizing some of the techniques mentioned in the literature such as [69] or that of *Reversi* [65], all the components in modern processors can be validated and useful error trace(s) can be obtained for off-line debug analysis.

8.2.7 Interplay of Hardware Security and Debug Requirements

The contents stored in trace buffers can be reused to obtain sensitive information about the design [214, 215]. Furthermore, trace buffers can be reused for performance enhancement [216] as victim caches. This necessitates analysis of different attack strategies possible through trace buffers. One of the possible research directions is to constrain the signal tracing mechanism (or, the trace signal selection) such that security specifications are not compromised. Another research direction could be the investigation of possibilities of on-chip encryption of the traced data so that their off-line analysis does not reveal any secrets to unauthorized entities.

– ∗ – ∗ –

Appendix A

Details of Experimental Setup

Apart from the benchmark circuits available from ISCAS'89 and ITC'99 suites, we obtained some benchmark circuits after synthesizing the RTL obtained from Opencores.org [143]. These circuits are then translated from the synthesized description into .bench format (a sample of which is shown in Figure A.1 where ..... indicates similar entries in the succeeding lines). Since the circuits from ISCAS'89 and ITC'99 have been widely utilized in research in several decades, we do not describe them in detail. The steps to obtain .bench description of circuits from [143] are below:

- Synthesize the original RTL descriptions (in verilog) using Synopsys Design Compiler (DC) tool.
- Rename the synthesized gates into respective trivial gates (AND, OR, NAND, NOR, NAND, XOR, XNOR, NOT) and similarly for flip-flops.
- Arrange the netlist into .bench format (i.e., five lines in the top, listing of inputs, listing of outputs, space after each such listing etc.,) as per the formats described/available at [217].

The functionalities of the benchmark circuits from [143] are described as below:

1. **p16c5x**: single cycle core for the emulation of PIC16C5x microcomputers

```
# 11 inputs
# 20 outputs
# 800 D-type flipflops
# 3500 inverters
# 6100 gates (1700 ANDs + 1900 NANDs + 400 ORs + 2100 NORs)

INPUT(C_7)
........

OUTPUT(W)
..........

Y_1 = DFF(II6)
Y_2 = DFF(II5)
..........
.........

II6 = NOT(II104)
II50 = NOT(II92)
...............
...............

II127_1 = AND(II41, Y_3, II109)
II127_2 = AND(II96, II113, Y_4)
.................

II135_1 = OR(II43, II104)
II135_2 = OR(Y_2, II100)
.......................
.......................

II5 = NAND(II135_1, II135_2)
II92 = NAND(Y_2, Y_1)
.........................
```

Figure A.1: Sample .bench format

2. **usb**: a Serial Interface Engine (SIE) typically utilized for communication between a processor and it's peripherals.

3. **softusb**: communication controller for interfacing

4. **mips**: 5-stage pipeline MIPS 32-bit processor.

To simulate these .bench descriptions (along with the .bench corresponding to ITC'99 and ISCAS'89 circuits), we developed our own simulator in C language. The same simulator was extended to perform signal restoration on the traced data. By minor modification in this simulator, we performed the stuck-at and bit-flip simulation tasks of Chapter 4. We performed most of the scripting tasks in Python and utilized many shell (bash) functions. For all the other tools like Design Compiler, IC Compiler, RTL Complier etc., appropriate .tcl scripts were written and the respective tasks were done at servers maintained by VLSI LAB, IIT Bombay.

Bibliography

[1] K. Basu and P. Mishra. Rats: Restoration-aware trace signal selection for post-silicon validation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(4):605–613, April 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2012.2192457.

[2] Daniel J. Sorin, Mark D. Hill, and David A. Wood. *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers, 1st edition, 2011. ISBN 1608455645, 9781608455645.

[3] R. Rodrigues, I. Koren, and S. Kundu. A mechanism to verify cache coherence transactions in multicore systems. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 211–216, Oct 2012. doi: 10.1109/DFT.2012.6378226.

[4] Harry Foster. 2014 wilson research group functional verification study. https://blogs.mentor.com/verificationhorizons/blog/2015/01/21, 2014.

[5] D. D. Josephson. The manic depression of microprocessor debug. In *Test Conference, 2002. Proceedings. International*, pages 657–663, 2002. doi: 10.1109/TEST.2002.1041817.

[6] A. W. DeOrio. Correct communication in multi-core processors. PhD Thesis, 2012.

[7] D. Lee, O. Matthews, and V. Bertacco. Low-overhead microarchitectural patching for multicore memory subsystems. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 17–25, Oct 2018.

[8] K. Constantinides, O. Mutlu, and T. Austin. Online design bug detection: Rtl analysis, flexible mechanisms, and evaluation. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 282–293, Nov 2008. doi: 10.1109/MICRO.2008.4771798.

[9] K. Rahmani, P. Mishra, and S. Ray. Efficient trace signal selection using augmentation and ilp techniques. In *Fifteenth International Symposium on Quality Electronic Design*, pages 148–155, March 2014. doi: 10.1109/ISQED.2014.6783318.

[10] http://15418.courses.cs.cmu.edu/tsinghua2017/lecture/coherence.

[11] John L Hennessy, David A Patterson, and Krste Asanović. Computer architecture: a quantitative approach, 2012.

[12] Nima Honarmand. Cse 610 - parallel computer architectures lecture notes. https://compas.cs.stonybrook.edu/ñhonarmand/courses/fa15/cse610/index.html, Fall 2015.

[13] M.A. Kinsy, M. Pellauer, and S. Devadas. Heracles: A tool for fast rtl-based design space exploration of multicore processors. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, FPGA '13, New York, NY, USA, 2013. ACM.

[14] D. Baik and K.K. Saluja. Progressive random access scan: A simultaneous solution to test power, test data volume and test time. In *Proc. of the International Test Conference*, pages 1–10, 2005.

[15] P. Mishra, R. Morad, A. Ziv, and S. Ray. Post-silicon validation in the soc era: A tutorial introduction. *IEEE Design Test*, 34(3):68–92, June 2017. ISSN 2168-2356. doi: 10.1109/MDAT.2017.2691348.

[16] K. Rahmani and P. Mishra. Feature-based signal selection for post-silicon debug using machine learning. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017. doi: 10.1109/TETC.2017.2787610.

[17] Eshan Singh, Clark W. Barrett, and Subhasish Mitra. E-QED: electrical bug localization during post-silicon validation enabled by quick error detection and formal methods. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, pages 104–125, 2017.

[18] D. Lee and V. Bertacco. Mtracecheck: Validating non-deterministic behavior of memory consistency models in post-silicon validation. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 201–213, June 2017.

[19] A. Adir, S. Copty, S. Landa, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann. A unified methodology for pre-silicon verification and post-silicon validation. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011. doi: 10.1109/DATE.2011.5763252.

[20] V. Bertacco. Post-silicon debugging for multi-core designs. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 255–258, Jan 2010. doi: 10.1109/ASPDAC.2010.5419885.

[21] Subhasish Mitra, Sanjit A. Seshia, and Nicola Nicolici. Post-silicon validation opportunities, challenges and recent advances. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 12–17, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0002-5.

[22] K. Rahmani, S. Proch, and P. Mishra. Efficient selection of trace and scan signals for post-silicon debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(1):313–323, Jan 2016. ISSN 1063-8210. doi: 10.1109/TVLSI.2015.2396083.

[23] D. Pal, S. Ma, and S. Vasudevan. Emphasizing functional relevance over state restoration in post-silicon signal tracing. *IEEE Transactions on Computer-Aided*

Design of Integrated Circuits and Systems, pages 1–1, 2018. ISSN 0278-0070. doi: 10.1109/TCAD.2018.2887047.

[24] K. Iwata, A. M. Gharehbaghi, M. B. Tahoori, and M. Fujita. Post silicon debugging of electrical bugs using trace buffers. In *2017 IEEE 26th Asian Test Symposium (ATS)*, pages 189–194, Nov 2017. doi: 10.1109/ATS.2017.44.

[25] M. Dehbashi and G. Fey. Automated debugging from pre-silicon to post-silicon. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2012 IEEE 15th International Symposium on*, pages 324–329, April 2012. doi: 10.1109/DDECS.2012.6219082.

[26] Kai hui Chang, I. L. Markov, and V. Bertacco. Automating post-silicon debugging and repair. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 91–98, Nov 2007. doi: 10.1109/ICCAD.2007.4397249.

[27] B. Vermeulen and S. K. Goel. Design for debug: catching design errors in digital chips. *IEEE Design Test of Computers*, 19(3):35–43, May 2002. ISSN 0740-7475. doi: 10.1109/MDT.2002.1003792.

[28] M. Gao, P. Lisherness, K. T. Cheng, and J. J. Liou. On error modeling of electrical bugs for post-silicon timing validation. In *17th Asia and South Pacific Design Automation Conference*, pages 701–706, Jan 2012. doi: 10.1109/ASPDAC.2012.6165046.

[29] M. Gao, P. Lisherness, and K. T. Cheng. Post-silicon bug detection for variation induced electrical bugs. In *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pages 273–278, Jan 2011. doi: 10.1109/ASPDAC.2011.5722197.

[30] C. C. Yen, T. Lin, H. Lin, K. Yang, T. Liu, and Y. C. Hsu. A general failure candidate ranking framework for silicon debug. In *26th IEEE VLSI Test Symposium (vts 2008)*, pages 352–358, April 2008. doi: 10.1109/VTS.2008.60.

[31] H. Mangassarian, A. Veneris, D. E. Smith, and S. Safarpour. Debugging with dominance: On-the-fly rtl debug solution implications. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 587–594, Nov 2011. doi: 10.1109/ICCAD.2011.6105390.

[32] B. Kumar, M. Fujita, and V. Singh. A methodology for sat-based electrical error debugging during post-silicon validation. In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pages 389–394, Jan 2019.

[33] B. Kumar, K. Basu, M. Fujita, and V. Singh. Post-silicon gate-level error localization with effective amp;amp; combined trace signal selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2018. ISSN 0278-0070. doi: 10.1109/TCAD.2018.2883899.

[34] A. Vali and N. Nicolici. Bit-flip detection-driven selection of trace signals. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5): 1076–1089, May 2018. ISSN 0278-0070. doi: 10.1109/TCAD.2017.2729458.

[35] J. S. Yang and N. A. Touba. Enhancing silicon debug via periodic monitoring. In *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pages 125–133, Oct 2008. doi: 10.1109/DFT.2008.57.

[36] S. Sarangi, S. Narayanasamy, B. Carneal, A. Tiwari, B. Calder, and J. Torrellas. Patching processor design errors with programmable hardware. *IEEE Micro*, 27 (1):12–25, Jan 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.19.

[37] T.R. Halfhill. The truth behind the pentium bug. http://www.byte.com/art/9503/sec13/art1.htm, March 1995.

[38] B Bentley. Validating a modern microprocessor. In *Proceedings of the 17th International Conference on Computer Aided Verification*, CAV '05, pages 2–4, 2005.

[39] A. DeOrio, A. Bauserman, and V. Bertacco. Post-silicon verification for cache coherence. In *2008 IEEE International Conference on Computer Design*, pages 348–355, Oct 2008. doi: 10.1109/ICCD.2008.4751884.

[40] Ilya Wagner and Valeria Bertacco. Mcjammer: Adaptive verification for multi-core designs. In *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*, pages 670–675, 2008.

[41] Meng Zhang, Jesse D. Bingham, John Erickson, and Daniel J. Sorin. Pvcoherence: Designing flat coherence protocols for scalable verification. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*, pages 392–403, 2014.

[42] C. C. Yen, T. Lin, H. Lin, K. Yang, T. Liu, and Y. C. Hsu. Diagnosing silicon failures based on functional test patterns. In *Seventh International Workshop on Microprocessor Test and Verification (MTV'06)*, pages 94–98, Dec 2006. doi: 10.1109/MTV.2006.9.

[43] D. Van Campenhout, T. Mudge, and J. P. Hayes. Collection and analysis of microprocessor design errors. *IEEE Design Test of Computers*, 17(4):51–60, Oct 2000. ISSN 0740-7475. doi: 10.1109/54.895006.

[44] M. N. Velev. Collection of high-level microprocessor bugs from formal verification of pipelined and superscalar designs. In *Test Conference. Proceedings. ITC 2003. International*, volume 1, pages 138–147, Sept 2003. doi: 10.1109/TEST.2003.1270834.

[45] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A reconfigurable design-for-debug infrastructure for socs. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 7–12, 2006. doi: 10.1109/DAC.2006.238683.

[46] https://software.intel.com/en-us/intel-platform-analysis-library. Intel Platform Analysis Li- brary.

[47] www.arm.com. CoreSight On-Chip Trace and Debug Architecture.

[48] C. S. Zhu, G. Weissenbacher, D. Sethi, and S. Malik. Sat-based techniques for determining backbones for post-silicon fault localisation. In *High Level Design Validation and Test Workshop (HLDVT), 2011 IEEE International*, pages 84–91, Nov 2011. doi: 10.1109/HLDVT.2011.6113981.

[49] S. BeigMohammadi and B. Alizadeh. Combinational trace signal selection with improved state restoration for post-silicon debug. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1369–1374, March 2016.

[50] X. Liu and Q. Xu. On signal selection for visibility enhancement in trace-based post-silicon validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(8):1263–1274, Aug 2012. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2189395.

[51] B. Kumar, A. Jindal, M. Fujita, and V. Singh. Post-silicon observability enhancement with topology based trace signal selection. In *2017 18th IEEE Latin American Test Symposium (LATS)*, pages 1–6, March 2017. doi: 10.1109/LATW.2017.7906761.

[52] P. Taatizadeh and N. Nicolici. Automated selection of assertions for bit-flip detection during post-silicon validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP(99):1–1, 2016. ISSN 0278-0070. doi: 10.1109/TCAD.2016.2538087.

[53] S. Chandran, P. R. Panda, D. Chauhan, S. Kumar, and S. R. Sarangi. Extending trace history through tapered summaries in post-silicon validation. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 737–742, Jan 2016. doi: 10.1109/ASPDAC.2016.7428099.

[54] D. Josephson and B. Gottlieb. The crazy mixed up world of silicon debug [ic validation]. In *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pages 665–670, Oct 2004. doi: 10.1109/CICC.2004.1358915.

[55] B. Kumar, A. Jindal, V. Singh, and M. Fujita. A methodology for trace signal selection to improve error detection in post-silicon validation. In *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, pages 147–152, Jan 2017. doi: 10.1109/VLSID.2017.66.

[56] M. Gort, F. M. De Paula, J. J. W. Kuan, T. M. Aamodt, A. J. Hu, S. J. E. Wilton, and J. Yang. Formal-analysis-based trace computation for post-silicon debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20 (11):1997–2010, Nov 2012. ISSN 1063-8210. doi: 10.1109/TVLSI.2011.2166416.

[57] O. Caty, P. Dahlgren, and I. Bayraktaroglu. Microprocessor silicon debug based on failure propagation tracing. In *IEEE International Conference on Test, 2005.*, pages 10 pp.–293, Nov 2005. doi: 10.1109/TEST.2005.1583986.

[58] G. J. Van Rootselaar and B. Vermeulen. Silicon debug: scan chains alone are not enough. In *Test Conference, 1999. Proceedings. International*, pages 892–902, 1999. doi: 10.1109/TEST.1999.805821.

[59] Miron Abramovici. In-system silicon validation and debug. *IEEE Des. Test*, 25 (3):216–223, May 2008. ISSN 0740-7475.

[60] M. Boule and Z. Zilic. Incorporating efficient assertion checkers into hardware emulation. In *2005 International Conference on Computer Design*, pages 221–228, Oct 2005. doi: 10.1109/ICCD.2005.66.

[61] M. Boule, J. S. Chenard, and Z. Zilic. Assertion checkers in verification, silicon debug and in-field diagnosis. In *8th International Symposium on Quality Electronic Design (ISQED'07)*, pages 613–620, March 2007. doi: 10.1109/ISQED.2007.38.

[62] M. H. Neishaburi and Z. Zilic. Enabling efficient post-silicon debug by clustering of hardware-assertions. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 985–988, March 2010. doi: 10.1109/DATE.2010.5456904.

[63] B. Kumar, K. Basu, M. Fujita, and V. Singh. Rtl level trace signal selection and coverage estimation during post-silicon validation. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 59–66, Oct 2017. doi: 10.1109/HLDVT.2017.8167464.

[64] N. Foutris, D. Gizopoulos, M. Psarakis, X. Vera, and A. Gonzalez. Accelerating microprocessor silicon validation by exposing isa diversity. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 386–397, Dec 2011.

[65] I. Wagner and V. Bertacco. Reversi: Post-silicon validation system for modern microprocessors. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 307–314, Oct 2008. doi: 10.1109/ICCD.2008.4751878.

[66] T. Hong, Y. Li, S. B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra. Qed: Quick error detection tests for effective post-silicon validation. In *2010 IEEE International Test Conference*, pages 1–10, Nov 2010. doi: 10.1109/TEST.2010.5699215.

[67] D. Lin, T. Hong, Y. Li, E. S, S. Kumar, F. Fallah, N. Hakim, D. S. Gardner, and S. Mitra. Effective post-silicon validation of system-on-chips using quick error detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1573–1590, Oct 2014. ISSN 0278-0070. doi: 10.1109/TCAD.2014.2334301.

[68] D. Lin, E. Singh, C. Barrett, and S. Mitra. A structured approach to post-silicon validation and debug using symbolic quick error detection. In *Test Conference (ITC), 2015 IEEE International*, pages 1–10, Oct 2015. doi: 10.1109/TEST.2015.7342397.

[69] S. B. Park and S. Mitra. Ifra: Instruction footprint recording and analysis for post-silicon bug localization in processors. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 373–378, June 2008.

[70] S. B. Park, A. Bracy, H. Wang, and S. Mitra. Blog: Post-silicon bug localization in processors using bug localization graphs. In *Design Automation Conference*, pages 368–373, June 2010.

[71] O. Friedler, W. Kadry, A. Morgenshtein, A. Nahir, and V. Sokhin. Effective post-silicon failure localization using dynamic program slicing. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014. doi: 10.7873/DATE.2014.332.

[72] Fong Pong and Michel Dubois. A new approach for the verification of cache coherence protocols. *IEEE Trans. Parallel Distrib. Syst.*, 6(8):773–787, August 1995. ISSN 1045-9219.

[73] Fong Pong and Michel Dubois. The verification of cache coherence protocols. In *Proceedings of the Fifth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '93, pages 11–20, New York, NY, USA, 1993. ACM. ISBN 0-89791-599-2.

[74] Jason F Cantin, Mikko H Lipasti, and James E Smith. Dynamic verification of cache coherence protocols. In *High Performance Memory Systems*, pages 25–42. Springer, 2004.

[75] D. Borodin and B. H. H. Juurlink. A low-cost cache coherence verification method for snooping systems. In *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, pages 219–227, Sep. 2008. doi: 10.1109/DSD.2008.33.

[76] R. Fernandez-Pascual, J. M. Garcia, M. E. Acacio, and J. Duato. A low overhead fault tolerant coherence protocol for cmp architectures. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 157–168, Feb 2007. doi: 10.1109/HPCA.2007.346194.

[77] Andrew DeOrio, Ilya Wagner, and Valeria Bertacco. Dacota: Post-silicon validation of the memory subsystem in multi-core designs. In *15th International*

Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, North Carolina, USA, pages 405–416, 2009.

[78] Sudheendra Hangal, Durgam Vahia, Chaiyasit Manovit, and Juin-Yeu Joseph Lu. Tsotool: A program for verifying memory systems using the memory consistency model. *SIGARCH Comput. Archit. News*, 32(2):114–, March 2004. ISSN 0163-5964.

[79] Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 29(12):66–76, December 1996. ISSN 0018-9162.

[80] Kaiyu Chen, Sharad Malik, and Priyadarsan Patra. Runtime validation of memory ordering using constraint graph checking. In *14th International Conference on High-Performance Computer Architecture (HPCA-14 2008), 16-20 February 2008, Salt Lake City, UT, USA*, pages 415–426, 2008.

[81] Albert Meixner and Daniel J. Sorin. Dynamic verification of memory consistency in cache-coherent multithreaded computer architectures. *IEEE Trans. Dependable Secur. Comput.*, 6(1):18–31, January 2009. ISSN 1545-5971.

[82] Biruk W. Mammo, Valeria Bertacco, Andrew DeOrio, and Ilya Wagner. Post-silicon validation of multiprocessor memory consistency. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(6):1027–1037, 2015.

[83] Marco Elver and Vijay Nagarajan. Mcversi: A test generation framework for fast memory consistency verification in simulation. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*, pages 618–630, 2016.

[84] Anders Landin, Erik Hagersten, and Seif Haridi. Race-free interconnection networks and multiprocessor consistency. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, ISCA '91, pages 106–115, New York, NY, USA, 1991. ACM. ISBN 0-89791-394-9.

[85] I. Wagner and V. Bertacco. Caspar: Hardware patching for multicore processors. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 658–663, April 2009.

[86] Hyungmin Cho, Eric Cheng, Thomas Shepherd, Chen-Yong Cher, and Subhasish Mitra. System-level effects of soft errors in uncore components. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 36(9):1497–1510, 2017.

[87] Debjit Pal, Abhishek Sharma, Sandip Ray, Flavio M. de Paula, and Shobha Vasudevan. Application level hardware tracing for scaling post-silicon debug. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 92:1–92:6, 2018.

[88] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[89] Zissis Poulos and Andreas G. Veneris. Clustering-based failure triage for RTL regression debugging. In *2014 International Test Conference, ITC 2014, Seattle, WA, USA, October 20-23, 2014*, pages 1–10, 2014.

[90] Andrew DeOrio, Qingkun Li, Matthew Burgess, and Valeria Bertacco. Machine learning-based anomaly detection for post-silicon bug diagnosis. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 491–496, 2013.

[91] Valeria Bertacco and Wade Bonkowski. Ithelps: Iterative high-accuracy error localization in post-silicon. In *33rd IEEE International Conference on Computer Design, ICCD 2015, New York City, NY, USA, October 18-21, 2015*, pages 196–199, 2015.

[92] A. DeOrio, D. S. Khudia, and V. Bertacco. Post-silicon bug diagnosis with inconsistent executions. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 755–761, Nov 2011. doi: 10.1109/ICCAD.2011.6105414.

[93] Biruk Mammo, Milind Furia, Valeria Bertacco, Scott A. Mahlke, and Daya Shanker Khudia. Bugmd: automatic mismatch diagnosis for bug triaging. In *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, page 117, 2016.

[94] A. DeOrio, J. Li, and V. Bertacco. Bridging pre- and post-silicon debugging with biped. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 95–100, Nov 2012.

[95] H. F. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(2):285–297, Feb 2009. ISSN 0278-0070. doi: 10.1109/TCAD.2008.2009158.

[96] S. Ma, D. Pal, R. Jiang, S. Ray, and S. Vasudevan. Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 1–8, Nov 2015. doi: 10.1109/ICCAD.2015.7372542.

[97] Andre Suelflow, Goerschwin Fey, Roderick Bloem, and Rolf Drechsler. Using unsatisfiable cores to debug multiple design errors. In *Proceedings of the 18th ACM Great Lakes Symposium on VLSI*, GLSVLSI '08, pages 77–82, 2008. ISBN 978-1-59593-999-9.

[98] S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design, 2007. FMCAD '07*, pages 13–19, Nov 2007.

[99] Sean Safarpour, Görschwin Fey, Andreas G. Veneris, and Rolf Drechsler. Utilizing don't care states in sat-based bounded sequential problems. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI 2005, Chicago, Illinois, USA, April 17-19, 2005*, pages 264–269, 2005.

[100] Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Silicon fault diagnosis using sequence interpolation with backbones. In *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2014, San Jose, CA, USA, November 3-6, 2014*, pages 348–355, 2014.

[101] Y. S. Yang, A. Veneris, N. Nicolici, and M. Fujita. Automated data analysis techniques for a modern silicon debug environment. In *17th Asia and South Pacific Design Automation Conference*, pages 298–303, Jan 2012. doi: 10.1109/ASPDAC.2012.6164963.

[102] M. Li and A. Davoodi. A hybrid approach for fast and accurate trace signal selection for post-silicon debug. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 485–490, March 2013.

[103] K. Rahmani and P. Mishra. Efficient signal selection using fine-grained combination of scan and trace buffers. In *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, pages 308–313, Jan 2013. doi: 10.1109/VLSID.2013.206.

[104] S. BeigMohammadi and B. Alizadeh. Combinational trace signal selection with improved state restoration for post-silicon debug. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1369–1374, March 2016.

[105] P. Komari and R. Vemuri. A novel simulation based approach for trace signal selection in silicon debug. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 193–200, Oct 2016. doi: 10.1109/ICCD.2016.7753280.

[106] X. Liu and Q. Xu. On multiplexed signal tracing for post-silicon debug. In *Design, Automation Test in Europe*, pages 1–6, March 2011. doi: 10.1109/DATE.2011.5763116.

[107] P. Thakyal and P. Mishra. Layout-aware selection of trace signals for post-silicon debug. In *2014 IEEE Computer Society Annual Symposium on VLSI*, pages 326–331, July 2014. doi: 10.1109/ISVLSI.2014.19.

[108] X. Liu and Q. Xu. On signal selection for visibility enhancement in trace-based post-silicon validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(8):1263–1274, Aug 2012. ISSN 0278-0070. doi: 10.1109/TCAD.2012.2189395.

[109] D. Chatterjee, C. McCarter, and V. Bertacco. Simulation-based signal selection for state restoration in silicon debug. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 595–601, Nov 2011. doi: 10.1109/ICCAD.2011.6105391.

[110] K. Rahmani, S. Ray, and P. Mishra. Postsilicon trace signal selection using machine learning techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):570–580, Feb 2017. ISSN 1063-8210. doi: 10.1109/TVLSI.2016.2593902.

[111] K. Basu, P. Mishra, P. Patra, A. Nahir, and A. Adir. Dynamic selection of trace signals for post-silicon debug. In *14th International Workshop on Microprocessor Test and Verification*, pages 62–67, Dec 2013. doi: 10.1109/MTV.2013.13.

[112] K. Basu, P. Mishra, and P. Patra. Efficient combination of trace and scan signals for post silicon validation and debug. In *2011 IEEE International Test Conference*, pages 1–8, Sept 2011. doi: 10.1109/TEST.2011.6139157.

[113] E. Hung and S. J. E. Wilton. On evaluating signal selection algorithms for post-silicon debug. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pages 1–7, March 2011. doi: 10.1109/ISQED.2011.5770739.

[114] H. F. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(2):285–297, Feb 2009. ISSN 0278-0070. doi: 10.1109/TCAD.2008.2009158.

[115] H. F. Ko and N. Nicolici. Automated trace signals selection using the rtl descriptions. In *2010 IEEE International Test Conference*, pages 1–10, Nov 2010.

[116] Binod Kumar, Ankit Jindal, Masahiro Fujita, and Virendra Singh. Combining restorability and error detection ability for effective trace signal selection. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 191–196, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4972-7.

[117] H. F. Ko and N. Nicolici. Mapping trigger conditions onto trigger units during post-silicon validation and debugging. *IEEE Transactions on Computers*, 61(11): 1563–1575, Nov 2012. ISSN 0018-9340. doi: 10.1109/TC.2011.192.

[118] S. Prabhakar and M. S. Hsiao. Multiplexed trace signal selection using non-trivial implication-based correlation. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 697–704, March 2010. doi: 10.1109/ISQED.2010.5450503.

[119] B. Kumar, K. Basu, A. Jindal, M. Fujita, and V. Singh. Improving post-silicon error detection with topological selection of trace signals. In *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, Oct 2017. doi: 10.1109/VLSI-SoC.2017.8203485.

[120] K. Han, J. S. Yang, and J. A. Abraham. Enhanced algorithm of combining trace and scan signals in post-silicon validation. In *VLSI Test Symposium (VTS), 2013 IEEE 31st*, pages 1–6, April 2013. doi: 10.1109/VTS.2013.6548915.

[121] H. F. Ko and N. Nicolici. Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging. In *2010 15th IEEE European Test Symposium*, pages 62–67, May 2010. doi: 10.1109/ETSYM.2010.5512781.

[122] E. Anis and N. Nicolici. Low cost debug architecture using lossy compression for silicon debug. In *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007. doi: 10.1109/DATE.2007.364595.

[123] E. Anis Daoud and N. Nicolici. On using lossy compression for repeatable experiments during silicon debug. *IEEE Transactions on Computers*, 60(7):937–950, July 2011. ISSN 0018-9340. doi: 10.1109/TC.2010.122.

[124] E. Anis and N. Nicolici. On using lossless compression of debug data in embedded logic analysis. In *2007 IEEE International Test Conference*, pages 1–10, Oct 2007. doi: 10.1109/TEST.2007.4437613.

[125] H. F. Ko, A. B. Kinsman, and N. Nicolici. Distributed embedded logic analysis for post-silicon validation of socs. In *2008 IEEE International Test Conference*, pages 1–10, Oct 2008. doi: 10.1109/TEST.2008.4700594.

[126] S. Prabhakar, R. Sethuram, and M. S. Hsiao. Trace buffer-based silicon debug with lossless compression. In *2011 24th Internatioal Conference on VLSI Design*, pages 358–363, Jan 2011. doi: 10.1109/VLSID.2011.31.

[127] Kanad Basu and Prabhat Mishra. Efficient trace data compression using statically selected dictionary. In *29th IEEE VLSI Test Symposium, VTS 2011, May 1-5, 2011, Dana Point, California, USA*, pages 14–19, 2011. doi: 10.1109/VTS.2011.5783748.

[128] R. Kapur and T. W. Williams. Tough challenges as design and test go nanometer. *Computer*, 32(11):42–45, Nov 1999. ISSN 0018-9162. doi: 10.1109/2.803639.

[129] H Ando. Testing vlsi with random acecss scan. In *Digest of Computer Society International Conference*, pages 50–52, 1980.

[130] Dong Hyun Baik, K. K. Saluja, and S. Kajihara. Random access scan: a solution to test power, test data volume and test time. In *VLSI Design. Proceedings. 17th International Conference on*, pages 883–888, 2004. doi: 10.1109/ICVD.2004.1261042.

[131] Mark S. Papamarcos and Janak H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proceedings of the 11th Annual Symposium on Computer Architecture, Ann Arbor, USA, June 1984*, pages 348–354, 1984.

[132] Daniel J. Sorin, Milo M. K. Martin, Mark D. Hill, and David A. Wood. Safetynet: Improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, pages 123–134, Washington, DC, USA, 2002. IEEE Computer Society.

[133] https://people.eecs.berkeley.edu/p̃attrsn/252F96/Lecture18.pdf.

[134] http://projects.csail.mit.edu/heracles/.

[135] Giorgio Delzanno. Automatic verification of parameterized cache coherence protocols. In *Proceedings of the 12th International Conference on Computer Aided Verification*, CAV '00, pages 53–68, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67770-4.

[136] Allon Adir, Maxim Golubev, Shimon Landa, Amir Nahir, Gil Shurek, Vitali Sokhin, and Avi Ziv. Threadmill: A post-silicon exerciser for multi-threaded processors. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 860–865, New York, NY, USA, 2011. ISBN 978-1-4503-0636-2.

[137] R. Fernandez-Pascual, J. M. Garcia, M. E. Acacio, and J. Duato. A fault-tolerant directory-based cache coherence protocol for cmp architectures. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 267–276, June 2008. doi: 10.1109/DSN.2008.4630095.

[138] A. Meixner and D. J. Sorin. Error detection via online checking of cache coherence with token coherence signatures. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 145–156, Feb 2007. doi: 10.1109/HPCA.2007.346193.

[139] Milos Prvulovic, Zheng Zhang, and Josep Torrellas. Revive: Cost-effective architectural support for rollback recovery in shared-memory multiprocessors. *SIGARCH Comput. Archit. News*, 30(2):111–122, May 2002. ISSN 0163-5964.

[140] Kai hui Chang, I. L. Markov, and V. Bertacco. Automating post-silicon debugging and repair. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 91–98, Nov 2007. doi: 10.1109/ICCAD.2007.4397249.

[141] M. Li and A. Davoodi. Multi-mode trace signal selection for post-silicon debug. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 640–645, Jan 2014. doi: 10.1109/ASPDAC.2014.6742963.

[142] E. Hung and S. J. E. Wilton. Scalable signal selection for post-silicon debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(6):1103–1115, June 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2012.2202409.

[143] http://www.opencores.org/.

[144] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT*, page 2008.

[145] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems,*, pages 1929–1934 vol.3, May 1989. doi: 10.1109/ISCAS.1989.100747.

[146] L. Basto. First results of itc'99 benchmark circuits. *IEEE Design Test of Computers*, 17(3):54–59, July 2000. ISSN 0740-7475. doi: 10.1109/54.867895.

[147] H. Sabaghian-Bidgoli, P. Behnam, B. Alizadeh, and Z. Navabi. Reducing search space for fault diagnosis: A probability-based scoring approach. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 545–550, July 2017.

[148] J. S. Yang and N. A. Touba. Efficient trace signal selection for silicon debug by error transmission analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):442–446, March 2012. ISSN 0278-0070.

[149] B. Kumar, A. Jindal, and V. Singh. A trace signal selection algorithm for improved post-silicon debug. In *2016 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–4, Oct 2016. doi: 10.1109/EWDTS.2016.7807700.

[150] A. Bhattacharya, S. Koley, and A. Banerjee. Considering multi-cycle influences for signal selection for post silicon validation. In *Electronic Design, Computer Networks Automated Verification (EDCAV), 2015 International Conference on*, pages 160–164, Jan 2015. doi: 10.1109/EDCAV.2015.7060559.

[151] P. Taatizadeh and N. Nicolici. Automated selection of assertions for bit-flip detection during post-silicon validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):2118–2130, 2016. ISSN 0278-0070. doi: 10.1109/TCAD.2016.2538087.

[152] Jianliang GAO, Yinhe HAN, and Xiaowei LI. A novel post-silicon debug mechanism based on suspect window. *IEICE Transactions on Information and Systems*, E93.D(5):1175–1185, 2010.

[153] Prateek Thakyal and Prabhat Mishra. Layout-aware signal selection in reconfigurable architectures. In *18th International Symposium on VLSI Design and Test, VDAT 2014, Coimbatore, India, July 16-18, 2014*, pages 1–6, 2014.

[154] Yu HU, Jing YE, Zhiping SHI, and Xiaowei LI. Laps: Layout-aware path selection for post-silicon timing characterization. *IEICE Transactions on Information and Systems*, E100.D(2):323–331, 2017. doi: 10.1587/transinf.2016EDP7184.

[155] J. Ye, Y. Huang, Y. Hu, W. T. Cheng, R. Guo, L. Lai, T. P. Tai, X. Li, W. Changchien, D. M. Lee, J. J. Chen, S. C. Eruvathi, K. K. Kumara, C. Liu, and S. Pan. Diagnosis and layout aware (dla) scan chain stitching. In *2013 IEEE International Test Conference (ITC)*, pages 1–10, Sept 2013. doi: 10.1109/TEST.2013.6651929.

[156] L. H. Goldstein and E. L. Thigpen. Scoap: Sandia controllability/observability

analysis program. In *Papers on Twenty-five Years of Electronic Design Automation*, 25 years of DAC, pages 397–403, New York, NY, USA, 1988. ACM. ISBN 0-89791-267-5.

[157] C. C. Yu and J. P. Hayes. Scalable and accurate estimation of probabilistic behavior in sequential circuits. In *2010 28th VLSI Test Symposium (VTS)*, pages 165–170, April 2010. doi: 10.1109/VTS.2010.5469586.

[158] C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, Sept 1961. ISSN 0367-9950. doi: 10.1109/TEC.1961.5219222.

[159] A. Nahir, M. Dusanapudi, S. Kapoor, K. Reick, W. Roesner, K. D. Schubert, K. Sharp, and G. Wetli. Post-silicon validation of the ibm power8 processor. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2014. doi: 10.1145/2593069.2593183.

[160] S. Chandran, P. R. Panda, S. R. Sarangi, A. Bhattacharyya, D. Chauhan, and S. Kumar. Managing trace summaries to minimize stalls during postsilicon validation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(6): 1881–1894, June 2017. ISSN 1063-8210. doi: 10.1109/TVLSI.2017.2657604.

[161] R. McLaughlin, S. Venkataraman, and C. Lim. Automated debug of speed path failures using functional tests. In *2009 27th IEEE VLSI Test Symposium*, pages 91–96, May 2009. doi: 10.1109/VTS.2009.53.

[162] A. G. Veneris and I. N. Hajj. Design error diagnosis and correction in vlsi digital circuits. In *Proceedings of 40th Midwest Symposium on Circuits and Systems.*, volume 2, pages 1005–1008 vol.2, Aug 1997. doi: 10.1109/MWSCAS.1997.662246.

[163] A. G. Veneris and I. N. Hajj. Design error diagnosis and correction in vlsi digital circuits. In *Proceedings of 40th Midwest Symposium on Circuits and Systems.*, volume 2, pages 1005–1008 vol.2, Aug 1997. doi: 10.1109/MWSCAS.1997.662246.

[164] M. S. Abadir, J. Ferguson, and T. E. Kirkland. Logic design verification via test generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):138–148, Jan 1988. ISSN 0278-0070. doi: 10.1109/43.3141.

[165] S. Choudhary, A. M. Gharehbaghi, T. Matsumoto, and M. Fujita. Trace signal selection methods for post silicon debugging. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 258–263, Oct 2015. doi: 10.1109/VLSI-SoC.2015.7314426.

[166] X. Liu and R. Vemuri. Effective signal restoration in post-silicon validation. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 169–176, Nov 2017. doi: 10.1109/ICCD.2017.34.

[167] P. Komari and R. Vemuri. A novel simulation based approach for trace signal selection in silicon debug. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 193–200, Oct 2016. doi: 10.1109/ICCD.2016.7753280.

[168] Yun Cheng, Huawei Li, Ying Wang, Yingke Gao, Bo Liu, and Xiaowei Li. Flip-flop clustering based trace signal selection for post-silicon debug. In *2017 IEEE 35th VLSI Test Symposium (VTS)*, pages 1–6, April 2017. doi: 10.1109/VTS.2017.7928929.

[169] K. Rahmani, P. Mishra, and S. Ray. Scalable trace signal selection using machine learning. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 384–389, Oct 2013. doi: 10.1109/ICCD.2013.6657069.

[170] K. Rahmani and P. Mishra. Feature-based signal selection for post-silicon debug using machine learning. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017. doi: 10.1109/TETC.2017.2787610.

[171] Eman El Mandouh and Amr G. Wassal. Application of machine learning techniques in post-silicon debugging and bug localization. *Journal of Electronic Testing*, 34 (2):163–181, Apr 2018.

[172] C. S. Zhu and S. Malik. Optimizing dynamic trace signal selection using machine learning and linear programming. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1289–1292, March 2015. doi: 10.7873/DATE.2015.0573.

[173] Li-C. Wang. Data mining in design and test processes: basic principles and promises. In *International Symposium on Physical Design, ISPD'13, Stateline, NV, USA, March 24-27, 2013*, pages 41–42, 2013.

[174] Magdy S. Abadir, Nik Sumikawa, Wen Chen, and Li-C. Wang. Data mining based prediction paradigm and its applications in design automation. In *Proceedings of Technical Program of 2012 VLSI Design, Automation and Test, VLSI-DAT 2012, Hsinchu, Taiwan, April 23-25, 2012*, page 1, 2012.

[175] Pouria Bastani, Nicholas Callegari, Li-C. Wang, and Magdy S. Abadir. Feature-ranking methodology to diagnose design-silicon timing mismatch. *IEEE Design & Test of Computers*, 27(3):42–53, 2010.

[176] Sean H. Wu, Dragoljub Gagi Drmanac, and Li-C. Wang. A study of outlier analysis techniques for delay testing. In *2008 IEEE International Test Conference, ITC 2008, Santa Clara, California, USA, October 26-31, 2008*, pages 1–10, 2008.

[177] Xavier Amatriain. The recommender problem revisited. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 397–398, New York, NY, USA, 2014. ACM.

[178] A. Vali and N. Nicolici. Bit-flip detection-driven selection of trace signals. In *21th IEEE European Test Symposium (ETS)*, pages 1–6, May 2016. doi: 10.1109/ETS.2016.7519315.

[179] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[180] https://scikit-learn.org/stable/modules/neighbors.html.

[181] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html.

[182] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html.

[183] Y. Cheng, H. Li, Y. Wang, and X. Li. Cluster restoration based trace signal selection for post-silicon debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2018. ISSN 0278-0070. doi: 10.1109/TCAD.2018.2818690.

[184] A. Jindal, B. Kumar, K. Basu, and M. Fujita. Elura: A methodology for post-silicon gate-level error localization using regression analysis. In *2018 31st International Conference on VLSI Design(VLSID'18)*, Pune, India, Jan 2018.

[185] I. Choi, H. Oh, Y. Lee, and S. Kang. Test resource reused debug scheme to reduce the post-silicon debug cost. *IEEE Transactions on Computers*, 67(12):1835–1839, Dec 2018. ISSN 0018-9340. doi: 10.1109/TC.2018.2835462.

[186] Bart Vermeulen, Mohammad Zalfany Urfianto, and Sandeep Kumar Goel. Automatic generation of breakpoint hardware for silicon debug. In *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 514–517, 2004. doi: 10.1145/996566.996708.

[187] J. S. Yang and N. A. Touba. Improved trace buffer observation via selective data capture using 2-d compaction for post-silicon debug. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(2):320–328, Feb 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2012.2183399.

[188] H. Oh, T. Han, I. Choi, and S. Kang. An on-chip error detection method to reduce the post-silicon debug time. *IEEE Transactions on Computers*, PP(99):1–1, 2016. ISSN 0018-9340. doi: 10.1109/TC.2016.2561920.

[189] S. Deutsch and K. Chakrabarty. Massive signal tracing using on-chip dram for in-system silicon debug. In *2014 International Test Conference*, pages 1–10, Oct 2014. doi: 10.1109/TEST.2014.7035363.

[190] H. F. Ko, A. B. Kinsman, and N. Nicolici. Design-for-debug architecture for distributed embedded logic analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(8):1380–1393, Aug 2011. ISSN 1063-8210. doi: 10.1109/TVLSI.2010.2050501.

[191] H. Oh, I. Choi, and S. Kang. Dram-based error detection method to reduce the post-silicon debug time for multiple identical cores. *IEEE Transactions on Computers*, 66(9):1504–1517, Sep. 2017. ISSN 0018-9340. doi: 10.1109/TC.2017.2678504.

[192] S. Deutsch and K. Chakrabarty. Test and debug solutions for 3d-stacked integrated circuits. In *2015 IEEE International Test Conference (ITC)*, pages 1–10, Oct 2015. doi: 10.1109/TEST.2015.7342421.

[193] W. Jung, H. Oh, D. Kang, and S. Kang. A 2-d compaction method using macro block for post-silicon validation. In *2015 International SoC Design Conference (ISOCC)*, pages 41–42, Nov 2015. doi: 10.1109/ISOCC.2015.7401690.

[194] Xinli Gu, Weili Wang, K. Li, Heon Kim, and S. S. Chung. Re-using dft logic for functional and silicon debugging test. In *Test Conference, 2002. Proceedings. International*, pages 648–656, 2002. doi: 10.1109/TEST.2002.1041816.

[195] J. Tudu. Jscan: A joint-scan dft architecture to minimize test time, pattern volume, and power. In *Proc. of VLSI Design and Test Symposium*, pages 186–191, May 2016.

[196] S. Ahlawat, J. Tudu, A. Matrosova, and V. Singh. A high performance scan flip-flop design for serial and mixed mode scan test. *IEEE Transactions on Device and Materials Reliability*, 18(2):321–331, June 2018.

[197] S. R. Sarangi, B. Greskamp, and J. Torrellas. Cadre: Cycle-accurate deterministic replay for hardware debugging. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 301–312, June 2006. doi: 10.1109/DSN.2006.19.

[198] Yunji Chen, Shijin Zhang, Qi Guo, Ling Li, Ruiyang Wu, and Tianshi Chen. Deterministic replay: A survey. *ACM Comput. Surv.*, 48(2):17:1–17:47, September 2015. ISSN 0360-0300.

[199] S. Chen, M. Hsiao, W. Jone, and T. Chen. A configurable bus-tracer for error reproduction in post-silicon validation. In *2013 International Symposium onVLSI Design, Automation, and Test (VLSI-DAT)*, pages 1–4, April 2013.

[200] Ting Wang, Yannan Liu, Qiang Xu, Zhaobo Zhang, Zhiyuan Wang, and Xinli Gu. Retrodmr: Troubleshooting non-deterministic faults with retrospective dmr. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, pages 638–641, 2017.

[201] E. A. Daoud and N. Nicolici. Real-time lossless compression for silicon debug. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(9): 1387–1400, Sep. 2009.

[202] Y. Cheng, H. Li, and X. Li. An on-line timing error detection method for silicon debug. In *2014 IEEE 23rd Asian Test Symposium*, pages 263–268, Nov 2014. doi: 10.1109/ATS.2014.63.

[203] E. A. Daoud and N. Nicolici. Embedded debug architecture for bypassing blocking bugs during post-silicon validation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(4):559–570, April 2011. ISSN 1063-8210. doi: 10.1109/TVLSI.2009.2038390.

[204] L. Liu, D. Sheridan, W. Tuohy, and S. Vasudevan. A technique for test coverage closure using goldmine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(5):790–803, May 2012.

[205] L. Liu, C. H. Lin, and S. Vasudevan. Word level feature discovery to enhance quality of assertion mining. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 210–217, Nov 2012.

[206] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson. Goldmine: Automatic assertion generation using data mining and static analysis. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 626–629, March 2010. doi: 10.1109/DATE.2010.5457129.

[207] F. Farahmandi, R. Morad, A. Ziv, Z. Nevo, and P. Mishra. Cost-effective analysis of post-silicon functional coverage events. In *2017 Design, Automation Test in Europe Conference Exhibition (DATE 2017)*, March 2017.

[208] T. Inoue, K. Hasegawa, M. Yanagisawa, and N. Togawa. Designing hardware trojans and their detection based on a svm-based approach. In *2017 IEEE 12th International Conference on ASIC (ASICON)*, pages 811–814, Oct 2017.

[209] K. Hasegawa, M. Yanagisawa, and N. Togawa. Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[210] F. Chen and Q. Liu. Single-triggered hardware trojan identification based on gate-level circuit structural characteristics. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017.

[211] Q. Liu, P. Zhao, and F. Chen. A hardware trojan detection method based on structural features of trojan and host circuits. *IEEE Access*, 7:44632–44644, 2019.

[212] C. H. Kok, C. Y. Ooi, M. Moghbel, N. Ismail, H. S. Choo, and M. Inoue. Classification of trojan nets based on scoap values using supervised learning. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2019. doi: 10.1109/ISCAS.2019.8702462.

[213] R. O. Gallardo, A. J. Huy, A. Ivanov, and M. S. Mirian. Reducing post-silicon coverage monitoring overhead with emulation and bayesian feature selection. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 816–823, Nov 2015. doi: 10.1109/ICCAD.2015.7372655.

[214] Y. Huang, A. Chattopadhyay, and P. Mishra. Trace buffer attack: Security versus observability study in post-silicon debug. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 355–360, Oct 2015. doi: 10.1109/VLSI-SoC.2015.7314443.

[215] Jerry Backer, David Hely, and Ramesh Karri. Secure and flexible trace-based debugging of systems-on-chip. *ACM Trans. Des. Autom. Electron. Syst.*, 22(2): 31:1–31:25, December 2016. ISSN 1084-4309.

[216] Neetu Jindal, Preeti Ranjan Panda, and Smruti R. Sarangi. Reusing trace buffers as victim caches. *IEEE Trans. VLSI Syst.*, 26(9):1699–1712, 2018.

[217] http://www.pld.ttu.ee/ maksim/benchmarks/iscas89/bench/.