Algorithms and Low Cost Architectures for Trace Buffer-Based Silicon Debug

Sandesh Prabhakar

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

Master of Science
In
Computer Engineering

Michael S. Hsiao, Chair
Christopher L. Wyatt
Patrick Schaumont

December 1, 2009
Blacksburg, Virginia

Keywords: Silicon Debug, Logic Implications, Trace selection, State Restoration, Trace Compression

Copyright ©2009, Sandesh Prabhakar
Algorithms and Low Cost Architectures for Trace Buffer-Based Silicon Debug

Sandesh Prabhakar

ABSTRACT

An effective silicon debug technique uses a trace buffer to monitor and capture a portion of the circuit response during its functional, post-silicon operation. Due to the limited space of the available trace buffer, selection of the critical trace signals plays an important role in both minimizing the number of signals traced and maximizing the observability/restorability of other untraced signals during post-silicon validation. In this thesis, a new method is proposed for trace buffer signal selection for the purpose of post-silicon debug. The selection is performed by favoring those signals with the most number of implications that are not implied by other signals. Then, based on the values of the traced signals during silicon debug, an algorithm which uses a SAT-based multi-node implication engine is introduced to restore the values of untraced signals across multiple time-frames. A new multiplexer-based trace signal interconnection scheme and a new heuristic for trace signal selection based on implication-based correlation are also described. By this approach, we can effectively trace twice as many signals with the same trace buffer width. A SAT-based greedy heuristic is also proposed to prune the selected trace signal list further to take into account those multi-node implications. A state restoration algorithm is developed for the multiplexer-based trace signal interconnection scheme. Experimental results show that the proposed approaches select the trace signals effectively, giving a high restoration percentage compared with other techniques. We finally propose a lossless compression technique to increase the capacity of the trace buffer. We propose real-time compression of the trace data using Frequency-Directed Run-Length (FDR) code. In addition, we also propose source transformation functions, namely difference vector computation, efficient ordering of trace flip-flops and alternate vector reversal that reduces the entropy of the trace data, making them more amenable for compression. The order of the trace flip-flops is computed off-chip using a probabilistic algorithm. The difference vector computation and alternate vector reversal are implemented on-chip and incurs negligible hardware overhead. Experimental results for
sequential benchmark circuits shows that this method gives a better compression percentage compared to dictionary-based techniques and yields up to 3X improvement in the diagnostic capability. We also observe that the area overhead of the proposed approach is less compared to dictionary-based compression techniques.
Acknowledgments

I want to express my sincere gratitude to my advisor Professor Michael Hsiao for his continued guidance, motivation and encouragement throughout the duration of my research. I want to thank him for giving me the opportunity to work with him and for introducing me to the world of research. I will always cherish my interactions with him for several important things that I have learned about research and life as a whole. This work would not have been possible without his advice and encouragement at each stage of my graduate student life. I am also highly grateful to Professor Christopher Wyatt and Professor Patrick Schaumont for agreeing to be a part of my Masters committee.

I will always treasure my interaction with team-mates at PROACTIVE research lab for their help and useful suggestions.

I am grateful to Craig Borden and Alec Shen for giving me the opportunity to intern at Qualcomm Inc. and work on interesting DFT and digital design projects.

I want to thank Claudia Angelini for giving me challenging opportunities in the area of DFT during my tenure at ST Microelectronics.

I want to thank my parents, sister and all family members and friends for their words of encouragement, love and support.

Finally, I am grateful to God for always being there for me in my endeavor to become a better person.

December 1, 2009 Sandesh Prabhakar
Contents

List of Figures vii

List of Tables ix

1 Introduction 1

2 Background 5
   2.1 Digital Integrated Circuit (IC) Development Process 5
   2.2 Need for Silicon Debug 7
   2.3 Silicon Debug 7
   2.4 Static Logic Implications 10
   2.5 SAT-based Boolean Constraint Propagation (BCP) 15
   2.6 Restoration Ratio and Restoration Percentage 17
   2.7 Data Compression Codes 17
   2.8 Frequency Directed Run-length (FDR) Code 18
   2.9 Source Transformation 19
   2.10 Diagnosis and Compression Quality Metrics 19
   2.11 Entropy 20
   2.12 Average Hamming Distance and Toggling Percentage 21

3 Non-multiplexed Trace Selection and State Restoration 22
   3.1 Problem Formulation 22
   3.2 New Trace Signal Selection 22
   3.3 State Restoration from Traced Signals 25
   3.4 Experimental Results 27
   3.5 Summary 38

4 Multiplexed Trace Selection and State Restoration 39
   4.1 Problem Formulation 39
   4.2 Implication-based Correlation 40
   4.3 Modified State Restoration Algorithm 46
4.4 Multi-node implication-based trace list pruning ................. 47
4.5 Experimental Results .................................................. 50
4.6 Summary ................................................................. 54

5 Trace Compression using Source Transformation over FDR codes 55
5.1 The Proposed Approach ............................................... 55
5.2 The Compression Hardware .......................................... 60
5.3 Experimental Results .................................................. 63
5.4 Summary ................................................................. 68

6 Conclusion and Future Work ........................................... 69

Bibliography ................................................................. 71
List of Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Digital IC Development Process</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Design Implementation Hierarchy</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Trace buffer-based Silicon Debug Architecture</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Implication Graph Example</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>Sequential Circuit Fragment</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>Direct Implications for f=1</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>Adding Indirect Implications for f=1.</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>Adding Extended Backward Implications for f=1</td>
<td>15</td>
</tr>
<tr>
<td>2.9</td>
<td>Boolean Constraint Propagation</td>
<td>16</td>
</tr>
<tr>
<td>2.10</td>
<td>An FDR coding example</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>State Restoration- Forward and Backward Learning</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>TR_{all} vs. TB width</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>TR_{FFs} vs. TB width</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Restoration % (all) vs. vector # for TB width=8</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Restoration % (FFs) vs. vector # for TB width=8</td>
<td>32</td>
</tr>
<tr>
<td>3.6</td>
<td>Restoration % (all) vs. vector # for TB width=16</td>
<td>32</td>
</tr>
<tr>
<td>3.7</td>
<td>Restoration % (FFs) vs. vector # for TB width=16</td>
<td>33</td>
</tr>
<tr>
<td>3.8</td>
<td>Restoration % (all) vs. vector # for TB width=32</td>
<td>33</td>
</tr>
<tr>
<td>3.9</td>
<td>Restoration % (FFs) vs. vector # for TB width=32</td>
<td>34</td>
</tr>
<tr>
<td>3.10</td>
<td>Restoration % (all) vs. vector # for TB width=64</td>
<td>34</td>
</tr>
<tr>
<td>3.11</td>
<td>Restoration % (FFs) vs. vector # for TB width=64</td>
<td>35</td>
</tr>
<tr>
<td>3.12</td>
<td>Restoration % (all) vs. vector # for TB width=128</td>
<td>35</td>
</tr>
<tr>
<td>3.13</td>
<td>Restoration % (FFs) vs. vector # for TB width=128</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Balance Factor</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Implication-based Correlation</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Modified State Restoration</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Total Restoration Percentage (TR_{all}) for Benchmark Circuits</td>
<td>52</td>
</tr>
<tr>
<td>5.1</td>
<td>Transforming T to T_{diff}</td>
<td>56</td>
</tr>
<tr>
<td>5.2</td>
<td>Transforming T_{diff} to T_{order}</td>
<td>58</td>
</tr>
</tbody>
</table>
5.3 Entropy for various Benchmark circuits (N=32) .......................... 59
5.4 Transforming T_{order} to T_{reversal} .............................................. 60
5.5 Cmax % for various benchmark circuits (N=32) ............................. 60
5.6 Trace Compressor Architecture .................................................... 62
5.7 Actual Compression % for various benchmark circuits (N=32) ........ 64
5.8 Actual Compression % for various benchmark circuits (N=64) ........ 66
5.9 Actual Compression % for various benchmark circuits (N=128) ...... 66
5.10 Entropy for various Benchmark circuits (N=64) ............................ 67
5.11 Entropy for various Benchmark circuits (N=128) ............................ 67
List of Tables

3.1 Experimental Results (Trace Buffer Width=8) ........................................ 29
3.2 Experimental Results (Trace Buffer Width=16) ................................. 29
3.3 Experimental Results (Trace Buffer Width=32) ................................. 29
3.4 Experimental Results (Trace Buffer Width=64) ................................. 30
3.5 Experimental Results (Trace Buffer Width=128) .............................. 30
3.6 Experimental Results -Poor (P) vs. our Algo (A) for first 10 vectors .... 30
3.7 Forward Learning (H₁) vs. Backward Learning (H₂) (N=8) ............. 36
3.8 Forward Learning (H₁) vs. Backward Learning (H₂) (N=16) .......... 36
3.9 Forward Learning (H₁) vs. Backward Learning (H₂) (N=32) .......... 37
3.10 Forward Learning (H₁) vs. Backward Learning (H₂) (N=64) ........ 37
3.11 Forward Learning (H₁) vs. Backward Learning (H₂) (N=128) ....... 37
4.1 Results (H₁: non-mux+FL+PI, H₂: non-mux+FL+BL+PI+PO) .......... 52
4.2 Results (H₂: non-multiplexed, H₃: multiplexed) ............................... 53
4.3 Results (H₃: multiplexed, H₄: H₃+multi-node impl.) ......................... 53
4.4 Results (No Trace, Random vs. our Algorithms) .............................. 53
5.1 Average Toggling Percentage .......................................................... 57
5.2 Compression results (#Vectors=1000) ........................................... 65
5.3 GZIP vs. Our approach ................................................................. 65
5.4 Area Overhead of Compressor ....................................................... 66
Chapter 1

Introduction

Pre-silicon verification and post-silicon manufacturing test play an important role in guaranteeing the integrated circuit (IC) product quality. Pre-silicon verification techniques like formal verification and simulation-based functional validation are commonly used to verify if the design implementation matches the specification. Manufacturing test is used to screen out the fabrication defects that affect the IC behavior. With the growing complexity of system-on-chip (SOC) designs, pre-silicon verification and manufacturing test are becoming more challenging than ever. Many functional bugs may remain undetected after pre-silicon verification and several defects may escape the manufacturing test. The increasing demand for shorter time-to-market has made the discovery of these undetected bugs and defects even more critical. In-system silicon debug techniques using design for debug (DFD) hardware are employed to identify the root cause of first silicon failures.

There are several working groups who are involved in the standardization of on-chip debug processes and instruments [52]. Two existing types of silicon debug techniques are: scan-based and trace buffer-based. In the scan-based approach, the design’s existing test structure comprising of JTAG and scan chains are re-used. The captured data from the internal state elements corresponding to specific triggering events are off-loaded (or dumped) through the scan chains. In [6] and [7], the authors discuss the identification of failing state elements from the scan dump data using post-processing algorithms. In [5], the backward and forward logic implications of the scan-dump values are used to restore more circuit gate values. However, many complex, non-repeatable bugs may only manifest themselves after long period of operations, making repeated scan dump-based debug approach costly and cumbersome for silicon debug. A trace buffer-based technique is employed to acquire continuous data wherein an embedded logic analyzer (ELA) [8] is used to sample internal signal data into on-chip trace buffers. Then, state restoration software [9] reconstructs the internal signal values from the off-loaded trace data.
The amount of data which can be acquired by the trace buffer is limited by the buffer’s depth and width. The buffer’s depth limits the number of samples that can be stored and the width limits the number of trace signals which can be sampled and recorded in each clock cycle [3]. Methods for ELA design improvement were proposed in [4], [8] and [10]-[12]. The area of the trace buffer memory is limited. Thus it is highly desirable to select the best trace signals which can maximize the restoration of missing signal values. In [1], [2] and [3], trace selection and state restoration algorithms were proposed which uses restorability metrics that consider both the topology and behavior of logic gates. In [25] an interconnection fabric design for tracing signals was proposed which comprised of a multiplexer network and a non-blocking concentration network. The motivation in [25] was that it is not necessary to observe uncorrelated signals concurrently. However, in many cases it is essential to observe uncorrelated signals concurrently since these uncorrelated signals do not imply each other. On the other hand, it may not be necessary to trace highly correlated signals concurrently since one signal might restore (imply) the other. Another disadvantage of [25] is the area overhead of the multiplexer tree and the crossbar switches.

Trace compression techniques can be used to increase the storage efficiency of the trace buffer. Such techniques were proposed in [13]-[15] to increase the number of trace signal samples. In [14], dictionary-based algorithms such as Lempel-Ziv (LZ77) and its variants LZ78, LZW and word-based dynamic Lempel-Ziv (WDLZW) are used. In [39], a LZ-based data compression algorithm was used for program trace compression. A drawback of using a complete dictionary is that the size of the dictionary can become very large, resulting in too much overhead for the on-chip compressor.

Contributions of this thesis:

Our main objectives are to maximize the restoration of missing internal signals using a minimum number of trace signals and to increase the storage efficiency of the trace buffer. The first contribution of this thesis is a new algorithm using a logic implication-based [16] learning approach to intelligently select the trace signals. We favor selecting
those signals which contain more implications that are not implied by other signals. We show that our trace selection method is efficient and is able to achieve better restoration than other techniques. Then, based on the values of the traced signals during silicon debug, we introduce an algorithm which uses a SAT-based multi-node implication engine to restore the values of untraced signals across multiple time-frames. Experimental results for sequential benchmark circuits showed that the proposed approach selects the trace signals effectively, giving a high restoration percentage compared with other techniques.

Our second contribution is another algorithm which uses the correlation between the forward and backward implications of flip-flops (trace signals) across two consecutive time-frames as a parameter to intelligently select two sets of trace signals. The two sets of trace signals are then multiplexed such that the first set is traced during even time-frames and the second set is traced during odd time-frames. Our primary motivation is that the signal pairs with high implication-based correlation between them need not be traced concurrently. As a result, we can effectively trace twice as many signals with the same trace buffer width. We show that our new trace selection method is efficient and gives a better restoration percentage compared to previous techniques. We also propose a SAT-based greedy heuristic to prune the selected trace signal list, thus considering some corner cases where multi-node implications play a major role during state restoration. This further improves the restoration percentage for a few circuits.

Trace compression techniques [13-15] are used to increase the storage efficiency of the trace buffer. Golomb code [31] and FDR code [32] belong to the variable-to-variable category of lossless data compression codes. To obtain a high compression percentage with minimal hardware overhead, we propose enhancements for FDR codes that can be implemented with minimal hardware overhead. We implement source transformation functions on the captured data before encoding the data using the FDR codes. Source transformation functions convert the captured data into reduced entropy data-set and hence improve the achievable compression percentage. We show that our approach achieves better compression percentage compared to dictionary-based techniques. Moreover, the area overhead of our trace compressor is less compared to dictionary-based codes and yields up to 3X improvement in the diagnostic capability. This is our third contribution.
Organization of this thesis:

The rest of the thesis is organized as follows:

- Chapter 2: This chapter describes various silicon debug techniques used currently in the industry to identify the root cause of first silicon failures. It surveys the various trace selection, trace compression and state restoration schemes proposed in literature. This chapter also gives an overview of static logic implications and various compression techniques and defines parameters used to evaluate the quality of trace selection and trace compression.

- Chapter 3: This chapter discusses the proposed approach for non-multiplexed trace buffer signal selection and introduces a SAT-based heuristic for state restoration.

- Chapter 4: This chapter discusses the proposed approach for multiplexer-based trace signal selection and introduces a SAT-based greedy heuristic for pruning the selected trace signal list further. It also explains the state restoration algorithm for the new multiplexer-based scheme.

- Chapter 5: This chapter discusses the proposed source transformation functions for Frequency-Directed Run-Length (FDR) codes. It also presents hardware implementation scheme for the proposed trace data compression scheme.

- Chapter 6: This chapter concludes the thesis.
Chapter 2

Background

This chapter introduces the various steps involved in the development of a digital integrated circuit (IC). It also gives an overview of the various silicon debug techniques used currently in the industry and briefly introduces the trace selection, trace compression and state restoration schemes proposed in literature. Finally, it describes concepts such as static logic implications, SAT-based Boolean constraint propagation, dictionary and adaptive/dynamic code-based compression techniques, entropy, hamming distance and Burrows-Wheeler source transformation that we use in this thesis.

2.1 Digital Integrated Circuit (IC) Development Process

Figure 2.1 shows the steps involved in the development of a digital integrated circuit (IC) [33]. We explain each of these steps below:

*Design Specification:* Design specification is defined as the formulation of a VLSI device requirement in the form of a design documentation or behavioral reference model based on a customer or project need.

*Design Implementation:* Design implementation is a process of transforming a higher level description of a design into a lower level description. Figure 2.2 shows the steps involved in the design implementation. Starting from a design specification, a behavioral (architecture) level description is developed in a hardware description language (HDL) or as a C program. The design is then described at the register-transfer level (RTL). The RTL is then synthesized to produce the gate-level design of the circuit. Finally, the gate-level design is transformed to a physical-level description in order to obtain the physical placement and interconnection of the transistors in the VLSI device prior to fabrication. Verification is important at each stage of the design implementation to ensure that the functionality of the final design meets the design specifications including the timing and operating frequency specifications.

*Design Verification:* Design verification, also known as pre-silicon verification is a predictive analysis to check the correctness of the design implementation against its
specification. When a design error is found, modifications to the design are necessary and design verification must be repeated. Two commonly used verification techniques are: 1) formal verification and 2) simulation.

_Design Fabrication:_ Design fabrication is a multiple-step sequence of photographic and chemical processing steps during which electronic circuits are gradually created on a wafer made of pure semiconducting material.

_Manufacturing Test:_ Manufacturing test is a test which is applied to each fabricated circuit to detect physical defects such as shorts and opens and timing defects. The test procedure is based on the design specification and fault models associated with the implementation technology.

---

![Figure 2.1: Digital IC development process](image1)

![Figure 2.2: Design implementation hierarchy](image2)
2.2 Need for Silicon Debug

The time required for design verification in the pre-silicon stage is increasing with the growing complexity of integrated circuits. Insufficient verification may fail to detect design errors. Moreover, the accuracy of circuit models is inadequate to ensure the first silicon to be error free. Therefore, design verification has a definite impact on time-to-market and hence is economically significant, and it is important that the undetected design bugs are fixed as soon as the first silicon is available. In recent years, silicon debug has emerged as a key technique to detect and locate design errors in silicon. Even when the design is correct, defects may be introduced during the fabrication due to impurities. In other words, not all chips manufactured from the same design may be defect-free. Manufacturing test attempts to capture those defective parts. However, the defect may be located in a corner-case region for which there are few or no tests. In such cases, the chip may pass the manufacturing test and be shipped as if it was good. When put into a system, the chip may malfunction whenever the defect is exercised. In this case, silicon debug is likewise necessary to detect and locate the source of the problem.

2.3 Silicon Debug

In-system silicon debug techniques are employed to identify the root cause of first silicon failures. Silicon debug can be defined as the process of finding, locating and identifying design bugs in the post-silicon phase [18]. Three techniques used for silicon debug are: 1) physical probing, 2) scan-based and 3) trace buffer-based. Silicon debug can be divided into two main steps: data acquisition and analysis. A type of physical probing technique which uses time-resolved photo emission [20] is widely used to acquire circuit data for failure analysis. However, the decreasing feature size and growing complexity of designs make this technique cumbersome for data acquisition. The debug methods based on internal scan chains have been used extensively for debugging complex digital ICs [19]. Scan-based debug concepts have emerged from the manufacturing test research. In the scan-based approach, the internal scan chains are reused wherein the captured data from the internal state elements corresponding to specific triggering events are off-loaded (or dumped) through the scan chains. In [6-7], the authors discuss post-processing algorithms.
which can be used to identify the failing state elements from the scan dump data. In [5] a method was proposed which utilizes backward and forward logic implications of the scan-dump values to restore more circuit gate values. However, many complex, non-repeatable bugs may only manifest themselves after a long period of operations. Moreover, to complete a scan dump while continuing the real-time execution, it is necessary to double buffer the state elements in the scan chain, thus leading to unacceptable area penalty. This makes repeated scan dump-based debug approach costly and cumbersome for silicon debug. The trace buffer-based approach is a complementary technique which can be used to acquire continuous data. This debug technique has been influenced by software debugging used in embedded systems [22]. An embedded logic analyzer (ELA) [8] is used for sampling internal signal data into on-chip trace buffers. This is followed by a post processing stage [9] wherein the sampled data is off-loaded for analysis to reconstruct internal signal values and identify functional bugs.

![Figure 2.3: Trace buffer-based silicon debug architecture](image)

Figure 2.3 shows an example of trace buffer-based debug architecture. The trace signal interface shown in the figure is used to transfer the trace signals to on-chip trace buffers and/or off-chip trace ports for diagnosis. A trace control unit controls the start and stop of
the tracing, in which the control mechanism can be configured through JTAG interface. The trace buffer-based debug methods can be broadly classified as: special-purpose or generic. The special-purpose method [23-24] is applicable to embedded processors. The generic method is applicable to any type of custom SOCs [11, 17]. Another classification is based on centralized tracing or distributed sampling. In centralized tracing, one trace buffer is used per SOC (with different interconnect topologies between the embedded cores and the trace buffer) [17, 23, 24]. In distributed sampling the trace buffers are allocated to individual cores [11]. Regardless of the above classification, the primary benefit of the trace buffer-based method is that it provides real-time visibility to the circuit under debug (CUD) and enables in-field at-speed debug. The amount of data which can be acquired by the trace buffer is limited by two parameters: the buffer’s depth and width. The former limits the number of samples that can be stored and the latter limits the number of trace signals which can be sampled and recorded in each clock cycle [3]. The space of the available trace buffer memory is normally very limited and hence only a small number of internal signals can be observed together real-time.

The selection of the critical trace signals which can maximize the restoration of the missing signal values is highly desirable. In [1-3], algorithms were proposed for trace signal selection and state restoration using restorability metrics that consider both the topology and behavior of logic gates. Ko et al. [3] used the state restoration concept to select the best signals that can eventually restore maximum number of other signals and hence improve the observability of the circuit under debug (CUD). Liu et al. [1] proposed refinements to take care of a few limitations in [3]. The restorability formulation in [1] and [3] is probabilistic in nature. The restorability calculation is computationally intensive since the restorability for all flip-flops is recomputed for every iteration.

The interconnection fabric used to interconnect trace signals to the trace buffers and/or trace ports involves non-trivial area overhead. The existing solutions [26-28] use pipelined multiplexer (MUX) trees for the interconnection fabric design. However, these ad-hoc techniques limits the visibility to the circuit under debug (CUD) since any signal going through the same multiplexer cannot be observed concurrently. In [25], an interconnection fabric design was proposed to take care of the above problem. It consisted of two main parts: 1) a MUX network that connects those mutually-exclusive
tapped signals, which can be designated by designers and/or extracted automatically based on structural analysis; 2) a non-blocking concentration network that is able to transfer any $m$ out of $n$ inputs ($m \leq n$) to the trace buffers/ports. The motivation in [25] was that it is not necessary to observe uncorrelated signals concurrently. However, in many cases it is essential to observe uncorrelated signals concurrently since these uncorrelated signals do not imply each other. On the other hand, it may not be necessary to trace highly correlated signals concurrently since one signal might restore (imply) the other. Another disadvantage of [25] is the area overhead of the multiplexer tree and the crossbar switches.

Trace compression [13-15] is another sought-after method to increase the number of trace signal samples, thus increasing the storage efficiency of the trace buffer. In [14] dictionary-based algorithms (LZ77 and its variants) were used. A dedicated fast parallel search engine called content addressable memory (CAM) was used in [14] in order to perform fast search in hardware between the incoming symbol and the dictionary entries. The main limitation of dictionary-based method proposed in [6] was the large area overhead due to different content-addressable memory (CAM) sizes. A LZ-based data compression algorithm was used in [39] for program trace compression, however the area overhead was large.

In the subsequent sections, we define and explain a few concepts that we use in this thesis.

### 2.4 Static Logic Implications

Let us consider a circuit with $n$ gates. Logic implications determine the effect of assigning logic values (0 or 1) to one or more gates in the circuit. The implications are stored using a directed implication graph $G (V, E)$ where $V$ (vertices) $c$ the set of $2n$ nodes corresponding to both value assignments (0 and 1) and $E$ (edges) $c$ single-node implications. For sequential circuits, each edge is annotated with an integer weight $w$ that indicates the number of time frames that this implication spans. For example, consider an AND gate and its implication graph, shown in Figure 2.4. The AND gate has three signals, $a$, $b$, and $c$ and the associated implication graph has six nodes. An edge in the
implication graph indicates the implication relationship and the annotated weight indicates the number of time-frames spanned by the implication. From Figure 2.4, clearly \( c=1 \) has two implications: \( b=1 \) and \( a=1 \) and the edge weight is 0.

Static logic implications can be sub-divided into direct, indirect and extended backward implications [16, 33]. Indirect and extended backward implications use logic simulation as well as the contra-positive and transitive laws extensively. These learned implications are thus non-trivial.

![Implication graph example](image)

We define a few terms and concepts for single-node implications which we use for the discussion:

a) \([N,v,t]\): Assign logic value \( v \) to gate \( N \) in time frame \( t \).

b) \([N,v,t_1] \rightarrow [M,w,t_2]\): Assigning logic value \( v \) to gate \( N \) in time frame \( t_1 \) would imply a logic value \( w \) to gate \( M \) in time frame \( t_2 \).

c) \( DI[N,v,t] \): Set of direct implications resulting from assigning node \( N \) in time frame \( t \) to value \( v \). For \( t=0 \), \( DI[N,v,t] \) is simply represented as \( DI[N,v] \) or \( DI[N=v] \).

d) \( IND[N,v,t] \): Set of indirect implications resulting from assigning node \( N \) in time frame \( t \) to value \( v \). For \( t=0 \), \( IND[N,v,t] \) is simply represented as \( IND[N,v] \) or \( IND[N=v] \).

e) \( EB[N,v,t] \): Set of extended backward implications resulting from assigning node \( N \) in time frame \( t \) to value \( v \). For \( t=0 \), \( EB[N,v,t] \) is simply represented as \( EB[N,v] \) or \( EB[N=v] \).
f) \( Impl[N,v,t] \): Set of single-node implications resulting from assigning node \( N \) in time frame \( t \) to value \( v \). For \( t=0 \), \( Impl[N,v,t] \) is simply represented as \( Impl[N,v] \) or \( Impl[N=v] \). Note that \( Impl[N,v,t] = DI[N,v,t] \cup IND[N,v,t] \cup EB[N,v,t] \).

g) Transitive law: If \([M,w] \rightarrow [N,v,t_1] \) AND \([N,v] \rightarrow [L,y,t_2] \), then \([M,w] \rightarrow [L,y,t_1+t_2] \).

In set notation, if \([N,v,t_1] \in Impl[M,w] \) and \([L,y,t_2] \in Impl[N,v] \), then \([L,y,t_1+t_2] \in Impl[M,w] \).

h) Contrapositive law: If \([M,w] \rightarrow [N,v,t] \), then \([N,v'] \rightarrow [M,w',-t] \). In set notation, if \([N,v,t] \in Impl[M,w] \), then \([M,w',-t] \in Impl[N,v'] \).

i) Conflicting assignments: If \([M,w] \rightarrow [N,v,t] \) AND \([M,w] \rightarrow [N,v',t] \), then \([M,w] \) is an impossible setting. This means that \( M \) is a constant node holding the value \( w' \) permanently.

\[ \text{Figure 2.5: Sequential circuit fragment} \]

We illustrate the direct, indirect and extended backward implications using the following example. Consider gate \( f=1 \) in the sequential circuit fragment shown in Figure 2.5. Let \( Impl[f=1]=\emptyset \) initially:

1. Direct Implications: In Figure 2.5, \( g \) and \( k \) are directly connected to gate \( f \). Clearly, \( f=1 \) would directly imply \( g=k=1 \). Moreover, \( f=1 \) has two implications: \( d=1 \) and \( e=1 \). Let \( DI[f=1] \) denote the set containing direct implications for \( f=1 \). Thus, \( DI[f=1]=\{(f,1,0), (g,1,0), (k,1,0), (d,1,0), (e,1,0)\} \). Similarly, the direct
implications for $g=1$, $DI[g=1]=\{(g,1,0),(j,1,0),(f,1,0)\}$. These implications are stored in the form of a graph, where each node represents a gate (with a logic value). A directed edge between two nodes represents an implication, and a weight along an edge represents the relative time frame associated with the implication. The graph shown in Figure 2.6 represents a portion of direct implications for $f=1$ in this example. The complete set of implications resulting from setting $f=1$ can be obtained by applying transitive law and traversing the graph rooted at node $f=1$. Computing the set of all nodes reachable from this root node ($f=1$) (transitive closure on $f=1$) would return the set $DI[f=1]$. Thus, the complete set of direct implications using the implication graph shown in the figure for $f=1$ is $DI[f=1]=\{(f,1,0),(d,1,0),(e,1,0),(g,1,0),(k,1,0),(j,1,0),(c,1,-1)\}$. After learning direct implications, $Impl[f=1]=Impl[f=1] \cup DI[f=1]$.

2. **Indirect implications:** Note that neither $j=1$ nor $k=1$ implies a logic value on gate $x$ individually. However, if they are taken collectively, they imply $x=1$. Thus, indirectly, $f=1$ would imply $x=1$. This is an indirect implication of $f=1$, and it can be computed by performing a logic simulation on the current set of implications of the root node on the circuit. In this example, by inserting the implications of $f=1$ into the circuit, followed by a run of logic simulation, $x=1$ would be obtained as a result. Thus, $IND[f=1]=\{(x,1,0)\}$ and $Impl[f=1]=Impl[f=1] \cup IND[f=1]$. This new implication is then added as an additional outgoing dashed edge from $f=1$ in the implication graph as shown in Figure 2.7. Another nontrivial implication that can be inferred from each indirect implication is based on the contrapositive law. Since $[f,1] \rightarrow [x,1,0]$, by contrapositive law, $[x,0] \rightarrow [f,0,0]$.

3. **Extended backward (EB) implications:** The unjustified implied nodes in the implication list can be used to learn more implications, known as extended backward implications for any single node. Using the same circuit shown in Figure 1 again, in the implication list of $f=1$, $d=1$ is an unjustified gate because none of $d$’s inputs has been implied to a logic value of 1. Thus, $d$ is a candidate for the application of EB implications. To obtain EB implications on $d$, a transitive closure is first performed for each of its unspecified inputs. In this case, $Impl[a=1]$ and $Impl[b=1]$ are first computed. The implications of $f=1$ are logic
simulated together with each of d’s unspecified input’s implication sets in turn, creating a set of newly found logic assignments for each input of the chosen unjustified gate. For this example, when the implications of (a=1) and (f=1) are simulated, the new assignments (set_a) found include (w,0,0) and (z,0,0). Similarly, for the combined implication set of (b=1) and (f=1), the new assignments (set_b) found include (y,0,0) and (z,0,0). All logic assignments that are not already in Impl[f=1] which are common to set_a and set_b are the EB implications. These new implications are added as new edges to the original node f=1. Thus, EB[f=1]={([z,0,0])} and Impl[f=1]=Impl[f=1]U EB[f=1]. In this running example, because (z,0,0) is common in set_a and set_b, it is a new implication. The corresponding new implication graph is illustrated in Figure 2.8, where the new implication is shown as a dotted edge.
2.5 SAT-based Boolean Constraint Propagation (BCP)

The implication graph stores only the single node implications, i.e., one node implying another node. For determining multi-node implications, i.e., a set of nodes together implying a single node, we use a SAT-based approach. We illustrate this method using an example. Every Boolean formula can be expressed in *conjunctive normal form (CNF).*

For the circuit shown in Figure 2.9 the CNF formula can be expressed as:

\[ (\neg a \lor b \lor c) \land (a \lor \neg c) \land (\neg b \lor \neg c) \land (\neg a \lor d) \land (\neg b \lor d) \land (a \lor b \lor \neg d) \]

Suppose we wish to determine the multi-node implications of the signal assignments \( b=0 \) and \( d=1 \). We use *Boolean Constraint Propagation (BCP)* to quickly identify the multi-node implications. First, set \( b=0 \) and \( d=1 \) in the above CNF formula. It gets simplified to:

\[ (\neg a \lor c) \land (a \lor \neg c) \land (1) \land (1) \land (1) \land (a) \]

The sixth clause has become a *unit clause* (a clause with one unassigned/free literal). Therefore \( a=1 \) is an implication. Next, set \( a=1 \) in the simplified CNF formula. It gets simplified to:

\[ (c) \land (1) \land (1) \land (1) \land (1) \land (1) \]

The first clause is now a unit clause, hence \( c=1 \) is an implication. Since all the other clauses are *satisfied* (clause evaluates to 1), we stop and conclude that \( \{a=1, c=1\} \) are the multi-node implications of \( b=0 \) and \( d=1 \). In other words, if \( S \) is the set containing the signal assignments \( b=0 \) and \( d=1 \), i.e. \( S = \{b=0, d=1\} \), then \( \text{BCP}(S) = \{a=1, c=1\} \) Note that
if any clause evaluates to 0 (i.e. unsatisfied) we conclude that a conflict has occurred and hence the given set of signal assignments $S$ do not have any multi-node implications, i.e., $BCP(S) = \phi$. For the same circuit, $\{d=1\}$ is the single-node implication of the signal assignment $S= \{a=1\}$.

![Figure 2.9: Boolean Constraint Propagation](image)

We define a few terms for single-node [16] and multi-node implications which we use in the following:

j) Impl[$N,v,t$]: Set of single-node implications resulting from assigning node $N$ in time frame $t$ to value $v$.

k) Impl[$N,t$]: Set of single-node implications resulting from assigning node $N$ in time frame $t$ to value $v$ and $\bar{v}$ respectively (i.e., Impl[$N,v,t$] $\cup$ Impl[$N,\bar{v},t$]).

l) $BCP(S)$: Set of multi-node implications resulting from assigning nodes $N_1, N_2, \ldots, N_n$ in time frame $t$ to values $v_1, v_2, \ldots, v_n$ respectively, where $S=\{N_i,v_i,t\}_{i=1,2,..,n}$

m) Impl$_f[N, t]$: Set of single-node forward implications in time frame $t$ resulting from assigning node $N$ in time frame $t$ to value $v$ and $v'$ respectively.

n) Impl$_b[N, t]$ : Set of single-node backward implications in time frame $t-1$ resulting from assigning node $N$ in time frame $t$ to value $v$ and $v'$ respectively.
2.6 Restoration Ratio and Restoration Percentage

In [1] and [3], a parameter called *restoration ratio (RR)* is used as an evaluation metric to measure the quality of the trace signals selected. It is calculated as:

\[
RR = \frac{N_{\text{traced (FFs)}} + N_{\text{restored (FFs)}}}{N_{\text{traced (FFs)}}}
\]  

(2.1)

Where \( N_{\text{traced (FFs)}} \) and \( N_{\text{restored (FFs)}} \) are the number of traced states and the number of restored states respectively across all the time-frames under consideration.

We define a new parameter called *restoration percentage* to measure the quality of trace signal selection more coherently. Total restoration percentage for all signals (including flip-flops) is calculated as:

\[
TR_{\text{all}} = \frac{N_{\text{traced (FFs)}} + N_{\text{restored (all)}}}{N_{\text{total (all)}}} \times 100
\]  

(2.2)

Where \( N_{\text{restored (all)}} \) is the number of restored signals in the circuit including PIs. \( N_{\text{total (all)}} \) is the total number of signals in the circuit.

The restoration percentage for only flip-flops is calculated as:

\[
TR_{\text{FFs}} = \frac{N_{\text{traced (FFs)}} + N_{\text{restored (FFs)}}}{N_{\text{total (FFs)}}} \times 100
\]  

(2.3)

Where \( N_{\text{total (FFs)}} \) is the total number of flip-flops in the circuit.

\( TR_{\text{all}} \) and \( TR_{\text{FFs}} \) are the total restoration percentages calculated across all the time-frames. Equations (2.2) and (2.3) can also be used to determine the individual restoration percentages for each time-frame.

2.7 Data Compression Codes

Lossless data compression codes are classified into four categories depending on whether the symbols have a fixed size or a variable size, and whether the code-words have a fixed size or a variable size. Dictionary-based codes belong to fixed-to-fixed codes. A drawback of using a complete dictionary is that the size of the dictionary can become
very large, resulting in too much overhead for the on-chip compressor. Huffman code [33, 41] and run-length code [33, 40] are fixed-to-variable and variable-to-fixed codes, respectively. Adaptive/dynamic statistical coding algorithms, such as Huffman coding [33, 41] can provide a greater compression ratio but implementing them in hardware can incur exorbitant real estate cost. Golomb code [31] and FDR code [32] belong to the variable-to-variable category. Both have evolved from the run-length code and are able to achieve greater compression. Since it was shown in [37] that FDR code is superior to Golomb code, we use FDR code in our work.

2.8 Frequency Directed Run-length (FDR) Code

Since our technique enhances FDR codes, we will briefly describe them. A detailed description is available elsewhere [32]. In FDR coding scheme, runs of zero’s are encoded as shown in Figure 2.10. Here, labels $A_i (i=1,2,\ldots)$ is used to represent a Group and the corresponding group prefix is shown in the column labeled Group prefix. The total number of code-words in a group $A_i$ is $2^i$. To encode a given run of zeros, a code-word is constructed by concatenating the group prefix and a tail. For example, the sequence 000001 (Run-length=5) is encoded as 1011 (Group prefix=10, tail=11). We will now illustrate a more complex example.

<table>
<thead>
<tr>
<th>Group</th>
<th>Run-length</th>
<th>Group prefix</th>
<th>Tail</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2</td>
<td></td>
<td>00</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>01</td>
<td>1001</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>$A_3$</td>
<td>6</td>
<td>10</td>
<td>000</td>
<td>110000</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td>001</td>
<td>110001</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>010</td>
<td>110010</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td>011</td>
<td>110011</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td>100</td>
<td>110100</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
<td>101</td>
<td>110101</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td>110</td>
<td>110110</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td></td>
<td>111</td>
<td>110111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 2.10: An FDR coding example
Example 1. Let the un-encoded sequence be $T = 0001\ 00001\ 1\ 00001\ 00001\ 001\ 0000001\ 001$. The run-lengths of zeroes in this un-encoded sequence are 3, 5, 0, 4, 4, 6, 2, 7, and 2. From Fig. 1 we obtain the encoded sequence, $T_E = 1001\ 1011\ 00\ 1010\ 1010\ 110000\ 1000\ 110001\ 1000$.

2.9 Source Transformation

Source transformation refers to the idea of transforming un-encoded data set $T$ into a new data set, $T'$, which is more amenable for compression. Burrows-Wheeler transform [38] (BWT) is a widely used transformation that, when used on top of run length encoding, can give very impressive compression ratio. Bzip2 employs BWT transforms.

2.10 Diagnosis and Compression Quality Metrics

We use the following parameters to evaluate the quality of trace compression obtained using our proposed approach. Diagnostic Resolution and Compression Percentage: For trace compression, we consider the trace data set as an $n \times N$ matrix, where $n$ is the total number of time-frames and $N$ is the total number of trace flip-flops. Each row represents the current state of the trace flip-flops. $T$ refers to the un-encoded sequence obtained from the $n \times N$ matrix by concatenating the $n$ rows. $T_E$ is used to refer to the encoded trace data sequence. Diagnostic Resolution ($D_R$) is defined by:

$$D_R = \frac{S_u}{S_e} = \frac{n \times N}{S_e}$$

(2.4)

where $S_u$ is the size of $T$ (i.e., the number of bits in the un-encoded sequence $T$) and $S_e$ is the size of $T_E$ (i.e., the number of bits in the encoded sequence $T_E$).

In other words, we can trace $N$ signals using an $N/D_R$ wide trace buffer. Thus, the diagnostic capability is improved by $D_R$ times.

Compression percentage ($C$) is defined by:

$$C = (1 - \frac{1}{D_R}) \times 100$$

(2.5)

For the un-encoded sequence of Example 1, $S_u = 42$ and $S_e = 38$. Hence, $D_R = S_u/S_e = 42/38 = 1.105$, and $C = (1 - 1/D_R) \times 100 = (1 - 1/1.105) \times 100 = 9.5\%$. 

19
2.11 Entropy

In information theory, entropy of a stream of data quantifies the overall information stored in that data. The entropy $E$, of trace buffer data is defined as:

$$E = - \sum_{i=0}^{n} p_i \log_2 (p_i)$$  \hspace{1cm} (2.6)

where $p_i$ is the probability of occurrence of a symbol $X_i$. Entropy can be used to compute the theoretical limits for achieving maximum compression using any encoding technique.

For a variable-to-variable encoding, the theoretical maximum compression is:

$$C_{\text{max}} = \frac{(S_l - E)}{S_l}$$  \hspace{1cm} (2.7)

where $E$ is the entropy and $S_l$ is the average symbol length defined as:

$$S_l = \sum_{i=0}^{n} p_i \cdot |X_i|$$  \hspace{1cm} (2.8)

where $p_i$ is the probability of occurrence of the symbol $X_i$ and $|X_i|$ is the length of $X_i$.

Clearly from Equation (2.7), the maximum compression is inversely proportional to the entropy. The computation of entropy and theoretical maximum compression is illustrated in the example below.

**Example 2.** Let us consider the un-encoded sequence $T$ of Example 1. Let $t_i$ be the total number of times run-length $i$ occurs. For our example, $t_0=1$, $t_1=0$, $t_2=2$, $t_3=1$, $t_4=2$, $t_5=1$, $t_6=1$, $t_7=1$, $t_8=0$. Let $R$ be the total number of run-lengths. Thus, $R=\sum_{i=0}^{8} t_i=9$. The probability of occurrence of run-length $i$ is $p_i=t_i/R$. Thus, $p_0=1/9=0.11$, $p_1=0$, $p_2=2/9=0.22$, $p_3=1/9=0.11$, $p_4=2/9=0.22$, $p_5=1/9=0.11$, $p_6=1/9=0.11$, $p_7=1/9=0.11$, $p_8=0$. From Equation (2.6), the entropy is $E=\sum_{i=0}^{8} p_i \log_2 (p_i)= - (0.11 \log_2 (0.11) + 0.22 \log_2 (0.22) + 0.11 \log_2 (0.11) + 0.22 \log_2 (0.22) + 0.11 \log_2 (0.11) + 0.11 \log_2 (0.11) + 0.11 \log_2 (0.11)) = 2.79$. The length of run-length $i$ is $|X_i|=i+1$ if the run-length sequence end with a 1, otherwise $|X_i|=i$. For example, 001 has a run-length of 2, but the size is 3 including the 1 at the end. However, if we have an un-encoded sequence which ends with a zero, say 001 0001 00, the size of the last run-length will be 2 since it does not have a terminating 1. For our example, from Equation (2.8) the average symbol length $S_l=\sum_{i=0}^{8} p_i \cdot |X_i| = (p_0 \cdot |X_0| + p_2 \cdot |X_2| + p_3 \cdot |X_3| + p_4 \cdot |X_4| + p_5 \cdot |X_5| + p_6 \cdot |X_6| + p_7 \cdot |X_7|) = (0.11 \cdot 1 + 0.22 \cdot 3 + 0.11 \cdot 4 + 0.22 \cdot 5 + 0.11 \cdot 6 + 0.11 \cdot 7 + 0.11 \cdot 8) = 4.62$. Thus, from Equation (2.7) the theoretical maximum compression, $C_{\text{max}} = (S_l - E)/(S_l) = (4.62 - 2.79)/(4.62) = 0.396$, i.e., 39.6%.
2.12 Average Hamming Distance and Toggling Percentage

The Hamming distance, $H$, between two strings of equal length is the number of positions at which the corresponding symbols are different. For the $n \times N$ matrix, let the Hamming distance between two successive rows $k$ and $k+1$ be $H_k$. Note that $0 \leq H_k \leq N$, where $N$ is the total number of trace flip-flops. This means that $H_k$ trace flip-flops out of the total $N$ trace flip-flops toggle between the successive rows $k$ and $k+1$.

The average Hamming distance for the trace data set represented by the $n \times N$ matrix is given by:

$$H_{\text{avg}} = \frac{\sum_{i=1}^{n-1} D_i}{n-1}$$

(2.9)

The average toggling percentage for the trace data set represented by the $n \times N$ matrix is given by:

$$TP_{\text{avg}} = \frac{H_{\text{avg}}}{N} \times 100$$

(2.10)
Chapter 3
Non-multiplexed Trace Selection and State Restoration

This chapter is organized as follows. Section 3.1 formulates the problem. Section 3.2 discusses the proposed approach for non-multiplexed trace signal selection. Section 3.3 introduces the algorithm used for state restoration. Section 3.4 reports experimental results, and Section 3.5 summarizes the observation.

3.1 Problem Formulation

Let $G$ represent the set of all gates in the circuit and $S$ represent the set of trace signals to be selected. We define the problem statement as follows: Find the smallest subset of signals $S \subseteq G$ such that $\forall$ legal valuations $r$ of $S$, the values of the signals in $G-S$ can be restored.

In other words, our main objective is to maximize the restoration of missing internal signals using a minimum number of trace signals. In general, if we know all the primary input (PI) and current internal state element (flip-flop) values, it is possible to determine all the internal signal values through simple logic simulation. However, since the trace buffer capacity is limited in large designs, it is not possible to trace all the flip-flops. Hence, our objective is to select the best subset of flip-flops $f_i, i=1,2,...,N$ as trace signals such that most of the missing internal signal values can be restored. Throughout the paper we will assume that all PI values are known.

3.2 New Trace Signal Selection

In our trace signal selection algorithm, we use the number of single node implications (for both 0 and 1 assignment) per flip-flop $N_{f_i, i=1,2,...,N}$ as our trace signal selection restorability metric. First, we determine direct, indirect and extended backward implications [16] across a combinational time-frame and store them in an implication
graph. Next, we order the flip-flops $f_i$, where $i = 1, 2, ..., N$ in the descending order of the total number of implications for both 0 and 1 assignment and term it as set $O_i$. Let us assume that the $PI_k, i = 1, 2, ..., p$ ($p$ primary inputs) values are known. We define **checked implications** $CI(f_k)$ of a flip-flop $f_k$ in the ordered set $O_i$ as those implications which are also implied by the primary inputs in the set $U_{i\in 1,2,...,p} Impl[PI_i,0]$ and other flip-flops in the set $U_{i\in 1,2,...,k-1} Impl[f_i,0]$. In other words,

$$CI(f_k) = Impl[f_k,0] \cap \left((U_{i\in 1,2,...,k-1} Impl[f_i,0]) \cup (U_{i\in 1,2,...,p} Impl[PI_i,0]))\right)$$  

(3.1)

We define the **unchecked implications** $UI(f_k)$ of a flip-flop $f_k$ in the set $O_i$ as those implications which are not implied by those in the set $(U_{i\in 1,2,...,k-1} Impl[f_i,0]) \cup (U_{i\in 1,2,...,p} Impl[PI_i,0])$. In other words,

$$UI(f_k) = Impl[f_k,0] - CI(f_k)$$

(3.2)

To prune the ordering further, we remove the **checked implications** corresponding to each flip-flop from the sorted list and order them again in the descending order of number of **unchecked implications** and term it as set $O_f$. Based on a given trace buffer width $n < N$, where $N$ is the total number of flip-flops, we select the first $n$ flip-flops from the set $O_f$ as our trace signals. The basic idea is that the number of internal signals which can be restored using a flip-flop (trace signal) is directly proportional to the number of **unchecked implications** of that flip-flop. Hence, by ordering the flip-flops on the basis of number of **unchecked implications**, we aim to select the flip-flops which yield the best restorability individually. We will illustrate the trace signal selection method using the following example. Let us consider a circuit with one primary input $PI_1$, three flip-flops $f_i, f_2, f_3$ and six gates $g_1, g_2, g_3, g_4, g_5, g_6$. Suppose we obtain the following information from the implication graph:

$Impl[PI_1,0,0]: \{PI_1=0, g_6=1\}$
$Impl[PI_1,1,0]: \{PI_1=1, g_6=0\}$
$Impl[f_1,0,0]: \{f_1=0, g_1=1, g_2=1, g_3=0, g_5=1\}$
$Impl[f_1,1,0]: \{f_1=1, g_1=1, g_2=0, g_3=0, g_4=1\}$
$Impl[f_2,0,0]: \{f_2=0, g_1=1, g_2=1, g_3=1\}$
$Impl[f_2,1,0]: \{f_2=1, g_1=1, g_2=0, g_3=1, g_4=1\}$
$Impl[f_3,0,0]: \{f_3=0, g_1=0, g_3=1\}$
CI(f1,1,0): {f1=1, g1=1, g2=0, g3=0}

Thus, we get:

\[ \text{Impl}[PI_1,0]=\text{Impl}[PI_1,0] \cup \text{Impl}[PI_1,1,0]: \{PI_1=0, PI_1=1, g_6=0, g_6=1 \} \]

\[ \text{Impl}[f_1,0]=\text{Impl}[f_1,0,0] \cup \text{Impl}[f_1,1,0]: \{f_1=1, f_1=0, g_1=1, g_2=1, g_2=2, g_3=0, g_4=1, g_5=1 \} \]

\[ \text{Impl}[f_2,0]=\text{Impl}[f_2,0,0] \cup \text{Impl}[f_2,1,0]: \{f_2=1, f_2=0, g_1=1, g_2=1, g_2=0, g_3=1, g_4=1 \} \]

\[ \text{Impl}[f_3,0]=\text{Impl}[f_3,0,0] \cup \text{Impl}[f_3,1,0]: \{f_3=1, f_3=0, g_1=1, g_1=0, g_2=0, g_3=1, g_4=0 \} \]

\[ N_{f1}=|\text{Impl}[f_1,0]|=8, N_{f2}=|\text{Impl}[f_2,0]|=7, N_{f3}=|\text{Impl}[f_3,0]|=7 \]

On ordering the flip-flops in the descending order of number of implications for both 0 and 1 assignment, we get \( O_i = \{f_1, f_2, f_3\} \). Since \( f_1 \) has the most number of implications, we will start with this flip-flop. Then we remove the checked implications from \( f_1, f_2 \) and \( f_3 \). Using Equations (3.1) and (3.2), we get:

\[ CI(f_1)=\text{Impl}[f_1,0] \cap \text{Impl}[PI_1,0]=\emptyset \]

\[ UI(f_1)=\text{Impl}[f_1,0]-CI(f_1): \{f_1=1, f_1=0, g_1=1, g_2=1, g_2=0, g_3=0, g_4=1, g_5=1 \} \]

\[ CI(f_2)=\text{Impl}[f_2,0] \cap (\text{Impl}[f_1,0] \cup \text{Impl}[PI_1,0]): \{g_1=1, g_2=1, g_2=0, g_3=1 \} \]

\[ UI(f_2)=\text{Impl}[f_2,0]-CI(f_2): \{f_2=1, f_2=0, g_3=1 \} \]

\[ CI(f_3)=\text{Impl}[f_3,0] \cap (\text{Impl}[f_2,0] \cup \text{Impl}[f_1,0] \cup \text{Impl}[PI_1,0]): \{g_1=1, g_2=0, g_3=1 \} \]

\[ UI(f_3)=\text{Impl}[f_3,0]-CI(f_3): \{f_3=1, f_3=0, g_1=0, g_4=0 \} \]

Next, we set the implications to be the unchecked ones:

\[ \text{Impl}[f_1,0]=UI(f_1): \{f_1=1, f_1=0, g_1=1, g_2=1, g_2=0, g_3=0, g_4=1, g_5=1 \} \]

\[ \text{Impl}[f_2,0]=UI(f_2): \{f_2=1, f_2=0, g_3=1 \} \]

\[ \text{Impl}[f_3,0]=UI(f_3): \{f_3=1, f_3=0, g_1=0, g_4=0 \} \]

\[ N_{f1}=|\text{Impl}[f_1,0]|=8, N_{f2}=|\text{Impl}[f_2,0]|=3, N_{f3}=|\text{Impl}[f_3,0]|=4 \]

On ordering the flip-flops in the descending order of number of unchecked implications for both 0 and 1 assignment, we get \( O_i = \{f_1, f_3, f_2\} \). If we assume that the trace buffer width is 2, we will select \( f_i, f_3 \) as our trace signals.

**Algorithm 3.1:** Unchecked implication-based trace signal selection

1. Compute direct, indirect and extended backward implications and store them in an implication graph.
2. \( O_i = \text{Set of flip-flops } f_i \text{, where } i \in 1, 2, \ldots, N \text{ in the descending order of number of implications } |\text{Impl}[f_i,0]| \text{, where } i \in 1, 2, \ldots, N \)
3. \( \text{reference\_list} = \bigcup_{i \in 1,2,...,p} \text{Impl}[\text{PI}_i,0]; \)

4. \textbf{for each} (flip-flop \( f_i \) where \( i \in 1,2,...,N \))
   \[ \textbf{for each} (\text{implication} m \in \text{Impl}[f_i,0]) \]
   \[ \text{if } (m \setminus \text{reference\_list} \neq \emptyset) \text{ then} \]
   \[ \text{Impl}[f_i,0] = \text{Impl}[f_i,0]-m; \]
   \[ \text{else} \]
   \[ \text{reference\_list} = \text{reference\_list} \cup m; \]

5. \( O_f = \text{Set of flip-flops } f_i \text{ where } i \in 1,2,...,N \text{ in the descending order of number of unchecked implications } |\text{Impl}[f_i,0]| \text{ where } i \in 1,2,...,N \)

6. \text{If trace buffer width}=n, select first } n \text{ flip-flops from the set } O_f \text{ as trace signals.}

We measure the quality of trace signals selected by Algorithm 3.1 using the two parameters defined in Section 2.6, \textit{Restoration Ratio} and \textit{Restoration Percentage}. In the next section, we present a novel state restoration algorithm which is used to obtain the above evaluation metrics.

### 3.3 State Restoration from Traced Signals

Our main objective is to maximize the number of internal signals which can be restored using the selected trace signal data. Given the traced signal values, we use a SAT-based multi-node implication based approach discussed in Section 2.5 to determine the restored signals across several time frames. For each time frame \( k \in 0,1,2,...T \) we provide the SAT-based multi-node implication engine a signal assignment set \( S_k = S_{PI} \cup S_t \), where \( S_{PI} \) is the set containing \( PI \) values and \( S_t \) is the set containing the current state values of the selected traced flip-flops in time frame \( k \). We enlarge the set \( S_k \) for each time frame by including current state values of non-traced flops determined using forward and backward learning, which are defined as follows:

\textit{Forward Learning: } For a signal assignment set \( S_{k-1} \) in time frame \( k-1 \), if \{signal \( g=v \)\} \in \text{BCP}(S_{k-1}) and \( g \) is the fan-in signal of non-traced flip-flop \( f \), then for time frame \( k \), \( S_k = S_k \cup \{f=v\} \).
Backward Learning: For a set of trace flip-flops \( \{f_1, f_2, ..., f_n\} \) and the corresponding fan-in signals \( \{g_1, g_2, ..., g_n\} \), if in time frame \( k \) the current state values of the traced flip-flops are \( v_1, v_2, ..., v_n \) then in time frame \( k-1 \), \( S_{k-1}=S_k \cup \{g_1=v_1, g_2=v_2, ..., g_n=v_n\} \). If \{signal \( g=v \}\} e\text{BCP}(S_{k-1}) \) and \( g \) is the fan-in signal of non-traced flip-flop \( f \), then for time frame \( k \), \( S_k=S_k \cup \{f=v\} \).

For time frame \( k \), suppose \( S_f \) is the set of non-trace flip-flop assignments determined by forward learning and \( S_b \) is the set of non-trace flip-flop assignments determined by backward learning. Then, for time frame \( k \), \( S_k=S_{pf} \cup S_f \cup S_{bf} \).

We use Figure 3.1 to illustrate the concept of forward and backward learning. Figure 3.1 shows a 3-frame expansion of a sequential circuit. Let us assume that the trace flip-flops are \( \{f_1, f_2, f_3\} \), the non-trace flip-flops are \( \{f_4, f_5, f_6\} \) and the PIs are \( \{p_1, p_2, ..., p_n\} \) assigned to values \( v_1, v_2, ..., v_n \) respectively. The gates \( g_1, g_2, g_3, g_4, g_5, g_6 \) are the input signals of \( f_1, f_2, f_3, f_4, f_5, f_6 \) respectively. For time-frame 0, the signal assignment set \( S_0=S_{pf} \cup S_{bf} \) where \( S_f=\{f_1=0, f_2=0, f_3=0\} \). Suppose \( \{g_4=0\} \) for forward learning, i.e. \( S_f=\{f_4=0\} \). Therefore, in time frame 1, \( S_1=S_f \cup S_{bf} \cup \{f_4=0\} \). Note that the subscript \( f \) for the values \( 0_0, 1_1 \) and \( X_f \) indicate the values learned by forward learning. If the current state values of traced flip-flops \( \{f_1, f_2, f_3\} \) in time-frame 1 are \( \{0,1,0\} \), then in time frame 0, \( S_0=S_{pf} \cup S_f \cup \{g_1=0, g_2=1, g_3=0\} \) where \( S_f=\{f_1=0, f_2=0, f_3=0\} \). Suppose \( \{g_3=1\} \) for backward learning, i.e. \( S_b=\{f_3=1\} \). Therefore, in time frame 1, \( S_1=S_f \cup S_{bf} =S_f \cup \{f_3=1\} \). Note that the subscript \( b \) for the values \( 0_b, 1_b \) and \( X_b \) indicate the values learned by backward learning. Hence, considering both forward and backward learning for time frame 1, we get \( S_1= S_f \cup S_{bf} = S_f \cup \{f_4=0\} \cup \{f_3=1\} \). Similarly, for time frame 2, \( S_2= S_{pf} \cup S_f \cup S_{bf} = S_{pf} \cup \{f_4=1, f_5=0, f_6=0\} \cup \{\Phi\} \).

Finally, we determine the multi-node implications BCP\( (S_k) \) for these assignments in time frame \( k \) using our SAT-based implication engine and count the number of restored internal signals. Restoration Ratio and Restoration Percentage are then calculated using equations described in Section 2.6. We also vary the trace buffer width and observe the effect on the restoration percentage across a set of 100 vectors (time-frames) for each.
circuit. The experimental results are given in the next section. Algorithm 3.2 gives an overview of our state restoration approach.

Algorithm 3.2: State restoration using multi-node implications

1. \texttt{trace\_signal\_list} = flip-flops \( f_i \), where \( i \neq 1,2,...,n \)
2. \texttt{for each} (vector \( V_i \), where \( i \neq 0,1,2,...,T \))
   Perform logic simulation using \( V_i \);
   Signal assignment set \( S_i = S_{PI} \cup S_f \cup S_t \cup S_b \);
   Perform SAT-based multi-node implications \( BCP(S_i) \);
   \( N_{restored(all)} \) = number of signals including the PIs implied to either 0 or 1.
   \( N_{restored(FFs)} \) = number of non-trace flip-flops implied to either 0 or 1.
   Use equations provided in Section II to calculate restoration ratio and restoration percentage.

3.4 Experimental Results

The above algorithms were written in C++ and experiments were conducted for ISCAS’89 sequential benchmark circuits on a Linux workstation with 2GB RAM. The results are reported in Tables 3.1-3.5. During state restoration, we do not assume any knowledge of an initial state other than the traced signals. We consider five different
trace buffer widths: 8, 16, 32, 64 and 128. The trace buffer depth is assumed to be 100 in contrast to [1] in which the trace buffer depth is assumed to be 4k. Note that less restoration is possible in the beginning and more restoration toward the end of the vector sequence since more values can be learned later from the earlier vectors. Therefore, each random pattern used for our experiments has 100 vectors each for a more competitive comparison. We compare our results with [1] using the parameter $RR$ (Restoration Ratio).

For a sanity check, we also perform an experiment in which we perform a poor trace signal selection by selecting flip-flops with the least number of unchecked implications. We compare the total restoration percentage $TR_{all} (P)$ obtained by the poor trace signal selection with total restoration percentage $TR_{all} (A)$ obtained by the proposed trace signal selection algorithm for the first 10 vectors for different trace buffer widths reported in Table 3.6. Tables 3.1-3.5 give the results for trace buffer widths of 8, 16, 32, 64 and 128 using only forward learning and assuming that the primary input values are known. For each circuit, the first two columns give the total number of flip-flops in the circuit and number of vectors considered respectively, followed by restoration ratio ($RR$) obtained in [1], the trace signal execution time in [1], the restoration ratio ($RR$) obtained by our method, total restoration percentage (only flip-flops) across 100 vectors ($TR_{FFs}$), total restoration percentage (all gates) across 100 vectors ($TR_{all}$), the restoration percentage (only flip-flops) obtained in the final vector ($FR_{FFs}$), the restoration percentage (all gates) obtained in the final vector ($FR_{all}$), and the trace selection execution time. Compared to [1] our approach has a better restoration ratio ($RR$). We also observe that our approach is considerably faster for all circuits than [1]. For example, consider s15850 of Table 3.1 with only 8 traced signals; we achieved a high $RR$ of 55.6 and execution time of only 17.9 seconds as compared to a $RR$ of 19.93 and execution time of 298.9 seconds achieved by [1]. Note that the trace selection execution time in our method is independent of the trace buffer width and depth, but only on the circuit size. In s15850, we were able to restore 92.6% of all signals with only 8 trace signals. We were able to achieve high restoration percentage ($FR_{all}$) for all circuits, sometimes with only 8 or 16 trace signals. We can observe that with an increase in the trace buffer width (toward 128) the restoration percentage also tends to increase, as expected. For s35932 and s38584, the restoration percentage approaches 100 percent as we approach the final vector. Figure 3.2
and Figure 3.3 illustrate this graphically. For a trace buffer width of 32, this trend is illustrated graphically in Figure 3.8 and Figure 3.9.

Table 3.6 compares our method with the poor trace signal selection. Results for the first 10 vectors, which are the hardest to restore (e.g., initial state of non-trace FFs is unknown), are reported. Our restoration percentage $TR_{all}(A)$ is superior than the $TR_{all}(P)$ obtained using the poor selection of trace signals for all trace buffer widths. Among the circuits, s35932 is a special case: similar results were observed for both cases, since the number of unchecked implications per flip-flop is uniformly distributed in this circuit.

Finally, Tables 3.7-3.11 compare the results obtained using forward learning only (H$_1$) with the results obtained using both forward learning and backward learning (H$_2$) for trace buffer widths of 8, 16, 32, 64 and 128. Clearly, backward learning further improves the restoration percentage. For example, consider s38584 in Table 3.9 for a trace buffer width of 32, H$_2$ achieved a high restoration percentage ($TR_{all}$) of 92.9% as compared to a restoration percentage ($TR_{all}$) of only 86.4% achieved using H$_1$.

<table>
<thead>
<tr>
<th>Name</th>
<th>#FFs</th>
<th>#Vec.</th>
<th>$RR_{old}$</th>
<th>$Time_{old}$(s)</th>
<th>$RR_{new}$</th>
<th>$TR_{FFs}$</th>
<th>$TR_{all}$</th>
<th>$FR_{FFs}$</th>
<th>$FR_{all}$</th>
<th>$Time_{new}$(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5378</td>
<td>179</td>
<td>100</td>
<td>14.68</td>
<td>14.3</td>
<td>19.3</td>
<td>86.3</td>
<td>90.4</td>
<td>87.7</td>
<td>91.1</td>
<td>1.3</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>100</td>
<td>4.767</td>
<td>26.3</td>
<td>20.3</td>
<td>77.1</td>
<td>90.5</td>
<td>83.4</td>
<td>93.4</td>
<td>9.6</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>100</td>
<td>19.95</td>
<td>298.9</td>
<td>55.6</td>
<td>83.2</td>
<td>91.3</td>
<td>85.6</td>
<td>92.6</td>
<td>17.9</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>43.3</td>
<td>54.3</td>
<td>61.3</td>
<td>57.9</td>
<td>64.2</td>
<td>16.8</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>100</td>
<td>19.24</td>
<td>388.6</td>
<td>130.1</td>
<td>72.9</td>
<td>82.4</td>
<td>98.8</td>
<td>99.3</td>
<td>726</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>100</td>
<td>64.0</td>
<td>1407.6</td>
<td>209.6</td>
<td>97.0</td>
<td>97.6</td>
<td>100</td>
<td>100</td>
<td>193</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>#FFs</th>
<th>#Vec.</th>
<th>$RR_{old}$</th>
<th>$Time_{old}$(s)</th>
<th>$RR_{new}$</th>
<th>$TR_{FFs}$</th>
<th>$TR_{all}$</th>
<th>$FR_{FFs}$</th>
<th>$FR_{all}$</th>
<th>$Time_{new}$(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5378</td>
<td>179</td>
<td>100</td>
<td>8.996</td>
<td>35.9</td>
<td>9.7</td>
<td>86.5</td>
<td>90.6</td>
<td>87.7</td>
<td>91.1</td>
<td>1.4</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>100</td>
<td>7.182</td>
<td>75.2</td>
<td>10.3</td>
<td>77.8</td>
<td>91.2</td>
<td>83.4</td>
<td>93.4</td>
<td>9.5</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>100</td>
<td>24.22</td>
<td>764.4</td>
<td>27.8</td>
<td>83.3</td>
<td>91.33</td>
<td>85.6</td>
<td>92.6</td>
<td>18.0</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>24.5</td>
<td>61.4</td>
<td>68.8</td>
<td>64.6</td>
<td>70.9</td>
<td>16.9</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>100</td>
<td>13.96</td>
<td>802.9</td>
<td>66.02</td>
<td>74.1</td>
<td>83.3</td>
<td>98.8</td>
<td>99.3</td>
<td>726</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>100</td>
<td>38.13</td>
<td>5251.1</td>
<td>104.8</td>
<td>97.04</td>
<td>97.56</td>
<td>100</td>
<td>100</td>
<td>195</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>#FFs</th>
<th>#Vec.</th>
<th>$RR_{old}$</th>
<th>$Time_{old}$(s)</th>
<th>$RR_{new}$</th>
<th>$TR_{FFs}$</th>
<th>$TR_{all}$</th>
<th>$FR_{FFs}$</th>
<th>$FR_{all}$</th>
<th>$Time_{new}$(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5378</td>
<td>179</td>
<td>100</td>
<td>4.726</td>
<td>74.9</td>
<td>4.84</td>
<td>86.7</td>
<td>90.8</td>
<td>87.7</td>
<td>91.1</td>
<td>1.3</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>100</td>
<td>4.672</td>
<td>148.2</td>
<td>5.2</td>
<td>78.5</td>
<td>91.8</td>
<td>83.9</td>
<td>93.6</td>
<td>9.5</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>100</td>
<td>13.3</td>
<td>1654.6</td>
<td>13.9</td>
<td>83.5</td>
<td>91.5</td>
<td>85.6</td>
<td>92.6</td>
<td>17.9</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>13.1</td>
<td>65.6</td>
<td>74.4</td>
<td>68.5</td>
<td>76.3</td>
<td>16.9</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>100</td>
<td>8.679</td>
<td>2826.0</td>
<td>34.8</td>
<td>78.1</td>
<td>86.4</td>
<td>98.8</td>
<td>99.3</td>
<td>726</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>100</td>
<td>21.06</td>
<td>10496.2</td>
<td>52.4</td>
<td>97.07</td>
<td>97.57</td>
<td>100</td>
<td>100</td>
<td>194</td>
</tr>
</tbody>
</table>
Table 3.4: Experimental Results (Trace Buffer Width=64)

<table>
<thead>
<tr>
<th>Name</th>
<th>#FFs</th>
<th>#Vec.</th>
<th>RR</th>
<th>Time</th>
<th>RR</th>
<th>TR_FF</th>
<th>TR_all</th>
<th>FR_FF</th>
<th>FR_all</th>
<th>Time_new</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5378</td>
<td>179</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>2.6</td>
<td>92.4</td>
<td>94.9</td>
<td>93.3</td>
<td>95.3</td>
<td>1.5</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>2.6</td>
<td>79.5</td>
<td>92.6</td>
<td>84.4</td>
<td>94.0</td>
<td>9.7</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>7.0</td>
<td>84.1</td>
<td>92.1</td>
<td>85.6</td>
<td>92.6</td>
<td>18.1</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>6.9</td>
<td>69.7</td>
<td>78.2</td>
<td>71.5</td>
<td>79.5</td>
<td>16.9</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>17.8</td>
<td>80.1</td>
<td>88.4</td>
<td>98.8</td>
<td>99.3</td>
<td>723</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>26.2</td>
<td>97.13</td>
<td>97.59</td>
<td>100</td>
<td>100</td>
<td>194</td>
</tr>
</tbody>
</table>

Table 3.5: Experimental Results (Trace Buffer Width=128)

<table>
<thead>
<tr>
<th>Name</th>
<th>#FFs</th>
<th>#Vec.</th>
<th>RR</th>
<th>Time</th>
<th>RR</th>
<th>TR_FF</th>
<th>TR_all</th>
<th>FR_FF</th>
<th>FR_all</th>
<th>Time_new</th>
</tr>
</thead>
<tbody>
<tr>
<td>s5378</td>
<td>179</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>1.4</td>
<td>99.7</td>
<td>99.9</td>
<td>100</td>
<td>100</td>
<td>1.4</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>1.4</td>
<td>83.9</td>
<td>94.5</td>
<td>85.6</td>
<td>92.6</td>
<td>9.9</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>3.5</td>
<td>84.2</td>
<td>92.2</td>
<td>85.6</td>
<td>92.6</td>
<td>19.8</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>3.9</td>
<td>77.3</td>
<td>82.9</td>
<td>78.8</td>
<td>83.7</td>
<td>18.8</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>9.7</td>
<td>87.3</td>
<td>93.1</td>
<td>99.9</td>
<td>99.8</td>
<td>785</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>13.13</td>
<td>97.24</td>
<td>97.64</td>
<td>100</td>
<td>100</td>
<td>215</td>
</tr>
</tbody>
</table>

Table 3.6: Experimental Results-Poor (P) vs. our Algo (A) for the first 10 vectors

<table>
<thead>
<tr>
<th>Name</th>
<th>TB Width=8</th>
<th>TB Width=16</th>
<th>TB Width=32</th>
<th>TB Width=64</th>
<th>TB Width=128</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TR_all(P)</td>
<td>TR_all(A)</td>
<td>TR_all(P)</td>
<td>TR_all(A)</td>
<td>TR_all(P)</td>
</tr>
<tr>
<td>s5378</td>
<td>73.7</td>
<td>82.8</td>
<td>74.2</td>
<td>84.9</td>
<td>81.5</td>
</tr>
<tr>
<td>s9234</td>
<td>57.8</td>
<td>72.3</td>
<td>58.1</td>
<td>78.4</td>
<td>59.8</td>
</tr>
<tr>
<td>s15850</td>
<td>75.4</td>
<td>79.0</td>
<td>75.9</td>
<td>79.6</td>
<td>76.1</td>
</tr>
<tr>
<td>s13207</td>
<td>29.7</td>
<td>42.0</td>
<td>30.1</td>
<td>52.5</td>
<td>31.1</td>
</tr>
<tr>
<td>s38584</td>
<td>38.9</td>
<td>41.1</td>
<td>40.3</td>
<td>43.9</td>
<td>41.3</td>
</tr>
<tr>
<td>s35932</td>
<td>75.5</td>
<td>75.6</td>
<td>75.6</td>
<td>75.6</td>
<td>75.9</td>
</tr>
</tbody>
</table>

Figure 3.2: TR_all vs. TB width
Figure 3.3: $\text{TR}_{\text{FFs}}$ vs. TB width

Figure 3.4: Restoration % (all) vs. vector # for TB width=8
Figure 3.5: Restoration % (FFs) vs. vector # for TB width=8

Figure 3.6: Restoration % (all) vs. vector # for TB width=16
Figure 3.7: Restoration % (FFs) vs. vector # for TB width=16

Figure 3.8: Restoration % (all) vs. vector # for TB width=32
Figure 3.9: Restoration % (FFs) vs. vector # for TB width=32

Figure 3.10: Restoration % (all) vs. vector # for TB width=64
Figure 3.11: Restoration % (FFs) vs. vector # for TB width=64

Figure 3.12: Restoration % (all) vs. vector # for TB width=128
Figure 3.13: Restoration % (FFs) vs. vector # for TB width=128

Table 3.7: Forward Learning ($H_1$) vs. Backward Learning ($H_2$) (N=8)

<table>
<thead>
<tr>
<th></th>
<th>$H_1$</th>
<th></th>
<th>$H_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR$_{old}$</td>
<td>TR$_{all}$</td>
<td>TR$_{FF}$</td>
</tr>
<tr>
<td>s5378</td>
<td>14.68</td>
<td>90.4</td>
<td>86.3</td>
</tr>
<tr>
<td>s9234</td>
<td>4.767</td>
<td>90.5</td>
<td>77.1</td>
</tr>
<tr>
<td>s13207</td>
<td>-</td>
<td>61.3</td>
<td>54.3</td>
</tr>
<tr>
<td>s15850</td>
<td>19.93</td>
<td>91.3</td>
<td>83.3</td>
</tr>
<tr>
<td>s38584</td>
<td>19.24</td>
<td>82.5</td>
<td>73.1</td>
</tr>
<tr>
<td>s35932</td>
<td>64.0</td>
<td>97.6</td>
<td>97.0</td>
</tr>
<tr>
<td>s38417</td>
<td>18.6</td>
<td>36.3</td>
<td>21.8</td>
</tr>
</tbody>
</table>

Table 3.8: Forward Learning ($H_1$) vs. Backward Learning ($H_2$) (N=16)

<table>
<thead>
<tr>
<th></th>
<th>$H_1$</th>
<th></th>
<th>$H_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RR$_{old}$</td>
<td>TR$_{all}$</td>
<td>TR$_{FF}$</td>
</tr>
<tr>
<td>s5378</td>
<td>8.996</td>
<td>90.6</td>
<td>86.5</td>
</tr>
<tr>
<td>s9234</td>
<td>7.182</td>
<td>91.2</td>
<td>77.8</td>
</tr>
<tr>
<td>s13207</td>
<td>-</td>
<td>70.8</td>
<td>62.9</td>
</tr>
<tr>
<td>s15850</td>
<td>24.22</td>
<td>91.3</td>
<td>83.3</td>
</tr>
<tr>
<td>s38584</td>
<td>13.96</td>
<td>83.3</td>
<td>74.1</td>
</tr>
<tr>
<td>s35932</td>
<td>38.13</td>
<td>97.6</td>
<td>97.0</td>
</tr>
<tr>
<td>s38417</td>
<td>18.6</td>
<td>36.6</td>
<td>21.8</td>
</tr>
</tbody>
</table>
Table 3.9: Forward Learning (H₁) vs. Backward Learning (H₂) (N=32)

<table>
<thead>
<tr>
<th></th>
<th>H₁</th>
<th>H₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RRₜₜ</td>
<td>TRₜₜ</td>
</tr>
<tr>
<td>s5378</td>
<td>4.726</td>
<td>90.8</td>
</tr>
<tr>
<td>s9234</td>
<td>4.672</td>
<td>91.8</td>
</tr>
<tr>
<td>s13207</td>
<td>-</td>
<td>74.3</td>
</tr>
<tr>
<td>s15850</td>
<td>13.3</td>
<td>91.5</td>
</tr>
<tr>
<td>s38584</td>
<td>8.679</td>
<td>86.4</td>
</tr>
<tr>
<td>s35932</td>
<td>21.06</td>
<td>97.6</td>
</tr>
<tr>
<td>s38417</td>
<td>14.2</td>
<td>37.2</td>
</tr>
</tbody>
</table>

Table 3.10: Forward Learning (H₁) vs. Backward Learning (H₂) (N=64)

<table>
<thead>
<tr>
<th></th>
<th>H₁</th>
<th>H₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RRₜₜ</td>
<td>TRₜₜ</td>
</tr>
<tr>
<td>s5378</td>
<td>-</td>
<td>94.9</td>
</tr>
<tr>
<td>s9234</td>
<td>-</td>
<td>92.6</td>
</tr>
<tr>
<td>s13207</td>
<td>-</td>
<td>78.1</td>
</tr>
<tr>
<td>s15850</td>
<td>-</td>
<td>92.1</td>
</tr>
<tr>
<td>s38584</td>
<td>-</td>
<td>88.4</td>
</tr>
<tr>
<td>s35932</td>
<td>-</td>
<td>97.6</td>
</tr>
<tr>
<td>s38417</td>
<td>-</td>
<td>39.4</td>
</tr>
</tbody>
</table>

Table 3.11: Forward Learning (H₁) vs. Backward Learning (H₂) (N=128)

<table>
<thead>
<tr>
<th></th>
<th>H₁</th>
<th>H₂</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RRₜₜ</td>
<td>TRₜₜ</td>
</tr>
<tr>
<td>s5378</td>
<td>-</td>
<td>99.9</td>
</tr>
<tr>
<td>s9234</td>
<td>-</td>
<td>94.5</td>
</tr>
<tr>
<td>s13207</td>
<td>-</td>
<td>82.9</td>
</tr>
<tr>
<td>s15850</td>
<td>-</td>
<td>92.2</td>
</tr>
<tr>
<td>s38584</td>
<td>-</td>
<td>93.1</td>
</tr>
<tr>
<td>s35932</td>
<td>-</td>
<td>97.6</td>
</tr>
<tr>
<td>s38417</td>
<td>-</td>
<td>44.6</td>
</tr>
</tbody>
</table>
3.5 Summary

We have proposed a new trace signal selection method using non-trivial logic implications for the purpose of post-silicon debug. The selection of trace signals is performed by choosing those signals with the most number of implications that are not implied by other signals. Results show that our approach gives a better restoration ratio than previous approaches. Moreover, since our method is learning-based, it is considerably faster than the earlier search based methods which use restorability metrics that consider both the topology and behavior of logic gates.
Chapter 4
Multiplexed Trace Selection and State Restoration

This chapter is organized as follows. Section 4.1 formulates the problem. Section 4.2 discusses the proposed approach for multiplexed trace signal selection using a new concept known as implication-based correlation. Section 4.3 introduces the algorithm used for state restoration for multiplexed trace signals. Section 4.4 introduces the SAT-based greedy heuristic used for pruning the selected trace signal list further. Section 4.5 reports experimental results, and Section 4.6 summarizes the observation.

4.1 Problem Formulation

Let $G$ represent the set of all gates in the circuit and $T_1$ and $T_2$ represent the set of trace signals to be selected for the even and odd time frames respectively. We define the problem statement as follows: Find the smallest subset of signals $T_1 \subseteq G$ and $T_2 \subseteq G$ such that $T_1 \cap T_2 = \emptyset$ and $\forall$ legal valuations $r_1$ of $T_1$ and $r_2$ of $T_2$ the values of the signals in $G - T$ can be restored across two consecutive time-frames, where $T = T_1 \cup T_2$. In other words, our objective is to increase the storage efficiency of the trace buffer by storing two different sets of trace signals in consecutive time-frames, thus widening the debug observation window. By multiplexing two different sets of trace signals, cumulatively we are tracing twice the number of trace signals as compared to the non-multiplexed scheme where the same set of trace signals are traced in every time-frame.
4.2 Implication-based Correlation

We use both forward and backward implications of the flip-flops and the correlation between them to drive our new multiplexed trace signal selection heuristic. Next, we derive a mathematical model to represent the implication-based correlation between two flip-flops in consecutive time-frames. Let us first consider two flip-flops $f_i$ and $f_j$ ($i \neq j$) in time-frames $t$ and $t+1$ respectively. Let $\text{Impl}_{f_i}[f_i,t]$ represent the forward implications of $f_i$ in time-frame $t$ resulting from assigning flip-flop $f_i$ in time frame $t$ to value 0 and 1 respectively and $\text{Impl}_{f_j}[f_j,t+1]$ represent the backward implications of $f_j$ in time-frame $t$ resulting from assigning flip-flop $f_j$ in time frame $t+1$ to value 0 and 1 respectively. An implication-based correlation parameter $C'_{ij}$ is derived using the following objectives:

1. Maximize the size $S_U = |\text{Impl}_{f_i}[f_i,t] \cup \text{Impl}_{f_j}[f_j,t+1]|$
2. Minimize the size $S_I = |\text{Impl}_{f_i}[f_i,t] \cap \text{Impl}_{f_j}[f_j,t+1]|$
3. The size $S_f = |\text{Impl}_{f_i}[f_i,t]|$ and $S_b = |\text{Impl}_{f_j}[f_j,t+1]|$ should be balanced. We denote it using a parameter called balance factor ($y$).

We will illustrate the derivation of balance factor using Figure 4.1. Let Y-axis represent the balance factor, $y$, where $0 \leq y \leq 1$. Let X-axis represent the size ratio, $x$, where $0 \leq x \leq 1$. Size Ratio, $x$ is given by the following equation:

$$x = \frac{S_f}{S_f + S_b}$$

(4.1)

Figure 4.1 represents a parabola with vertex at (0.5, 1) and passing through the points (0, 0) and (1, 0). The best case of balance occurs when $S_f = S_b$, which means $x = 0.5$ and $y = 1$. 

Figure 4.1: Balance Factor
Therefore, the vertex of the parabola is (0.5, 1). The worst case of balance occurs in two situations: (a) $S_f = 0$ and $S_b \neq 0$, which means $x=0$ and $y=0$, and (b) $S_b = 0$ and $S_f \neq 0$, which means $x=1$ and $y=0$. Therefore, the parabola passes through the points (0, 0) and (1, 0).

A parabola can be represented using the following equation:

$$y = k + a \times (x - h)^2$$

(4.2)

Where $(h, k)$ is the vertex of parabola.

The parabola passes through the point (0, 0) and has a vertex at (0.5, 1). Putting $x=0, y=0, h=0.5$ and $k=1$ in Equation 4.2 we get $a=-4$. Therefore, the balance factor, $y$ is given by:

$$y = 1 - 4 \times (x - 0.5)^2$$

(4.3)

Thus, the correlation parameter $C_{ij}$ is represented by the following equation:

$$C_{ij} = \frac{S_u}{S_t} \times y$$

(4.4)

Note that if $S_t=0$, we give a default value of 0.5 to $S_t$ so that the denominator does not become 0. Let us now consider flip-flops $f_j$ and $f_i$ $(j \neq i)$ in time-frames $t$ and $t+1$ respectively. Thus,

$$S_U = \text{Impl}_{f_j,t} \cup \text{Impl}_{b,f_i,t+1}$$

$$S_I = \text{Impl}_{f_j,t} \cap \text{Impl}_{b,f_i,t+1}$$

$$S_f = \text{Impl}_{f_j,t}$$

and $S_b = \text{Impl}_{b,f_i,t+1}$

The correlation parameter $C_{ij}^2$ for this case can also be determined using Equation 4.4, and the implication-based correlation parameter for two flip-flops $f_i$ and $f_j$ is given by:

$$C_{ij} = C_{ij}^1 + C_{ij}^2$$

(4.5)

Algorithm 4.1 gives an overview of the multiplexed trace signal selection method. We first order the first set of flip-flops $O_t$ in the descending order of number of unchecked forward implications (steps 4-6) and form the set $O'$. We then determine $C_{ij}$ for all flip-flop pairs between $O'$ and $O_{t+1}$ and order the pairs in the descending order of $C_{ij}$. Steps 8-13 in algorithm 4.1 are then used to determine the two sets of trace signals $T_1$ and $T_2$. 
Algorithm 4.1: Multiplexed Trace signal selection using implication-based correlation.

1. Unroll the sequential circuit into three time frames t-1, t and t+1. Determine SAT-based implications for PIs, POs and flip-flops.

2. $O_t = \text{Set of flip-flops } f_i \text{ where } i=1,2,...,N \text{ in time-frame } t \text{ in the descending order of number of forward implications }\{|\text{Impl}_{f_i, t}\}| \text{ where } i=1,2,...,N$

3. $O_{t+1} = \text{Set of flip-flops } f_i \text{ where } i=1,2,...,N \text{ in time-frame } t+1 \text{ in the descending order of number of backward implications }\{|\text{Impl}_{f_i, t+1}\}| \text{ where } i=1,2,...,N$

4. reference_list = $\bigcup_{i=1}^{p} \text{Impl}[P_{i}, t] \bigcup_{i=1}^{m} \text{Impl}[P_{O_i}, t]$;

5. for each (flip-flop $f_i$ where $i=1,2,...,N$ in time-frame $t$)
   
   for each (implication $m \in \text{Impl}_{f_i, t}$)
   
   if ($m \cap \text{reference_list} \neq \emptyset$) then
   
   $\text{Impl}_{f_i, t} = \text{Impl}_{f_i, t} - m$;

   else
   
   reference_list = reference_list $\cup m$;

6. $O'_t = \text{Set of flip-flops } f_i \text{ where } i=1,2,...,N \text{ in time-frame } t \text{ in the descending order of number of unchecked forward implications }\{|\text{Impl}_{f_i, t}\}| \text{ where } i=1,2,...,N$

7. for each (flip-flop $f_i$ where $i=1,2,...,N-1$ in $O'_t$)

   for each (flip-flop $f_j$ where $j=i+1,...,N$ in $O_{t+1}$)

   Calculate $C_{ij}$

8. $O_k = \text{Flip-flop pairs } (f_i, f_j) \text{ in the descending order of } C_{ij}$.

9. $T_1 = T_2 = \emptyset, k=1$

10. Let s be the size of $T_1$ and $T_2$, where $s \leq N/2$

11. while($|T_1| \neq s$)

    $O_k = (f_i, f_j)$

    if $f_i \notin T_1 \cup T_2$ and $f_j \notin T_1 \cup T_2$

    $T_1 = T_1 \cup f_i, T_2 = T_2 \cup f_j$

    $k=k+1$

12. Select first n flip-flops ($n \leq s$) from $T_1$ to be traced in even time-frames.

13. Select first n flip-flops ($n \leq s$) from $T_2$ to be traced in odd time-frames
We use Figure 4.2 to illustrate the concept of implication-based correlation. Let a sequential circuit be unrolled into three time-frames as shown in Figure 4.2. Let \( f_{i, i=1,2,\ldots,6} \) represent the flip-flops and \( g_{i, i=1,2,\ldots} \) represent the logic gates (excluding flip-flops). In the figure, the forward implications of a flip-flop are the gates covered in the right cone of that flip-flop. The backward implications of a flip-flop are the gates covered in the left cone of that flip-flop. Note that \( g_i \) means \( g_i = 1 \), \( g' \) means \( g_i = 0 \). We use two time frames 0 and 1 to represent the forward and backward implications for clarity. Let us consider the flip-flop pair \( f_3 \) and \( f_4 \). First we will calculate the correlation parameter \( C^{1}_{34} \). The forward implications of \( f_3 \) and backward implications of \( f_4 \) are shown in time frame 1. Clearly, \( \text{Impl}_{f}[f_3,1]=\{g_1', g_2, g_3, g_4, g_5, g_6\} \) and \( \text{Impl}_{b}[f_4,2]=\{g_1', g_3, g_5, g_6, g_7\} \). Thus, \( S_f=|\text{Impl}_{f}[f_3,1]|=6 \)
\( S_b=|\text{Impl}_{b}[f_4,2]|=5 \)
\( S_U=|\text{Impl}_{f}[f_3,1] \cup \text{Impl}_{b}[f_4,2]|=7 \) \( S_I=|\text{Impl}_{f}[f_3,1] \cap \text{Impl}_{b}[f_4,2]|=4 \)
Using Equation 4.1, \( x=S_f/(S_f+S_b) =6/(6+5) =6/11=0.55 \)
Using Equation 4.3, \( y=1-4(x-0.5)^2=0.99 \)
Using Equation 4.4, \( C^{1}_{34} = (S_U/S_I) \times y = (7/4) \times 0.99 = 1.73 \)
Next we calculate the correlation parameter \( C^{2}_{34} \). The forward implications of \( f_4 \) and backward implications of \( f_3 \) are shown in time frame 0. Clearly, \( \text{Impl}_{f}[f_4,0]=\{g_4', g_5, g_9, g_{10}\} \) and \( \text{Impl}_{b}[f_3,1]=\{g_5, g_6, g_9, g_{10}\} \). Thus, \( S_f=|\text{Impl}_{f}[f_4,0]|=4 \)
\( S_b=|\text{Impl}_{b}[f_3,1]|=4 \)
\( S_U=|\text{Impl}_{f}[f_4,0] \cup \text{Impl}_{b}[f_3,1]|=5 \) \( S_I=|\text{Impl}_{f}[f_4,0] \cap \text{Impl}_{b}[f_3,1]|=3 \)
\( x=S_f/(S_f+S_b)=4/(4+4)=4/8=0.5 \)
\( y=1-4(x-0.5)^2=1 \)
\( C^{2}_{34} = (S_U/S_I) \times y = (5/3) \times 1 = 1.67 \)
Hence, \( C_{34} = C^{1}_{34} + C^{2}_{34}=1.73+1.67=3.4 \)
Using the following example, we briefly explain the determination of unchecked implications (steps 2-6 in algorithm 4.1) under the assumption that primary input and output values are known. Let us consider a circuit with two flip-flops $f_1, f_2$, one primary input $PI_1$, one primary output $PO_1$ and five gates $g_1, g_2, g_3, g_4, g_5$. Suppose we obtain the following information from the implication graph:

- $\text{Impl}[PI_1,0,0] = \{ PI_1 = 0, g_1 = 1, g_5 = 0 \}$
- $\text{Impl}[PI_1,1,0] = \{ PI_1 = 1, g_2 = 1, g_4 = 0 \}$
- $\text{Impl}[PO_1,0,0] = \{ PO_1 = 0, g_1 = 0, g_5 = 1 \}$
- $\text{Impl}[PO_1,1,0] = \{ PO_1 = 1, g_2 = 1, g_4 = 1 \}$
- $\text{Impl}[f_1,0,0] = \{ f_1 = 0, g_1 = 1, g_2 = 1, g_3 = 0, g_5 = 1 \}$
- $\text{Impl}[f_1,1,0] = \{ f_1 = 1, g_1 = 1, g_2 = 0, g_3 = 0, g_4 = 1 \}$
- $\text{Impl}[f_2,0,0] = \{ f_2 = 0, g_1 = 1, g_2 = 1, g_3 = 1 \}$
- $\text{Impl}[f_2,1,0] = \{ f_2 = 1, g_1 = 1, g_2 = 0, g_3 = 1, g_4 = 1 \}$

Thus, we get:

- $\text{Impl}[PI_1,0] = \text{Impl}[PI_1,0,0] \cup \text{Impl}[PI_1,1,0] = \{ PI_1 = 0, PI_1 = 1, g_1 = 1, g_2 = 1, g_4 = 0, g_5 = 0 \}$
- $\text{Impl}[PO_1,0] = \text{Impl}[PO_1,0,0] \cup \text{Impl}[PO_1,1,0] = \{ PO_1 = 0, PO_1 = 1, g_1 = 0, g_3 = 1, g_4 = 1, g_5 = 1 \}$

Let $CI$ denote the reference list containing the checked implications.
\( CI = \text{Impl}[PI_1,0] \cup \text{Impl}[PO_1,0]: \{ PI_1 = 0, \ PI_2 = 1, \ g_1 = 1, \ g_2 = 1, \ g_3 = 0, \ g_4 = 0, \ PO_1 = 0, \ PO_2 = 1, \ g_2 = 0, \ g_3 = 1, \ g_4 = 1, \ g_5 = 1 \} \)
\( \text{Impl}[f_1,0] = \text{Impl}[f_1,0,0] \cup \text{Impl}[f_1,1,0]: \{ f_1 = 1, f_1 = 0, \ g_1 = 1, \ g_2 = 0, \ g_3 = 0, \ g_4 = 1, \ g_5 = 1 \} \)
\( \text{Impl}[f_2,0] = \text{Impl}[f_2,0,0] \cup \text{Impl}[f_2,1,0]: \{ f_2 = 1, f_2 = 0, \ g_1 = 1, \ g_2 = 0, \ g_3 = 1, \ g_4 = 1, \ g_5 = 1 \} \)
\( |\text{Impl}[f_1,0]| = 8, \ |\text{Impl}[f_2,0]| = 7 \)

Since \( f_1 \) has the most number of implications, we will start with this flip-flop. Let \( UI(f_i) \) denote unchecked implications of flip-flop \( f_i \). Thus,
\( UI(f_1) = \text{Impl}[f_1,0] - CI = \{ f_1 = 1, f_1 = 0, \ g_2 = 0, \ g_3 = 0 \} \)
\( CI = CI \cup UI(f_1) = \{ PI_1 = 0, \ PI_2 = 1, \ g_1 = 1, \ g_2 = 1, \ g_3 = 0, \ g_4 = 0, \ g_5 = 0, \ PO_1 = 0, \ PO_2 = 1, \ g_1 = 0, \ g_3 = 1, \ g_4 = 1, \ g_5 = 1, f_1 = 1, f_1 = 0, \ g_2 = 0, \ g_3 = 0 \} \)
\( UI(f_2) = \text{Impl}[f_2,0] - CI = \{ f_2 = 1, f_2 = 0 \} \)
\( CI = CI \cup UI(f_2) = \{ PI_1 = 0, \ PI_2 = 1, \ g_1 = 1, \ g_2 = 1, \ g_3 = 0, \ g_4 = 0, \ g_5 = 0, \ PO_1 = 0, \ PO_2 = 1, \ g_1 = 0, \ g_3 = 1, \ g_4 = 1, \ g_5 = 1, f_1 = 1, f_1 = 0, \ g_2 = 0, \ g_3 = 0, \ f_2 = 1, f_2 = 0 \} \)

Next, we set the implications to be the unchecked ones:
\( \text{Impl}[f_1,0] = UI(f_1); \{ f_1 = 1, f_1 = 0, \ g_2 = 0, \ g_3 = 0 \} \)
\( \text{Impl}[f_2,0] = UI(f_2); \{ f_2 = 1, f_2 = 0 \} \)
\( |\text{Impl}[f_1,0]| = 4, \ |\text{Impl}[f_2,0]| = 2 \)

On ordering the flip-flops in the descending order of number of unchecked implications for both 0 and 1 assignment, we get \( O' = \{ f_1, f_2 \} \).

Next, we illustrate steps 7-13 of Algorithm 4.1 using the following example. Let us consider a circuit with four flip-flops \( f_1, f_2, f_3, f_4 \). Suppose the following are the correlation parameters \( C_{ij} \) calculated using Equations 4.1-4.5 for each possible pair of flip-flop: \( C_{12} = 10, \ C_{13} = 8, \ C_{14} = 20, \ C_{23} = 11, \ C_{24} = 5, \ C_{34} = 19 \). On ordering the flip-flop pairs in the descending order of \( C_{ij} \), we get \( O_k = \{ (f_1, f_4), (f_3, f_4), (f_2, f_3), (f_1, f_2), (f_1, f_3), (f_2, f_4) \} \). Let the trace buffer width be 2. Thus, the two trace signal lists \( T_1 \) and \( T_2 \) should contain two flip-flops each after trace selection is performed. First we initialize \( T_1 = T_2 = \emptyset \). Next consider the pair \( (f_1, f_4) \). Place \( f_1 \) in \( T_1 \) and \( f_4 \) in \( T_2 \). Thus \( T_1 = \{ f_1 \}, T_2 = \{ f_4 \} \). Consider the next pair \( (f_3, f_4) \). Since \( f_4 \) is already present in \( T_2 \) we skip \( (f_3, f_4) \) and go to the next pair \( (f_2, f_3) \). Both \( T_1 \) and \( T_2 \) do not contain either \( f_2 \) or \( f_3 \). Place \( f_2 \) in \( T_1 \) and \( f_3 \) in
4.3 Modified State Restoration Algorithm

We use Figure 4.3 to illustrate the concept of forward and backward learning for the case of multiplexed trace signals. Figure 4.3 shows a 3-frame expansion of a sequential circuit.

Let us assume that the trace flip-flops in even time-frames 0 and 2 are \{f_1, f_2, f_3\}, the non-trace flip-flops in even time-frames 0 and 2 are \{f_4, f_5, f_6\}, the trace flip-flops in odd time-frame 1 are \{f_4, f_5, f_6\}, the non-trace flip-flops in odd time-frame 1 are \{f_1, f_2, f_3\}, the \(PIs\) are \{p_1, p_2, ..., p_n\} and the \(POs\) are \{o_1, o_2, ..., o_n\}. The gates \(g_1, g_2, g_3, g_4, g_5, g_6\) are the input signals of \(f_1, f_2, f_3, f_4, f_5, f_6\) respectively.

For time-frame 0, the trace signal list \(T_0 = \{f_1, f_2, f_3\}\) and the signal assignment set \(S_0 = S_P \cup S_{PO} \cup S_{Io}\), where \(S_0 = \{f_1 = 0, f_2 = 0, f_3 = 0\}\).

For time-frame 1, the trace signal list \(T_1 = \{f_4, f_5, f_6\}\) and the signal assignment set \(S_1 = S_P \cup S_{PO} \cup S_{Io}\), where \(S_1 = \{f_4 = 0, f_5 = 1, f_6 = 0\}\).

For time-frame 2, the trace signal list \(T_2 = \{f_4, f_5, f_6\}\) and the signal assignment set \(S_2 = S_P \cup S_{PO} \cup S_{Io}\), where \(S_2 = \{f_1 = 1, f_2 = 1, f_3 = 0\}\).

Suppose \(g_2 = 0\) e BCP \((S_0)\). Since \(g_2\) is the input signal of non-trace flip-flop \(f_2\) in the odd time frame 1, the next state of \(f_2\) is learned to be 0 by forward learning, i.e. \(S_{fo} = \{f_2 = 0\}\). Therefore, in time frame 1, \(S_1 = S_1 \cup S_{fo} = S_1 \cup \{f_2 = 0\}\). Note that the subscript \(f\) for the values \(0, 1\) and \(X\) indicate the values learned by forward learning. If the current state values of traced flip-flops \(f_4, f_5, f_6\) in time-frame 1 are \(\{0, 1, 0\}\), then in time frame 0, \(S_0 = S_P \cup S_{Io}\) where \(S_0 = \{f_1 = 0, f_2 = 0, f_3 = 0\}\). Suppose \(g_3 = 1\) e BCP \((S_0)\). Since \(g_3\) is the input signal of non-trace flip-flop \(f_3\), the next state of \(f_3\) is learned to be 1 by backward learning, i.e. \(S_{bo} = \{f_3 = 1\}\). Therefore, in time frame 1, \(S_1 = S_1 \cup S_{bo} = S_1 \cup \{f_3 = 1\}\). Note that the subscript \(b\) for the values \(0, 1\) and \(X\) indicate the values learned by backward learning. Hence, considering both forward and backward learning for time frame 1, we get \(S_1 = S_1 \cup S_{fo} \cup S_{bo} = S_1 \cup \{f_2 = 0\} \cup \{f_3 = 1\}\).

Similarly, for time frame 2, \(S_2 = S_2 \cup S_{fo} \cup S_{bo} = S_2 \cup \{f_4 = 1, f_5 = 0, f_6 = 0\} \cup \{\Phi\}\). Algorithm 4.2 gives an overview of the state restoration approach for multiplexed trace signals.
Algorithm 4.2: State restoration for multiplexed trace signals

1. `trace_signal_list, T_e = flip-flops f_i, where i≠1,2,...,n in T_1`
2. `trace_signal_list, T_o = flip-flops f_i, where i≠1,2,...,n in T_2`
3. for each (vector \( V_i, \text{where } i≠0,1,2,...,T \))
   
   Perform logic simulation using \( V_i \);
   
   if (time-frame is even)
   
   Signal assignment set \( S_i = S_{PI} \cup S_{PO} \cup S_{in} \cup S_{fe} \cup S_{bc} \);

   else if (time-frame is odd)
   
   Signal assignment set \( S_i = S_{PI} \cup S_{PO} \cup S_{in} \cup S_{fo} \cup S_{bo} \);

Perform SAT-based multi-node implications \( BCP(S_i) \)

\( N_{\text{restored(all)}} = \text{number of signals including the PIs and POs implied to either 0 or 1.} \)

\( N_{\text{restored(FFs)}} = \text{number of non-trace flip-flops implied to either 0 or 1.} \)

Use equations (1), (2) and (3) to calculate restoration ratio and restoration percentage

4.4. Multi-node implication-based trace list pruning

The trace selection methods discussed so far are all based on single node implications. However, there could be a case where multi-node implications overshadow the single node-implications. Therefore, it is prudent to consider this aspect during trace selection.
We will explain this observation in detail in Section 4.5 later. Algorithms 4.3 and 4.4 give an overview of the proposed multi-node implication-based greedy heuristic for the non-multiplexed and multiplexed trace signal selection schemes respectively. We use our BCP-based state restoration algorithm as the engine for this greedy heuristic.

**Algorithm 4.3:** Non-multiplexed trace list pruning.

1. Form set $O_f$ using algorithm 3.1.
2. Trace width $n=2^l$, $l \geq 0$ Search space $s=2^k$, $k>l$, $s \leq N$
3. Number of iteration, $r = s/n$, $j=1$
4. while ($j \leq r$)
   
   trace_signal_list, $T_j =$ flip-flops $f_i$, $i=(j-1)n+1$ to $(j-1)n+n$ in $O_f$
   
   Perform state restoration using algorithm 3.2 for first five vectors (i.e. $V_i$, where $i \in \{0,1,2,..,5\}$)
   
   Restoration Percentage, $R_j =$ restoration percentage (all gates) obtained in the fifth vector
   
   $j=j+1$
5. $O_l$ = Set of trace signal lists $T_j$, $j=1,2,..,r$ in the descending order of restoration percentage $R_j$, $j=1,2,..,r$
6. Select the first trace signal list $T_i$ in $O_l$
7. Select $n$ flip-flops from $T_i$ as trace signals.

**Algorithm 4.4:** Multiplexed trace list pruning.

1. Form sets $T_1$ and $T_2$ using algorithm 4.1.
2. Trace width $n=2^l$, $l \geq 0$ Search space $s=2^k$, $k>l$, $s \leq N/2$
3. Number of iteration, $r = s/n$, $j=1$
4. while ($j \leq r$)
   
   trace_signal_list, $T_{je} =$ flip-flops $f_i$, $i=(j-1)n+1$ to $(j-1)n+n$ in $T_1$
   
   trace_signal_list, $T_{jo} =$ flip-flops $f_i$, $i=(j-1)n+1$ to $(j-1)n+n$ in $T_2$
   
   Perform state restoration using algorithm 4.2 for first five vectors (i.e. $V_i$, where $i \in \{0,1,2,..,5\}$)
   
   Restoration Percentage, $R_j =$ restoration percentage (all gates) obtained in the fifth vector
   
   $j=j+1$
5. $O_l=$Set of trace signal list pairs $(T_{le},T_{lo})_{j\neq 1,2,\ldots,r}$ in the descending order of restoration percentage $R_j$, $j\neq 1,2,\ldots,r$.

6. Select the first pair of trace signal list $(T_{re},T_{ro})$ in $O_l$.

7. Select $n$ flip-flops from $T_{re}$ as trace signals for even time-frames.

8. Select $n$ flip-flops from $T_{ro}$ as trace signals for odd time-frames.

We use the following example to illustrate the non-multiplexed trace list pruning method (Algorithm 4.3). Let us consider the set $O_f = \{f_2, f_3, f_5, f_4, f_6, f_7, f_8\}$ in which the flip-flops are ordered in the descending order of unchecked implications using Algorithm 3.1. Suppose the trace width, $n=2$ and the total flip-flops $N=8$. Thus $l=1$ since $n=2=2^1$. Let $k=2$, hence the search space, $s = 2^2=4 < N$, i.e., we consider the first 4 flip-flops only. The number of iterations, $r=s/n=4/2=2$. During the first iteration, we consider $(f_2, f_3)$ and place it in $T_1$. We perform state restoration for the first five vectors. Let the restoration percentage obtained for all gates in the fifth vector, $R_1=78\%$. During the second iteration, we consider $(f_5, f_4)$ and place it in $T_2$. Let $R_2=80\%$. Since $R_2>R_1$, we choose $T_2$ as our trace signal list.

Next, we illustrate the multiplexed trace list pruning method (Algorithm 4.4) using the following example. Let us consider two sets $T_1= \{f_2, f_3, f_5, f_4\}$ and $T_2= \{f_1, f_6, f_7, f_8\}$ obtained using Algorithm 4.1. Consider the same trace width, $n=2$ and the same total number of flip-flops $N=8$. Thus $l=1$ since $n=2=2^1$. Let $k=2$, hence the search space, $s = 2^2=4 < N/2$, i.e., we consider the first 4 flip-flops for each of the sets $T_1$ and $T_2$. The number of iterations, $r=s/n=4/2=2$. During the first iteration, we place $(f_2, f_3)$ from $T_1$ in $T_{1e}$ and $(f_1, f_6)$ from $T_2$ in $T_{1o}$. We perform state restoration for the first five vectors using $T_{1e}$ and $T_{1o}$. Let the restoration percentage obtained for all gates in the fifth vector, $R_1=78\%$. During the second iteration, we place $(f_5, f_4)$ from $T_1$ in $T_{2e}$ and $(f_7, f_8)$ from $T_2$ in $T_{2o}$. Let $R_2=80\%$. Since $R_2>R_1$, we choose $T_{2e}$ and $T_{2o}$ as our multiplexed trace signal lists.
4.5. Experimental Results

The above algorithms were written in C++ and experiments were conducted for ISCAS’89 sequential benchmark circuits on a Linux workstation with 2GB RAM. The results are reported in Tables 4.1-4.4. During state restoration, we do not assume any knowledge of an initial state other than the traced signals. We consider five different trace buffer widths: 8, 16, 32, 64 and 128. The trace buffer depth is assumed to be 100. Note that since more values can be learned later from the earlier vectors, less restoration is possible in the beginning and more restoration toward the end of the vector sequence. Therefore, each random pattern used for our experiments has 100 vectors each for a more competitive comparison in contrast to [1] in which the trace buffer depth is assumed to be 4k. In Table 4.1, we briefly summarize the results obtained for non-multiplexed trace selection scheme discussed in Chapter 3 for trace buffer width of 8. For each circuit in Table 4.1, $H_1$ contains the result using forward learning and considering that primary input values are known (FL+PI), $H_2$ contains the result obtained using forward and backward learning and considering both primary input and output values are known (FL+BL+PI+PO). The first sub-column under each column $H_1$ and $H_2$ gives the total number of flip-flops in the circuit, followed by restoration ratio ($RR$) obtained in [1], total restoration percentage (only flip-flops) across 100 vectors ($TR_{FFs}$), total restoration percentage (all gates) across 100 vectors ($TR_{all}$), the restoration percentage (only flip-flops) obtained in the final vector ($FR_{FFs}$), the restoration percentage (all gates) obtained in the final vector ($FR_{all}$) and the restoration ratio ($RR$) obtained by $H_1$ or $H_2$. From Table 4.1, it can be seen that $H_2$ has a better restoration ratio ($RR$) compared to [1]. For example, consider s15850, $H_2$ achieved a high $RR$ of 56.1 as compared to a $RR$ of only 19.93 achieved by [1]. For all further comparison with non-multiplexed scheme, we will use the values obtained using $H_2$. From Table 4.1 it can be observed that for s35932 and s38584, the restoration percentage approaches 100 percent as we approach the final vector even for a trace buffer width of 8. Therefore, we exclude these two circuits from further analysis since they are easy to restore. We consider all other circuits which are difficult to restore for our experiments in this work. Table 4.2 gives the results for multiplexed trace selection using implication-based correlation ($H_3$) for trace buffer widths of 32, 64 and 128. We compare the restoration percentage obtained for
multiplexed scheme using $H_3$ with the restoration percentage obtained for non-multiplexed scheme using $H_2$. We observe that in most of the cases $H_3$ gives a superior restoration percentage than $H_2$. For example, consider s9234 for N=64, $H_3$ achieved a total restoration percentage ($TR_{\text{all}}$) of 94.5% as compared to a $TR_{\text{all}}$ of 92.7% achieved by $H_2$. Moreover $TR_{\text{all}}$ of 94.5% obtained for s9234 for N=64 using $H_3$ is close to 94.7% achieved for the same circuit using $H_2$ for N=128. This means that for N=64 with multiplexing, we are getting results of N=128 without multiplexing. In other words, we are effectively tracing twice as many signals with the same trace buffer size and obtaining a restoration percentage which is equal to the restoration percentage obtained for twice the trace buffer width. From Table 4.2 for N=64 using $H_3$, we can also observe that for s5378, the restoration percentage approaches 100 percent as we approach the final vector. For N=128, fields for s5378 and s9234 are empty because these circuits have less than 256 flip-flops and hence cannot be considered for $H_3$. Table 4.3 gives the results for multiplexed trace selection using multi-node implication-based trace list pruning ($H_4$) for trace buffer widths of 16, 32 and 64. We prune the lists obtained using $H_3$ using Algorithm 4.4. We report results for those corner cases where multi-node implications had a major role to play. From Table 4.3, we can observe that $H_4$ is able to increase the restoration percentage of the corner cases significantly. For example, consider s5378 for N=32, $H_4$ achieved a high restoration percentage ($TR_{\text{all}}$) of 97.1% as compared to 94.3% obtained using $H_3$ only. Figure 4.4 shows the comparison of total restoration percentage for various benchmark circuits using $H_2$, $H_3$ and $H_4$.

Finally, Table 4.4 compares our method (for N=32) with two cases: (1) No flip-flop is traced (only primary input and output values assumed to be known) and (2) Random trace signal selection. Our restoration percentage is superior to both the cases considered. For example, consider s13207, $H_3$ achieved a high restoration percentage ($TR_{\text{all}}$) of 82.8% as compared to only 60.9% and 67.2% obtained by the above two cases. For s15850, $H_4$ achieved a high restoration percentage ($TR_{\text{all}}$) of 96.1% as compared to 91.5% and 92.1% obtained by the two cases.
Table 4.1: Results (H1: non-mux+FL+PI, H2: non-mux+FL+BL+PI+PO)

<table>
<thead>
<tr>
<th></th>
<th>#FFs</th>
<th>RR_{old}</th>
<th>H1</th>
<th>H2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>TR_{FF}</td>
<td>TR_{all}</td>
</tr>
<tr>
<td>s5378</td>
<td>179</td>
<td>14.68</td>
<td>86.3</td>
<td>90.4</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>4.767</td>
<td>77.1</td>
<td>90.5</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>19.93</td>
<td>83.2</td>
<td>91.3</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>-</td>
<td>54.3</td>
<td>61.3</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>19.24</td>
<td>72.9</td>
<td>82.4</td>
</tr>
<tr>
<td>s38417</td>
<td>1636</td>
<td>18.6</td>
<td>21.8</td>
<td>36.3</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>64.0</td>
<td>97.0</td>
<td>97.6</td>
</tr>
</tbody>
</table>

Figure 4.4: Total Restoration % (TR_{all}) for benchmark circuits.
### Table 4.2: Results (H$_2$: non-multiplexed, H$_3$: multiplexed)

<table>
<thead>
<tr>
<th></th>
<th>H$_2$</th>
<th>H$_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TR$_{FF}$</td>
<td>TR$_{all}$</td>
</tr>
<tr>
<td>s5378</td>
<td>88.6</td>
<td>92.2</td>
</tr>
<tr>
<td>s9234</td>
<td>78.8</td>
<td>92.0</td>
</tr>
<tr>
<td>s15850</td>
<td>84.4</td>
<td>92.2</td>
</tr>
<tr>
<td>s13207</td>
<td>76.4</td>
<td>82.2</td>
</tr>
<tr>
<td>s38417</td>
<td>25.8</td>
<td>41.4</td>
</tr>
</tbody>
</table>

### Table 4.3: Results (H$_3$: multiplexed, H$_4$: H$_3$+multi-node impl.)

<table>
<thead>
<tr>
<th></th>
<th>H$_3$</th>
<th>H$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TR$_{FF}$</td>
<td>TR$_{all}$</td>
</tr>
<tr>
<td>s5378</td>
<td>95.6</td>
<td>97.3</td>
</tr>
<tr>
<td>s9234</td>
<td>79.7</td>
<td>92.7</td>
</tr>
<tr>
<td>s15850</td>
<td>84.6</td>
<td>92.4</td>
</tr>
<tr>
<td>s13207</td>
<td>80.2</td>
<td>86.4</td>
</tr>
<tr>
<td>s38417</td>
<td>29.3</td>
<td>44.1</td>
</tr>
</tbody>
</table>

### Table 4.4: Results (No Trace, Random vs. our Algorithms)

<table>
<thead>
<tr>
<th></th>
<th>No Trace</th>
<th>Random</th>
<th>H$_3$</th>
<th>H$_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TR$_{all}$</td>
<td>TR$_{all}$</td>
<td>TR$_{all}$</td>
<td>TR$_{all}$</td>
</tr>
<tr>
<td>s5378</td>
<td>91.6</td>
<td>92.8</td>
<td>94.3</td>
<td>97.1</td>
</tr>
<tr>
<td>s9234</td>
<td>76.8</td>
<td>82.1</td>
<td>92.3</td>
<td>92.3</td>
</tr>
<tr>
<td>s15850</td>
<td>91.5</td>
<td>92.1</td>
<td>92.2</td>
<td>96.1</td>
</tr>
<tr>
<td>s13207</td>
<td>60.9</td>
<td>67.2</td>
<td>82.8</td>
<td>82.8</td>
</tr>
<tr>
<td>s38417</td>
<td>41.2</td>
<td>42.6</td>
<td>46.2</td>
<td>47.1</td>
</tr>
</tbody>
</table>
4.6. Summary

We have proposed a new multiplexer-based trace signal interconnection scheme and a new method for trace signal selection based on implication-based correlation. This approach uses the correlation between the forward and backward implications of flip-flops (trace signals) across two consecutive time-frames. We also proposed a SAT-based greedy heuristic to prune the selected trace signal list further to consider the multi-node implications. Results show that our approach gives a better restoration percentage than previous techniques.
Chapter 5

Trace Compression using Source Transformation over FDR codes

This chapter is organized as follows. Section 5.1 discusses the proposed heuristics and Section 5.2 discusses the hardware implementation of the proposed approach. Section 5.3 reports experimental results, and in Section 5.4 we summarize our observation.

5.1. The Proposed Approach

The proposed approach uses source transformation on top of FDR encoding to compress the trace buffer data. Sections 2.7 to 2.12 give an overview of the various compression concepts which we use. The transformation function converts the captured data into a reduced entropy data set. This makes the data set more amenable for compression using the FDR scheme. The extra hardware required to implement these transformation functions is very low. Three transformation functions are proposed in this work that uses the idea of a) Computing a difference vector of the captured data across time frames; b) Ordering based on probabilistic estimation of the captured data and, c) alternate vector reversal technique. These techniques will be described next.

A. Difference vector transformation: is implemented as a hardware block which outputs the difference vector of the current trace vector with that of the vector in the previous time frame. From our extensive experiments, we observed that there already exists correlation between trace vectors of successive cycles for a given design and this can be exploited to generate a data set with reduced entropy. To quantify this correlation, we conducted experiments with ISCAS’89 benchmark circuits. We applied a large number of random vectors and observed a selected set of trace signals. We then computed the average toggling percentage, $TP_{avg}$ for the $n \times N$ trace data matrix using Equation (2.10). This is shown in Table 5.1. In this table, $n$ represents the number of vectors used (i.e., total number of time-frames) for simulation. The column labeled Average Toggling
Percentage represents the average toggling percentage for different numbers, N, of trace signals selected. In our experiments, we considered N=32, 64, and 128. The trace signal selection algorithm is based on the technique proposed in Chapter 3. From the table, we note that for s13207 on an average only 13% of the selected 64 trace signals toggle for each successive vector. Hence, we form the difference vector matrix by XORing the successive rows of the original n×N matrix. If \( r_1, r_2, r_3, \ldots, r_n \) represent the rows in the original n×N trace data matrix, then \( r_1 \oplus r_2, r_2 \oplus r_3, \ldots, r_{n-1} \oplus r_n \) represents the rows in the n×N difference vector matrix. We use \( T_{\text{diff}} \) to represent the un-encoded difference vector sequence. The compression ratio can be improved by compressing \( T_{\text{diff}} \) instead of \( T \). This is because the \( T_{\text{diff}} \) has longer run of zeroes compared to \( T \) and hence the total number of run-lengths, \( R \) is reduced. This is illustrated using an example shown in Figure 5.1(a) and 5.1(b).

\[
\begin{align*}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{align*}
\]

\( T = 01 \ 001 \ 01 \ 00001 \ 0001 \ 1 \ 1 \ 00 \quad T_{\text{diff}} = 01 \ 001 \ 00001 \ 000001 \ 01 \ 00 \)

(a) (b)

Figure 5.1 Transforming \( T \) to \( T_{\text{diff}} \)

Figure 5.1(a) shows the un-encoded trace data sequence \( T = 01 \ 001 \ 01 \ 00001 \ 0001 \ 1 \ 1 \ 00 \). Clearly \( S_U = 20 \). The run-lengths of zeroes in this un-encoded sequence are 1, 2, 1, 4, 3, 0, 0, and 2. Hence, the total number of run-lengths, \( R \) is 8. Using Fig. 1 we obtain the encoded sequence, \( T_E = 01 \ 1000 \ 01 \ 1010 \ 1001 \ 00 \ 00 \ 1000 \). Clearly \( S_E = 24 \). Thus, for the trace data set \( T \), diagnostic resolution \( (D_R) = S_U/S_E = 20/24 = 0.8 \). Figure 5.1(b) shows the difference vector \( T_{\text{diff}} = 01 \ 001 \ 00001 \ 000001 \ 01 \ 00 \). The run-lengths of zeroes in this un-encoded sequence are 1, 2, 4, 5, 1 and 2. Hence, the total number of run-lengths, \( R \) for \( T_{\text{diff}} \) is 6 which is less than 8 obtained for \( T \). Clearly \( S_U = 20 \). The corresponding encoded sequence \( T_{E} = 01 \ 1000 \ 1010 \ 1011 \ 01 \ 1000 \). Clearly \( S_E = 20 \). Therefore \( D_R = S_U/S_E = 20/20 = 1.0 \) which is greater than 0.8 obtained for \( T \).
Table 5.1: Average Toggling Percentage

<table>
<thead>
<tr>
<th>Ckt.</th>
<th>#Vectors, n</th>
<th>Average Toggling Percentage, TP&lt;sub&gt;avg&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of Trace Signals, N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>s5378</td>
<td>1k</td>
<td>9</td>
</tr>
<tr>
<td>s9234</td>
<td>1k</td>
<td>12</td>
</tr>
<tr>
<td>s13207</td>
<td>1k</td>
<td>8</td>
</tr>
<tr>
<td>s15850</td>
<td>1k</td>
<td>8</td>
</tr>
<tr>
<td>s38584</td>
<td>1k</td>
<td>22</td>
</tr>
</tbody>
</table>

**B. Efficient ordering of trace flip-flops:** The ordering of the trace flip-flops can affect the compression quality, i.e., different flip-flop order can lead to different compression percentage. In this section, we describe an algorithm to determine a flip-flop order that maximizes the achievable compression percentage. Note that this is a hard problem because we have to determine one flip-flop order that yields an overall good compression percentage for any real time data being captured. The caveat here is that if there are \( N \) trace flip-flops then there are \( N! \) possible ways to order them. Now, a flip-flop order that produces a good compression percentage for one functional vector may not generate similar result for another vector. Moreover, when \( N! \) becomes large, the computational complexity also increases. So, this is an optimization problem and one can solve it using meta-heuristics such as genetic algorithm or simulated annealing. But our evaluation suggested that a simple probability-based algorithm can give good results for efficient flip-flop ordering, which is described next.

Our approach is to first compute the probability that the *difference vector* bit value for trace flip-flop \( k \) takes a Boolean 0 (1). This is denoted by \( P_{0k} (P_{1k}) \). Note that depending on the design and the cone of logic feeding into the trace flip-flop \( k \), it may tend to capture one Boolean value more often than the other. This determines the \( P_{0k}/P_{1k} \) values for the flop \( k \). After computing the \( P_{0k} \) values, we group the trace flip-flops in their decreasing order of the \( P_{0k} \) values. Now, probability computation can be either vectorless or vector-based. We can use a vector-less symbolic simulation based approach using BDDs to compute the probability value. This will be computationally expensive. Hence, we used a vector-based approach. We simulated the vector to obtain the difference vectors and computed the probability values. When the flip-flop captured an ‘X’, we assumed a random Boolean value in our software.
To understand how ordering the flops affect the entropy, let us consider the \( n \times N \) difference vector matrix where a row represents a difference vector. Ordering the flops based on probability will push the Boolean 0 values, probabilistically, towards the left portion of the matrix leaving the 1’s in the right portion. This operation increases the run length of 0. The impact of this ordering w.r.t. entropy and theoretical maximum compression \( C_{\text{max}} \) are shown in the plots in Figure 5.3 and Figure 5.5 respectively. Let \( T_{\text{order}} \) represent the un-encoded ordered trace data sequence. From Figure 5.3, we can see that \( T_{\text{order}} \) reduces the entropy as compared to the \( T_{\text{diff}} \) for all circuits. From Figure 5.5, we can see that \( T_{\text{order}} \) increases \( C_{\text{max}} \) as compared to the \( T_{\text{diff}} \) for all circuits. We will illustrate the use of this transformation function using the example shown in Figure 5.2(a) and 5.2(b). From Figure 5.2(a) we can obtain the probabilistic parameters, i.e., \( P_{01} = \frac{3}{4} = 0.75, \quad P_{02} = \frac{3}{4} = 0.75, \quad P_{03} = \frac{3}{4} = 0.75, \quad P_{04} = 1.0, \quad \text{and} \quad P_{05} = \frac{2}{4} = 0.5. \) The columns are ordered in the descending order of \( P_{0k}, k=1, 2, 3, 4, 5 \) as shown in Figure 5.2(b). From Figure 5.2(b), the un-encoded sequence \( T_{\text{order}} = 001 \ 01 \ 00001 \ 000001 \ 01 \ 0. \) The run-lengths of zeros in \( T_{\text{order}} \) are 2, 1, 4, 6, 1 and 1. Also \( S_L = 20, \quad T_E = 1000 \ 01 \ 1010 \ 110000 \ 01 \ 01 \) and \( S_E = 20. \) Therefore, \( CR = S_L/S_E = 20/20 = 1.0. \)

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
\end{array} \quad \begin{array}{cccc}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
\end{array}
\]

\( T_{\text{diff}} = 01 \ 001 \ 00001 \ 000001 \ 01 \ 00 \quad T_{\text{order}} = 001 \ 01 \ 00001 \ 0000001 \ 01 \ 0 \)

(a) (b)

Figure 5.2 Transforming \( T_{\text{diff}} \) to \( T_{\text{order}} \)

C. Alternate vector reversal: is implemented as a hardware block. It is an elegant technique which reduces the entropy by further increasing the length of run of zeros across time-frames. To understand this transformation, let \( r_1, \ r_2, \ r_3, \ldots, \ r_n \) represent the rows in the \( n \times N \) reordered trace data matrix. The alternate vector reversed trace data matrix is obtained by reversing the alternate rows in the ordered trace data matrix. This method is effective because reversing of alternate rows groups together the 0-biased flip-flops in those rows, thus increasing the run-length of zeroes in the un-encoded sequence. To understand this, consider two rows \( r_1 \) and \( r_2 \). Since the trace flip-flops are ordered, the
left portions of the vectors will have cluster of Boolean 0’s as described in Section 5.1 B. If \( r_1 \) is reversed and concatenated with \( r_2 \), we will have a longer run of 0 compared to just concatenating \( r_1 \) with \( r_2 \). Thus, either \( \{ r_1, R(r_2), r_3, R(r_4), \ldots \} \) or \( \{ R(r_1), r_2, R(r_3), r_4, \ldots \} \) can be used to represent the rows in the alternate vector reversed trace data matrix. Here \( R(r_k) \) means reverse of \( r_k \). We use \( T_{\text{reversal}} \) to represent the un-encoded alternate vector reversed trace data sequence.

Figure 5.4(a) and Figure 5.4(b) show the usage of alternate vector reversal technique where rows 2 and 4 of Figure 5.4(a) are reversed. From Figure 5.4(b), \( T_{\text{reversal}}=001 \ 01 \ 1 \ 00000000001 \ 01 \ 0 \). The run-lengths of zeroes in this un-encoded sequence are 2, 1, 0, 10, 1, and 1. Thus, the longest run of zero is 10 which is greater than 6 obtained for \( T_{\text{order}} \). Also \( S_U=20, \ T_E=1000 \ 01 \ 00 \ 110100 \ 01 \ 01, \ S_E=18\) and \( D_R=20/18=1.1 \).

The impact of alternate vector reversal w.r.t. entropy and \( C_{\text{max}} \) are shown in the plots in Figure 5.3 and Figure 5.5 respectively. From Figure 5.3 and Figure 5.5, we can see that alternate vector reversal reduces the entropy and increases \( C_{\text{max}} \) as compared to re-ordering for all circuits.
D. Decoding the data: The decoding of the trace buffer data in order to obtain the source transformed data is done off-chip using the decompression architecture described in [31] and [32]. After decoding the FDR code-words, we perform reversing, reordering, and XORing to obtain the original trace data.

Fig. 5.5: Cmax % for various benchmark circuits (N=32)

5.2. Compression Hardware

We implemented the proposed source transformed FDR-based trace compressor at RTL as a two-staged pipelined architecture (see Figure 5.6). There are four main blocks in our hardware implementation: a) Source transform block that implements the source transforms described in Section III, b) FDR FSM block that implements the FDR code using little hardware overhead, c) Control block that generates the required signals for trace compressor operation, and d) Trace buffer which is a memory implemented as a
regular SRAM. These blocks will now be described in detail next. Note that we use the trace selection method proposed in Chapter 3 to select the set of $N$ flip-flops to be traced.

A. Source transformation: As described in Section 5.1, we use three source transformations to decrease the entropy of the data to be encoded. From hardware point of view, the source transform block receives ordered parallel data from the trace flip-flops and converts it into difference vector and alternate vector-reversed parallel data. We will illustrate the generation of ordered parallel data using an example. Suppose we select four trace flip-flops labeled $f_1, f_2, f_3, f_4$ using the method proposed in Chapter 3. For our example, $N=4$. Next, we use software (a C++ program) to order these flip-flops based on the method described in section 5.1 B. The order generated by the software is communicated to the designer, which is then used for designing the hardware. For our example, suppose the order determined is \{ $f_2, f_1, f_4, f_3$ \}. Let $d[3:0]$ be the source transform block input data bus. To reflect the new order on hardware, the designer only needs to connect $f_2$ to $d[0]$, $f_1$ to $d[1]$, $f_4$ to $d[2]$ and $f_3$ to $d[3]$. Hence, we only incur routing overhead to generate the ordered data. To generate the difference vector we use $N$ flip-flops to store the trace vector in the previous time frame and $N$ XOR gates to generate the Boolean difference. To reverse the alternate vectors, we use $N$ multiplexers whose select signals are driven by a T-flop.

B. FDR FSM: The FDR can be implemented as a finite-state-machine (FSM). But synthesizing this code may cost more area overhead. Hence, we present an optimization where the overall design and source transform block is made to work at a lower frequency, $f_{source}$, compared to the clock frequency, $f_{FDR}$, of FDR FSM. This allows increasing the sequential depth of the FDR FSM by more than one that, in turn, reduces the hardware overhead to implement this block. For all of the cores in a design, this can be realized as follows. For cores in the design that operate at a lower frequency compared to the fastest core of the ASIC, we can use a separate compressor using the fastest clock of the design. For fastest cores in the design, we visualize them as a high throughput system. We convert a high throughput application into a low throughput application by lowering the clock frequency only during the silicon debug phase. This can be implemented using additional clock divider circuits that can be enabled during the silicon debug phase. This would still be valid if we are trying to catch functional/logical bugs.
and not speed-related defects for cores using the fastest clock. Speed related defects have to be exposed using diagnostic techniques using at-speed test vectors. With these assumptions, we will now describe our hardware implementation for this block. In our description, we refer a slow_clk as the clock signal used by the system and the source transform block, and fast_clk as the clock used by FDR FSM. Whenever new data becomes available at the output of source transform block, the start signal goes HIGH for one fast_clk cycle and then remains LOW. This interrupt pulse generated by the control block indicates FDR FSM block to start encoding. The encoded codeword is put on a serial output port on encountering each run of zeroes. The ready signal remains HIGH during the time when the encoded codeword is being serially shifted out. The ready signal can be used to generate the write enable signal wen and the address addr for writing into the trace buffer memory. The address addr is simply incremented for each encoded bit being serially shifted into the trace buffer through the sin pin. When the parallel trace data at the compressor input is completely processed (encoded), the done signal becomes HIGH and it remains HIGH until it encounters another start interrupt pulse indicating the arrival of new data at compressor input.

Figure 5.6: Trace compressor architecture
C. Control Block: This block generates the control signals necessary for the compression when the \texttt{enc\_en} signal is asserted on entering the silicon debug mode. The control logic for bit-reversal is also generated by this block.

5.3. Experimental Results

The technique proposed in Chapter 3 was written in C++ that was used for trace signal selection and state restoration. The FDR algorithm (refer Section 2.8), the source transformation functions (refer Section 5.1) and the quality metrics (refer Section 2.10) were also written in C++ and experiments were conducted for ISCAS’89 sequential benchmark circuits on a Linux workstation with 2GB RAM. The results are reported in Tables 5.2. Each vector used for our experiment comprised 1000 clock cycles. Table 5.2 gives the results for trace signal counts of 32, 64 and 128 for five benchmark circuits. For each circuit, the first column gives the total number of flip-flops in the circuit, followed by the compression percentage ($C$) containing four sub-columns $T$ (existing FDR), $T_{\text{diff}}$ (difference vector), $T_{\text{ord}}$ (difference vector + ordering) and $T_{\text{rev}}$ (difference vector + re-ordering + bit-reversal) respectively. The values under sub-column $T$ reflects the compression percentage obtained using the FDR code-based compression technique. The values under sub-columns $T_{\text{diff}}$, $T_{\text{ord}}$ and $T_{\text{rev}}$ reflect the compression percentage obtained after using our proposed source transformation techniques. The last two columns give the theoretical maximum compression percentage ($C_{\text{max}}$) and the diagnostic resolution ($D_R$) for the proposed approach. We see that the compression percentage obtained using the proposed approach is close to the theoretical maximum compression percentage in all cases. For example, consider s15850 of Table 5.2 with 32 traced signals; we achieved a compression percentage of 63\% by using all the three transformation functions as compared to a compression percentage of 29\% achieved by using the FDR codes without any source transformation. This trend is also shown in Fig. 5.7. In addition, this value is close to the theoretical maximum compression percentage ($C_{\text{max}}$ \%) of 66.8\%. This trend is shown in Figure 5.3 and Figure 5.5. We observe from Figure 5.3 that our proposed transformation functions reduce the entropy of the trace data and hence led to better compression.
We also observe that for some circuits we are able to achieve a diagnostic resolution ($D_R$) greater than 3X. For example, consider s9234 of Table 5.2 with 64 traced signals. We achieved a diagnostic resolution of 3.3X which means that with a trace buffer width of 64/3.3≈20, we are able to trace 64 signals. In other words, the diagnostic capability of the trace buffer is improved by 3.3X. Finally, we compare the compression % obtained by our method with the compression % obtained by using GZIP over the difference vector. This is shown in Table 5.3. We chose GZIP because it represents the state-of-the-art implementation of LZ77 and Huffman coding algorithm that is used by Anis et al. [14]. For all circuits, our proposed method gives better compression percentage than GZIP as the trace signal counts ($N$) increased in our experiments. For example, consider s13207 of Table 3 with 64 traced signals; we achieved a compression percentage of 57% as compared to a compression percentage of 48% achieved by using GZIP.

To evaluate the area overhead, we implemented the trace compressor hardware using Verilog and synthesized it using a commercial synthesis tool for three different compressor input counts. The area overhead of the source transformation block is negligible. We do not incur any area overhead for ordering since we do it using software. Only routing overhead is incurred. For the difference vector and reversal functions, we incur an overhead of $N$ XOR gates, $N$ flip-flops and $N$ multiplexers which is small. The area overhead (in NAND gate equivalent) of the trace compressor as a percentage of the
trace buffer area is shown in Table 5.4. For each trace signal count \( N \), the first column gives the trace buffer area \( M \) in NAND gate equivalent, followed by the trace buffer width \( T_{\text{B.Width}} \), the trace buffer depth \( T_{\text{B.Depth}} \) and the compressor area \( A \) in NAND gate equivalent, respectively. The last column gives the compressor area as a percentage of trace buffer area. We observed that the area overhead of the trace compressor is negligible as compared to the trace buffer area. For example, for a compressor input count of 128, the trace compressor area of 3.1K is only 0.5% of the trace buffer area (576K). In [39] it was reported that the area overhead of a LZ-based compressor was around 50k equivalent NAND gates. Also note that the main limitation of LZ77-based method proposed in [14] was related to the large area overhead involved owing to the use of different content-addressable memory (CAM) sizes.

Table 5.2: Compression results (#Vectors=1000)

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#FFs</th>
<th>( C% )</th>
<th>( C_{\text{Max}}% )</th>
<th>( D_{R} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( T )</td>
<td>( T_{\text{diff}} )</td>
<td>( T_{\text{ord}} )</td>
</tr>
<tr>
<td>s5378</td>
<td>179</td>
<td>-5.7</td>
<td>54</td>
<td>60</td>
</tr>
<tr>
<td>s9234</td>
<td>211</td>
<td>-45</td>
<td>42</td>
<td>53</td>
</tr>
<tr>
<td>s15850</td>
<td>534</td>
<td>-29</td>
<td>56</td>
<td>60</td>
</tr>
<tr>
<td>s13207</td>
<td>638</td>
<td>-43</td>
<td>51</td>
<td>60</td>
</tr>
<tr>
<td>s38584</td>
<td>1426</td>
<td>10</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

\( N=32 \)

\( N=64 \)

\( N=128 \)

Table 5.3: GZIP vs. Our approach

<table>
<thead>
<tr>
<th>( N )</th>
<th>GZIP(C%)</th>
<th>( T_{\text{reversal}}(C%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 64 )</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>s5378</td>
<td>41</td>
<td>-</td>
</tr>
<tr>
<td>s9234</td>
<td>64</td>
<td>56</td>
</tr>
<tr>
<td>s15850</td>
<td>68</td>
<td>61</td>
</tr>
<tr>
<td>s13207</td>
<td>48</td>
<td>50</td>
</tr>
<tr>
<td>s38584</td>
<td>17</td>
<td>20</td>
</tr>
</tbody>
</table>
Table 5.4: Area overhead of compressor

<table>
<thead>
<tr>
<th>$N$</th>
<th>TB Area (M)</th>
<th>$TB_{\text{width}}$</th>
<th>$TB_{\text{depth}}$</th>
<th>Compressor Area (A)</th>
<th>$(A/M %)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>145k</td>
<td>16</td>
<td>1024</td>
<td>2.0k</td>
<td>0.1%</td>
</tr>
<tr>
<td>64</td>
<td>290k</td>
<td>32</td>
<td>1024</td>
<td>2.4k</td>
<td>0.8%</td>
</tr>
<tr>
<td>128</td>
<td>576k</td>
<td>64</td>
<td>1024</td>
<td>3.1k</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

Figure 5.8: Actual Compression Percentage for various benchmark circuits, N=64

Figure 5.9: Actual Compression Percentage for various benchmark circuits, N=128
Figure 5.10: Entropy for various benchmark circuits (N=64)

Figure 5.11: Entropy for various benchmark circuits (N=128)
5.4. Summary

Trace signal selection is critical to increase the observable window during the debug phase. We presented a novel compression technique for trace buffers that can significantly enhance this observable window. Source transformations proposed in our work can improve the effectiveness of FDR codes with very little extra hardware overhead and help achieve close to theoretical maximum compression. Results indicate that our approach gives a better compression percentage and diagnostic capability than a state-of-the-art implementation of LZ77. Moreover, the area overhead of our trace compressor is significantly less compared to dictionary-based codes.
Chapter 6

Conclusion and Future Work

In this thesis, we have addressed an important challenge being faced by the semiconductor industry today - efficient and fast identification of root cause of silicon failures. This is very critical due to the increasing demand for shorter time-to-market. In-system silicon debug techniques like trace buffer-based technique are normally used to discover these undetected bugs and defects. However, due to limited availability of the trace buffer memory, it is important to use the available resource in the most efficient manner. In Chapter 3, we proposed an unchecked implication-based technique to drive the selection of critical trace signals which restore the maximum number of untraced signals. We also introduced an algorithm which uses a SAT-based multi-node implication engine to restore the values of untraced signals across multiple time-frames. We showed that our method is able to achieve a better restoration percentage than previous techniques. In Chapter 4, we proposed a new multiplexer-based trace signal interconnection scheme. We proposed a new heuristic based on implication-based correlation to intelligently select two sets of signals to be traced in even and odd time frames respectively and also introduced a state restoration algorithm for this scheme. We also proposed a SAT-based greedy heuristic to prune the selected trace signal list further to take into account the corner cases where multi-node implications play a major role. Experimental results showed that this new scheme is able to achieve a better restoration percentage than previous techniques. In Chapter 5, we proposed a compression scheme using source transformation techniques over Frequency-Directed Run-Length (FDR) codes in order to increase the capacity of the trace buffer. Source transformation reduces the entropy of the data to be compressed and hence, improves the compression percentage. Experimental results showed that the proposed method gives a better compression percentage compared to dictionary-based techniques. We also implemented the method on hardware and observed that the area overhead of the compressor is less compared to dictionary-based techniques and yields up to 3X improvement in the diagnostic capability.
Future Work:

The multiplexed trace signal selection scheme could be enhanced by considering multi-node implications during the formation of the implication-based correlation model itself. This could be an efficient alternative to the technique proposed in Chapter 4 in which a greedy heuristic was used to consider multi-node implications. Another scope of future work could be to consider the use of source transformation functions proposed in Chapter 5 over *enhanced* FDR codes [53] for trace compression wherein both run-lengths of 1s as well as 0s can be considered instead of only considering run-lengths of 0s.
Bibliography


