Novel Architectures for Trace Buffer Design to Facilitate Post-Silicon Validation and Test

Shuchi Pandit

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
A. Lynn Abbott
Chao Wang

April 30, 2014
Blacksburg, Virginia

Keywords: State Restoration, Invariants, Trace Buffer Architecture

Novel Architectures for Trace Buffer Design to Facilitate Post-Silicon Validation and Test

Shuchi Pandit

(ABSTRACT)

Post-Silicon validation is playing an increasingly important role as more chips are failing in the functional mode due to either manufacturing defects escaped during scan-based tests or design bugs missed during pre-silicon validation. Critical to the diagnosis engineer is the ability to observe as many relevant internal signal values as possible during debug. To do so, trace buffers have been proposed for enhancing the observability of internal signals during post-silicon debug. Trace Buffers are used to trace (record the values of) the internal signals in real-time when chip is in its normal operation. However, existing trace buffer architectures trace very few signals for a large number of cycles. Thus, even with a good subset of signals traced, one often still cannot restore all the relevant values in the circuit. In this work, we propose two different flexible trace buffer architectures that can restore the values for all signals by making the trace signals configurable. In addition, the buffer space can also be shared among different traced signals which makes the architectures highly flexible. As compared to conventional trace buffer architectures, the new architectures have comparable area overhead but offer the ability to restore all signals in the circuit. For cases of less than 100% restoration, the ability of circuit invariants to improve the signal restoration is explored. A promising direction for the future work is provided where targeted invariants may lead to better restoration scenario during post-silicon validation.

Acknowledgements

First of all, I would like to thank my Research Advisor Dr. Michael S. Hsiao for giving me the opportunity to work with him and keeping confidence in me throughout. I am grateful for his immense guidance in my research and his valuable ideas that form the foundation of this work. While working with him, I got an opportunity to learn the nuances of the field of Testing and Verification of Digital Systems, and also developed better understanding on other aspects of Computer Engineering. His advice on delving deeper into the research topic and being curious and excited about the work, has certainly inculcated in me the better sense of research and has helped me build self-motivation for my work.

I would like to thank Dr. Chao Wang and Dr. A. Lynn Abbott for serving on my thesis committee. The course on Verification of Digital Systems, taught by Dr. Chao Wang, has helped me employ better ideas in my work.

I would like to extend my thanks to Loganathan Lingappan and Vijay Gangaram from Intel Corporation. They were closely involved in the work and I was able to receive many valuable ideas through our meetings.

I would like to thank Sarvesh Prabhu for his equally enthusiastic participation in this work. He provided me with invaluable guidance as we worked together through the project.

I am very thankful to all the present and former members in my lab, for building a strong working environment in the lab blended with fun-filled moments.

Thanks to all my friends at Virginia Tech who made this journey of mine, a memorable one. Special thanks to my close friends for being my support system throughout.

I would like to express deep gratitude to my grandmother Mrs. Indira Pandit and my parents Mrs. Mamta Pandit and Mr. Mahadeo Pandit who kept faith in me and supported me as I embarked on this career path. I thank my brother Prashant Pandit for being a source of inspiration for me, always.

The research was supported in part by a gift from Intel.

Shuchi Pandit
April 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Post-Silicon Validation: An Introduction

The competitive semiconductor industry today deals with extremely complex digital designs which are not only difficult to model but also challenging to verify, test, and diagnose. As the time to market is decreasing rapidly, some corner-case design bugs may escape the pre-silicon validation step [1] or a number of hard-to-test defects that require multi-cycle at-speed testing may escape the scan-based manufacturing test. Thus, post-silicon validation has a very significant role to play in diagnosing these observed errors. Scan-based approach for post-silicon validation was introduced in the late 1990's [2] and has been widely adopted in industry due to its advantage of reusing the on-chip DFT architecture without requiring additional area overhead. Owing to the drawback of scan-based approach of stopping the normal operation of circuit to dump the internal states in each cycle (which may miss those defects that may require multi-cycle test application), the scan-based approach gave way to the new trace-based approach [6] which can be used to trace the real-time system response. In such an approach, the states of the selected internal signals are stored in an on-chip trace buffer; at an instance of error occurrence, the circuit operation is stopped and internal signal states stored in the trace buffer are off-loaded and analyzed for debug.

The size of the trace buffer required is determined both by its width (i.e., the number of signals selected for tracing) and its depth (i.e., number of cycles for which the signals are traced). Presently, the trace buffer architecture is of narrow-and-deep organization, where very few signals are traced for a large number of cycles. The small number of selected signals

are connected to the centralized trace buffer using the fixed on-chip routing. As the width of the trace buffer is very small, there is a need to select an extremely good set of trace signals for maximum restoration of signal states over the trace cycles. The situation is even more challenging as the trace signals need to be predicted and fixed during the pre-silicon design phase where knowledge of post-silicon conditions is not available. Once fixed, changing the set of trace signals would not be possible during diagnosis. Most of the fixed narrow-and-deep architectures have the trace buffer width set to less than 100 with the depth on the order of a few thousand cycles. The fixed architectures cannot be reconfigured during the post-silicon phase.

However, when a chip exhibits an erroneous behavior in the functional mode (and during the post-silicon debug), the knowledge of *all* relevant internal signal values near the occurrence of the error is critical for efficient debugging. In addition, there is a need to locate the cause of error which can be many cycles before the actual observation of error. Fig. 1.1 illustrates a scenario where an error is observed after n cycles of execution. The signals in the last k cycles are very critical to the diagnosis engineer to guide the search for the bug/defect. Frequently, near-100% restoration of all the internal signals in these last k cycles are crucial to debugging the chip. Also, to be able to learn the cause of error on the chip, restoration of the signals which point to the root of the cause of error may require an observation over a longer trace of m ($m > k$) cycles.

To satisfy both objectives, we propose two novel, flexible trace buffer architectures, which can be very effective for signal restoration in a post-silicon scenario.

The first architecture *extends* the current scan architecture [2] by adding a dedicated small k-bit register with each scan cell. The width of the proposed trace buffer is configurable; up to all scan flip-flops can be traced. The number of cycles that can be traced will depend on the set of trace signals. To minimize overhead, we propose the use of a small shift-register with each (scan) flip-flop (FF) to buffer the values during the trace cycles. In addition, whenever a FF is not traced, its corresponding shift-register may be used to buffer other traced signals. As compared to the conventional trace buffer architectures, this new architecture gives a better restoration; all signals may be restored in a single run in the best case.

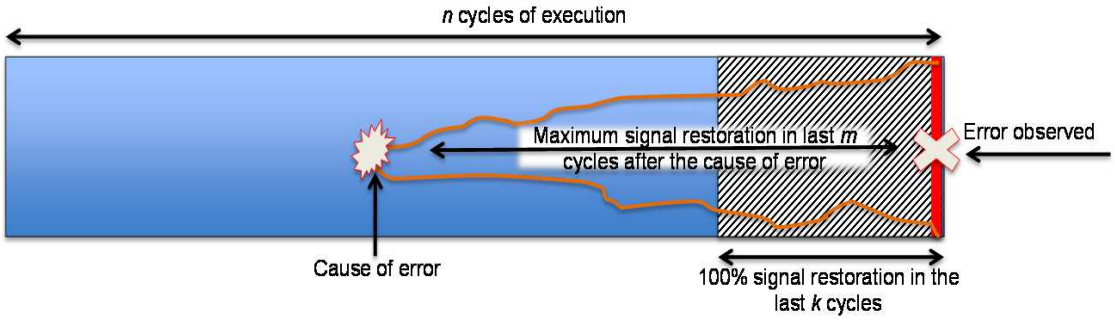


Figure 1.1: Error Occurrence Scenario During Post-Silicon Validation

Experimental results show that, for the same buffer capacity, the number of unrestored signals in the last k cycles of the execution is nearly 0 (i.e., nearing 100% restoration) in the proposed architecture. We study the effect of the number of the traced signals to the restoration quality. In particular, we wish to know how much we need to trace in order to obtain 100% restoration in a single application of the erroneous trace. Next, the proposed architecture allows tracing a smaller percentage of FFs and we get the advantage of higher number of trace cycles for some FFs which gives better restoration.

Finally, when we trace fewer signals over a longer depth, the proposed architecture has an ability to trace various combinations of signals in different debug runs and still restore all the internal signals in the circuit facilitating debug to a great extent. Compared to the related previous work, not only are we able to restore nearly all signals, the flexibility of the proposed architecture also makes the quality of trace signal selection less critical as the diagnosis engineer is capable of tracing different signals according to debug needs.

The second proposed architecture also facilitates wider tracing, but unlike first architecture, where each FF is provided with a shift register, this architecture has a provision of one shift register for a small group of FFs. When only one FF in a group is traced, the shift register for the group is used to store the trace values for the given FF. If two or more FFs in a group are traced, multiplexed tracing is performed, i.e., the selected FFs in the same group are traced in alternate cycles. For this second proposed architecture, the effect of multiplexed tracing on the signal restoration is investigated.

The experimental results reflect that the multiplexed architecture with group sizes of 4, 8 and 16 perform better than the case when no grouping is performed. Various group sizes differ in performance, based on the percentage of FFs traced and the number of last k cycles

where restoration is noted. Second experiment compares the first and second architecture in different grouping configurations and for different values of k, where restoration in last k cycles is noted. In different scenarios, the performance of first and second architecture varies and is illustrated in the results.

Our trace buffer architectures are compatible with any trace signal selection algorithm. We have used a method similar to [8] with an improved strategy in our experiments. In [8], the trace signal selection is based on the highest number of unchecked implications of a signal in single timeframe instance of the circuit. An implicant, x, is unchecked when no other implicate (in our case a selected trace signal) implies on x. In this work we consider the forward and backward implications from a signal up to 3 time-frames.

The two novel trace buffer architectures proposed, increase the state restoration in post-silicon scenario to a great extent. If signal restoration is enhanced further, it can be very useful for post-silicon debug. In this thesis work state restoration is performed using an implication-based restoration engine. The circuit is unrolled for a given number of times frames and then is converted to the conjunctive normal form (CNF). The traced values are updated in the CNF and signals are restored using recursive Boolean Constraint Propagation (BCP) which is performed until further restoration is not possible. Appending useful clauses to the CNF of a circuit can aid in reducing the search space to a great extent and perform better signal restoration.

Thus, the final part of this thesis focuses on finding Circuit Invariants and how they could be used to facilitate the signal restoration in post-silicon validation. Circuit invariants are the relations among different signals in the circuit that hold true for all times. Useful invariants in the circuit were identified and analyzed to determine if they can aid signal restoration. The motivation behind the work is to maximize the number of signals restored so that more debug information is available during defect or error analysis on the chip. Methods to reduce the number of potential invariants were also employed.

Experimental results in this chapter provide the number of 2-node and 3-node invariants that are mined for four ISCAS'89 benchmark circuits. For the case of 2-node invariants, number of potential invariants is reduced and is highlighted in the results. The results on restoration is noted when the constants, 2-node invariants and 3-node invariants are added to the circuit formula. The restoration is improved when constants and 2-node invariants are added to the circuit formula. The percentage reduction in number of unrestored signals

in last k cycles is shown in the results.

1.2 Contributions

Following are the major contributions of this thesis work:

1. The efficiency of wider tracing has been investigated, to maximize the restoration of the signals in the most critical cycles of debugging.

2. Two novel trace buffer architectures are proposed that are highly flexible and configurable. The idea of wider tracing is preserved in both the architectures and the flexibility for deeper tracing is provided. Thus the merits of tracing wide-and-shallow and narrow-and-deep can be combined and be useful for various debug needs.

3. The useful invariants from the circuit are mined and a study is performed, evaluating the efficiency of the mined invariants for increasing the signal restoration using the implication based restoration engine.

1.3 Organization

The rest of the thesis is organized as follows:

Chapter II presents the background and previous related work in the area of signal restoration for post silicon validation. It also explains why wider tracing is preferred for the new architectures. We also highlight the preliminaries of circuit invariants and work that had been done in the past in this area.

Chapter III gives detailed description of the first proposed architecture where each FF has a dedicated shift register associated with it. Experimental results show how this architecture is better than the conventional architectures.

Chapter IV elaborates on the second proposed architecture where a single shift register is provided to a group of FFs and there is multiplexed tracing. The results highlight the comparison of different variants of the second architecture. The first and second architectures are also compared for different configurations. Area overhead of tracing the primary inputs is determined and presented.

Chapter V provides details on finding useful invariants for a circuit which facilitates the signal restoration. Methods on reducing the number of potential invariants are also discussed.

Chapter VI concludes the thesis work, makes recommendations to enhance the signal restoration and provides scope and directions for future work

Chapter 2

Background

2.1 Previous Related Work

Although post-silicon validation is a relatively new research area, a sizable body of work has already been conducted in this field. The initial work on scan-based architecture for post-silicon debug is elaborated in [2] and [3]. [2] discusses on how their proposed scan DFT architecture can effectively balance testability needs with the design constraints. [3] gives details on combining the scan chains with other modules to incorporate design-for-debug on chip.

The trace buffer architecture for observing the real-time in-system response of the chip and its advantages are discussed in [5] and [6]. The ability to trace the signal values in real-time on chip highly reduces the time to market and efficiently detects the design bugs and defects that escape pre-silicon validation and manufacturing test respectively.

Various algorithms on automated trace signal selection and restorability metrics are proposed in [9], [10] and [11]. Since then, various methods on better trace selection algorithms have been offered. In [8], trace signals are selected based on the number of unchecked implications of a signal. A FF with the highest number of unchecked implications is chosen as the candidate for tracing. Once a signal is selected for tracing, its implications are marked checked and calculation for number of unchecked implications for all other FFs is performed again. FF with highest number of unchecked implications is the next candidate for tracing. In [7], the trace signal selection is based on the extent to which a FF is infected by other FFs and the algorithm uses the size of fan-in cone of a FF to determine the efficient trace signals.

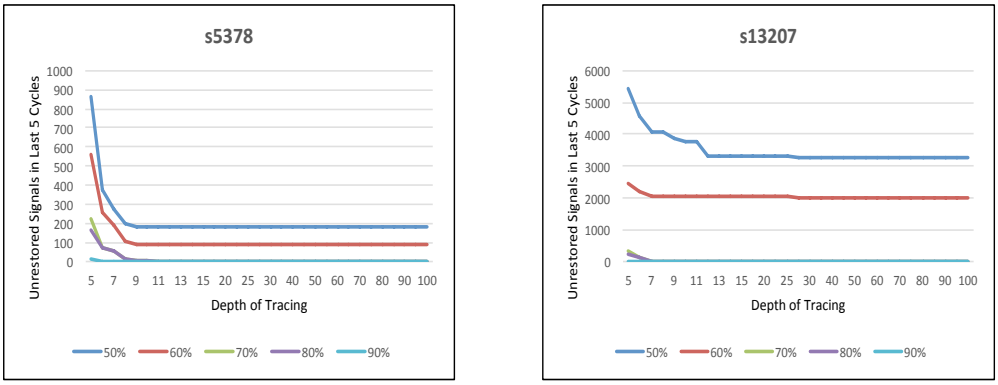
Recent advances in trace signal selection techniques include simulation-based trace signal selection [12], total restorability metric based trace signal selection [13], and hybrid trace selection technique that combines the advantages of both simulation-based and metric-based trace signal selection [14]. In [15], a dynamic trace signal selection method is proposed which gives a good signal restoration regardless of the input test patterns. In [16], the trace signals are selected based on the error visibility metric and an efficient interconnection MUX fabric for selecting a few trace signals out of a number of tap signals is proposed. Other multiplexer based architectures are described in [18] and [19]. The approaches in [20] - [23] describe various algorithms on combining the scan and trace signals for higher restoration. The work in [24] - [28] focusses on the compression of traced data so that it can be stored effectively on the on-chip trace buffer and can be used efficiently for restoration when offloaded.

Unlike the previous approaches where the quality of trace signal selection is critical, our approach also makes trace signal selection less critical to the entire process as the diagnosis engineer is able to trace different signals at will to restore all relevant values.

2.2 Preliminaries

During post-silicon validation, if only the outputs of the chip are observable, it generally is very difficult to diagnose the circuit and locate the root cause of the bug or defect. Although trace buffers offer increased visibility of internal signals, if only a small percentage of the signals is traced, the signal restoration generally would not be able to restore all values. Furthermore, in the presence of manufacturing defects, the restoration of untraced signals is based on the netlist in which the actual manufacturing defect is absent. Thus, the restored value may actually be different from the actual faulty value on the silicon. As a result, most diagnosis engineers would prefer to have all signals observed to allow them to perform a better root cause analysis.

In this section, we describe why wide and shallow trace buffers are preferred for post-silicon diagnosis. First we discuss initial experiments that we performed, results of which validate the hypothesis that wide and shallow tracing can give better restoration in last k cycles. Also the second experiment validates the need of flexible and configurable tracing. Further, the two important scenarios of presence of manufacturing defect and design bug on chip are discussed where wide tracing can be extremely important. In addition, a better metric for



(a) (b)

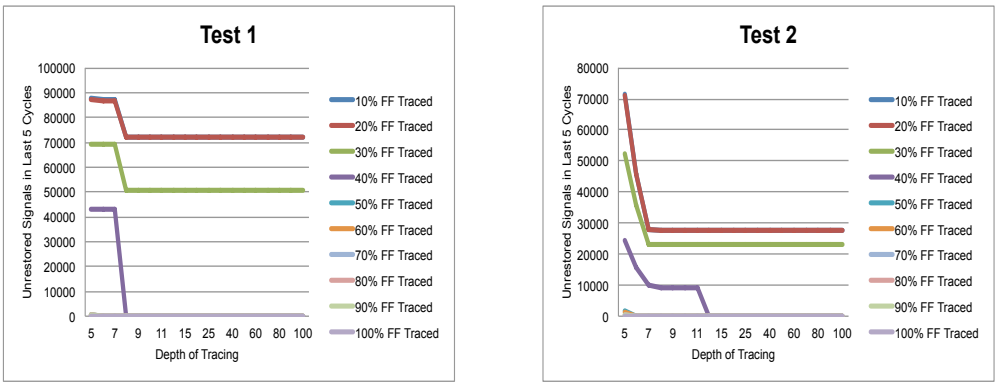
Figure 2.1: Restoration is Last 5 Cycles for Different Trace and Depth Configurations

restoration has been discussed. As against the popular metric of restoration ratio, a new metric which deals with restoration of all signals in the critical cycles has been offered.

2.2.1 Initial Experiments

In the initial phase of the work, after formulating the hypothesis that wider tracing can be very advantageous in signal restoration, experiments were performed to test the efficiency of wide and shallow tracing. For various benchmark circuits, different percentage of FFs were traced for different number of trace cycles. Using the traced values, restoration of other internal signals was performed and number of unrestored signals in last 5 cycles was noted. Figure 2.1 illustrates the results for circuits s5378 and s13207. As seen in the figure, for tracing 50% and 60% FFs, the restoration is improved (number of unrestored signals decrease) as the trace depth is increased to a certain extent. After a point, there is no further improvement in the restoration and a saturation in number of unrestored signals is observed. On the contrary, for cases of tracing 70% and higher, the number of unrestored signals becomes zero and 100% restoration is achieved even with very low depth of tracing. These results suggest that wide and shallow tracing can provide 100% restoration in last k cycles of interest.

Similarly second experiment was performed where two different set of vectors were used and restoration was noted for all the circuits. For some circuits, it was observed that restoration performance is highly dependent on the vector set and the FFs selected for tracing. Fig-



(a) (b)

Figure 2.2: Restoration for Different Vector Sets: aes_core

ure 2.2 illustrates the case of circuit aes_core. It can be observed that restoration differs for the different vector sets when same FFs are traced. This suggests that a configurable trace architecture is required so that FFs to be traced can be selected on the fly and good restoration can be achieved with varying test sets.

As an inference from the two above experiments it was concluded that a good trace buffer architecture design should facilitate wider tracing for 100% restoration in last k cycles and configurability of trace signal selection for a consistent restoration across various vector test sets. The following sections highlight the advantage provided by wider tracing in two important scenarios of presence of manufacturing defect and presence of design bug on chip.

2.2.2 Error due to a manufacturing defect

In Fig. 2.3, the manufacturing defect is activated in time-frame 1, and its effect propagates to different FFs in succeeding frames and eventually is observed at an output in time-frame 4. Let us assume the fault-free output value is 0 and due to presence of manufacturing defect, the observed faulty value is 1. According to the conventional trace buffer architecture, if only a subset of FFs is traced, the restoration of states might be misleading in certain situations. In the given example in Fig. 2.3, if FF_2 alone is traced and the values of other signals are restored, the values restored at the other FFs in the subsequent time-frames may differ from the actual faulty values present in the circuit. Thus, the inaccurate restored values may mislead the diagnosis engineer in tracing back along wrong paths, hurting diagnosis. In

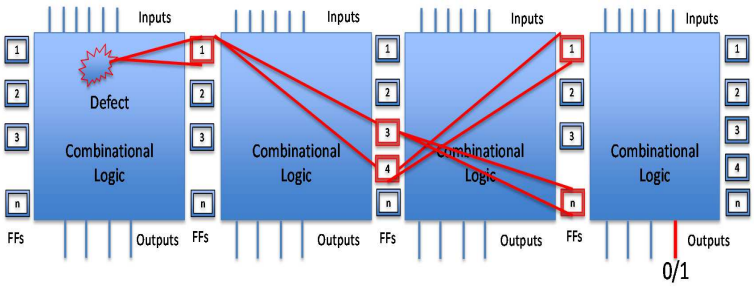


Figure 2.3: Presence of Manufacturing Defect on Chip

contrast, if tracing of a large number of FFs is allowed, the chances of tracing the FFs with the evidence of error is higher and restored values using those trace values would be much more beneficial. In the extreme case of tracing 100% of flip-flops, any mismatched values at the flip-flops in every time-frame can be captured. Note that, while using a faulty value of a signal (caused due to presence of manufacturing defect) for signal restoration, it is possible that implication engine used for the restoration might run into a conflict because the faulty value present may not be a feasible value in the normal operation of the circuit. This makes it even more critical that the actual traced values of signals are observed rather than simply relying on restored values.

2.2.3 Error due to a design bug

While diagnosing a design bug during post-silicon validation, generally the circuit is allowed to run in its application environment and the operation is stopped when an error is observed at an output of the circuit. Since the error is caused by a bug, the values stored in the trace buffer for the last d cycles, where d is the depth of the trace buffer, can be used for the signal restoration of other states. Nevertheless, locating the cause of bug is still a challenging task. In [7], a debug mechanism based on "suspect window" is proposed. Fig. 2.4 illustrates the mechanism of locating the bug using 2 debug runs. The first run is a free run where the trace buffer is operational and based on the trace values a suspect window is determined, which is the set of cycles in which the bug would have been triggered. In the second debug run, another free run is executed, but it only runs till the start of the suspect window. In the suspect window, a scan dump is performed for each cycle and the cause of bug can be

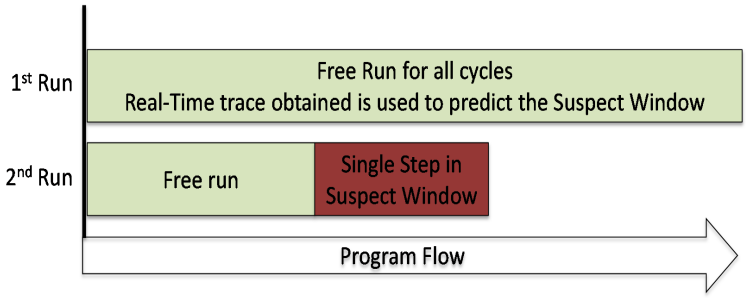


Figure 2.4: Trace-Buffer Debugging Flow as presented in J. Gao, Y. Han, and X. Li, "A new post-silicon debug approach based on suspect window," *in IEEE VLSI Test Symposium,* 2009, pp. 85-90. Used under fair use, 2014

located. In such a scenario, our proposed architecture would perform better. In the first run, our architecture can be configured as a conventional deep-trace buffer and the suspect window can be determined. In the second run, the new architecture does not need a scan dump in each cycle in the suspect window. Rather, as there is a small shift register associated with each FF, a "Multi-Cycle Scan" can be performed. This will reduce the needed debug time to a great extent.

2.2.4 Restoration Ratio Vs Unrestored Signals in the critical cycles

Restoration Ratio (RR) is a popular metric used to determine the restorability of a set of traced signals. In [9] and [10], RR is defined as:

$$RR = \frac{N_{traced(FFs)} + N_{restored(FFs)}}{N_{traced(FFs)}}$$

where $N_{traced(FFs)}$ and $N_{restored(FFs)}$ are the number of traced FFs and number of restored FFs, respectively.

In [8], other metrics such as Total Restoration Percentage for all signals and Total Restoration Percentage for FFs are defined as:

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

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

where $N_{total(all)}$ and $N_{total(FFs)}$ are the total number of signals in the circuit and total number of FFs in the circuit, respectively.

Although these metrics give an idea of the restorability of a given set of traced signals, they do not directly correlate to the debug efficiency. First, a higher value of RR (or TR) can mean a lot of signals are still unrestored. For example, suppose 32 FFs in a circuit are traced and the RR is 20. Then, according to the definition of RR, $N_{restored(FFs)}$ is 608. If the circuit has 1000 FFs, 392 FFs would still be unrestored and may be critical to diagnosing the error. Moreover, for two identical values of RR, there can be two totally different sets of signals restored. One set may contain more signals of interest for a diagnostic engineer. Thus, numeric metrics such as RR and TR may not be ideal.

In this thesis, emphasis has been given on restoring near-100% of all signals in the circuit for the critical cycles that would be demanded by the debugging engineer. Thus, the total number of unrestored signals in the last k cycles of a debug run is calculated, where k is determined as the critical window set by the debug engineer.

2.3 Invariants: Preliminaries and Related Work

For a given system, Invariants are those properties, that do not alter irrespective of any input set applied to the system. Circuit Invariants can be termed as relations among different signals in the circuit that hold true for all the times. Inductive Invariants are the circuit invariants, that hold true in all the reachable states of the circuit. As compared to implications in the circuit that can be found out using the structural information and are relatively easy to mine, invariants are hard to find relations and are mostly established by mining the patterns of signal values for a long vector sequence and then validating the potential invariants candidates.

Invariants can be single-node, 2-node or multi-node relations. While single-node and 2-node invariants can be similar to the non-trivial implications in the circuit, multi-node invariants are not covered by implications in the circuit and can provide important information about the circuit for many applications.

General strategy to find the circuit invariants has the following steps:

1. Circuit is simulated for a large set of vectors.

2. A set of candidate signals is identified for targeted invariants that can be more useful for a given application. This step is performed generally for huge circuits, where considering relations among all signals in the circuit is time-consuming and expensive.

3. For different combinations of the selected signals, various patterns are mined from the simulation data. Either the patterns can be statically present for all the vectors or can be absent in the entire simulation. Relations exhibiting such patterns are marked as potential invariant candidates, and are validated in the next step.

4. The validation for potential invariants can be done by different methods.

i) In one method, one potential invariant can be added at a time to the circuit formula (conjunctive normal form) and can be marked as true if it violates the formula. Once established to be true, the invariant can be added to the formula permanently which can help in reducing the search space for proving the following potential invariants candidates. This method depends on the sequence of picking the potential invariant candidates for validation. Proving one invariant before another can help validating the second invariant which may be proved false in absence of the first invariant.

ii) Another popular technique that is used to validate the potential invariants in sequential circuits is Assume and Verify [35] [36]. The circuit under consideration is unrolled for 2 time frames. In the first time frame, all the potential invariant candidates are assumed to be true, and one potential invariant is picked at a time and added to the second time frame and validated. If an invariant is proved to be true it is permanently added to the circuit formula and further candidates are verified. If it is proved to be false, it is removed from the formula and all the invariants proved in presence of the assumption of the false invariant are verified again. This process is repeated until a fixed point is reached.

iii) In another method, Binary decision diagrams for the signals can be used to validate the invariants. If two signals a and b cannot be (1, 1) simultaneously, the product of BDDs of signals a and b would yield a zero BDD. In case if a, b and c cannot be (1, 1, 0) at the same time, Product of BDDs of a, b and -c would yield a zero BDD.

In past, a great deal of innovative work has been done in the field of finding invariants and utilizing them for different applications. Circuit invariants have significant application in

formal verification techniques. [37] presents a tool to automatically extract design properties from a circuit using dynamic analysis. [38] provides an efficient tool for identifying true invariants from a set of initial invariant candidates. [39] describes an adaptive method of mining 3 different types of invariants in the circuit namely, global invariants, target state related invariants, and observability-don't-care extended invariants. The application of invariants for state justification is elaborated. [40] also presents techniques for efficiently mining invariants and highlight their role in model-checking application. [41], [42] and [43] apply the invariants for sequential equivalence checking and also propose efficient methods and filtering strategies for finding the invariants. [44] uses state-encoding to find the invariants at the RTL level for induction-based property checking.

In this thesis some of the popular techniques on invariants generation are used and the utility of invariants in the state restoration application is investigated.

Chapter 3

Proposed Architecture - 1

Distributed Trace Buffer Architecture for Flexible Trace Configuration

This chapter describes the first proposed architecture in detail. The first section explains the proposed architecture. Section 3.2 and Section 3.3 elaborate on the Grouping Strategy and the Restoration Strategy used in this work. Experimental results are shown in the section 3.4 and the discussion on the results is presented. Section 3.5 concludes this chapter.

3.1 Proposed Architecture

In this section, details on the first proposed flexible trace buffer architecture are presented. Fig. 3.1 illustrates a high-level view of a circuit focusing on a group of four FFs. In the following subsection it is described, how the proposed architecture enhances the existing scan chain in the design. In addition, important features of the trace buffer architecture are elaborated. Further a short description of the scan-trace decoder in the design is presented and the idea of offloading network that can be employed with such trace buffer design is discussed.

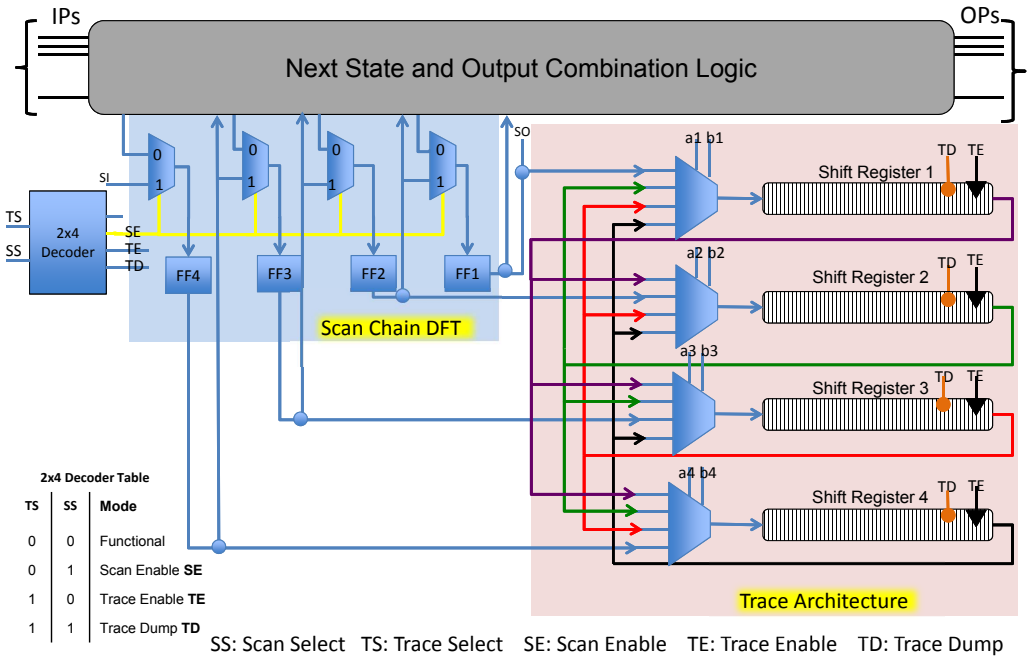


Figure 3.1: Proposed Architecture

3.1.1 Expanding on the Existing Scan Chain

In the left blue-shaded portion of Fig. 3.1, the four scan-FFs are still available for conventional scan. The pink-shaded part on the right is the trace architecture consisting of shift-registers that extend the existing scan flops. In contrast to a common on-chip buffer for a small set of trace signals, a dedicated buffer space is allotted for each scan FF in the circuit. In Fig. 3.1, each shift register i is associated with scan FF i. Since the size of each shift register is small and that area-efficient transmission-gate-based shift-registers are available [29], such an architecture is feasible. In an area-efficient shift-register, each cell consists of only 6 transistors, comparable to a memory bit in an SRAM-based buffer. In addition, the association of the shift-register to each FF will avoid the routing of all the traced signals to the common on-chip trace buffer, further reducing the area overhead of routing.

Sharing buffer space among the FFs in a group

As shown in Fig. 3.1, each shift register is preceded with a MUX to allow for sharing of the buffering space among the FFs in the group. In this example, four FFs are grouped together.

TE	TF1	TF2	TF3	TF4	a1	b1	a2	b2	a3	b3	a4	b4
0	0	0	0	0								
1	0	0	0	1	1	1	0	0	0	1	1	1
1	0	0	1	0	1	1	0	0	1	0	1	0
1	0	0	1	1	1	0	1	1	1	0	1	1
1	0	1	0	0	1	1	0	1	0	1	1	0
1	0	1	0	1	0	1	0	1	1	1	1	1
1	0	1	1	0	0	1	0	1	1	0	1	0
1	0	1	1	1	0	1	0	1	1	0	1	1
1	1	0	0	0	0	0	0	0	0	1	1	0
1	1	0	0	1	0	0	0	0	1	1	1	1
1	1	0	1	0	0	0	0	0	1	0	1	0
1	1	0	1	1	0	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	1	0	0	0	1	0	0	1	1
1	1	1	1	0	0	0	0	1	1	0	0	0
1	1	1	1	1	0	0	0	1	1	0	1	1

Figure 3.2: Truth Table to Determine the Select Lines for the Trace Architecture

However, the concept can be extended to larger groups, preferably of sizes that are powers of two, such as 4, 8 16 FFs, and so on. Each FF in Fig. 3.1 is associated with a 4×1 MUX. Based on which FFs are traced within a group, the shift registers can be configured to allow sharing among the traced FFs. For example, if only FF_1 is traced in a group, all the shift registers will be configured to store the traced values of FF_1. Thus, the number of cycles for which FF_1 can be traced would then be 4 times as the size of the shift register dedicated to FF_1.

The values to the select lines for the MUXs are determined by which FFs are traced in the group. Fig. 3.2 shows the truth table for determining the values for the select lines for each multiplexer. When the trace enable (TE) signal is set, signals TF_1, TF_2, TF_3 and TF_4 are used to determine which of the FF_1, FF_2, FF_3 and FF_4 are traced, respectively. If TF_i bit is 1, FF_i is selected to be traced. These bits can be stored in a separate configuration register of size n_{ffs} where n_{ffs} is the number of FFs in the circuit as shown in Fig. 3.4. This register can be populated on the fly by the diagnosis engineer, using the existing JTAG architecture or by provision of a small DFT design on chip. The ability of the diagnosis engineer to change the value of the configuration register during the test is highly useful.

In Fig. 3.2, consider row 2 of the truth table, the TE signal is high and TF_1, TF_2, TF_3 and TF_4 have values 0, 0, 0 and 1 respectively. This is the case in which only FF_4 is traced and is shown in Fig. 3.3a. Note the values of select lines for 4^{th} multiplexer namely (a_4,b_4) are (1,1). Thus, the fourth input to 4^{th} multiplexer which is FF_4 is fed in the shift register 4

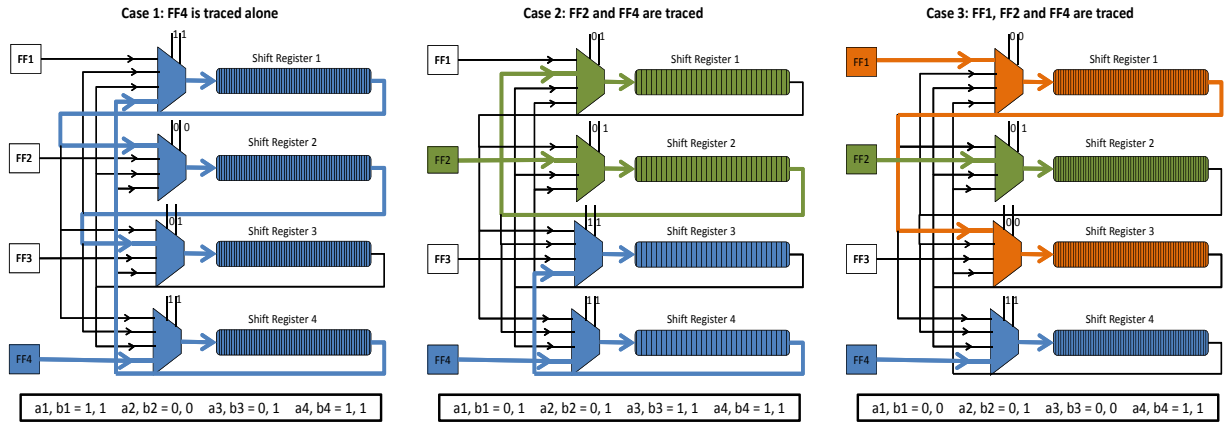


Figure 3.3: Buffer Sharing for Different Number of FFs Traced in a Group

when the trace enable is high. Now the select lines (a_1,b_1) are (1,1) passing the fourth input of multiplexer 1 to shift register 1. The fourth input to multiplexer 1 is connected to shift register 4. Thus, the values of shift register 4 are shifted to shift register 1 in each cycle and the new functional values of FF_4 are stored in shift register 4. Next, (a_2,b_2), the select lines of multiplexer 2 are (0,0), thus passing the values of shift register 1 to shift register 2 in case shift register 1 gets full and tracing is still on. Similarly (a_3,b_3) hold values (0,1) and connect the shift register 2 to shift register 3. Thus, FF_4 is traced for 4 times as many cycles as the size of shift register assigned to it.

Similarly, Fig. 3.3b shows the case when 2 FFs in a group are traced. In this example, FF_2 and FF_4 are being traced. The shift registers of untraced FFs namely FF_1 and FF_3 are being used by FFs FF_2 and FF_4 respectively. This can also be verified from row 6 in truth table in Fig. 3.2. Fig. 3.3c shows the case when 3 FFs in a group are traced. The shift register for untraced FF_3 is used by FF_1 which can be determined from row 14 in the truth table in Fig. 3.2. Depending on which FFs are traced the configuration can be easily obtained by reading from the truth table.

Thus, with the sharing of the shift-registers using minimal routing among the FFs in a group, one can trace a FF to a large number of cycles.

The proposed architecture employs a multiplexer with each FF in the circuit as against traditional interconnection networks where a sophisticated connection fabric is used for selecting the trace signals from comparatively smaller number of tap signals [18], [17]. As the

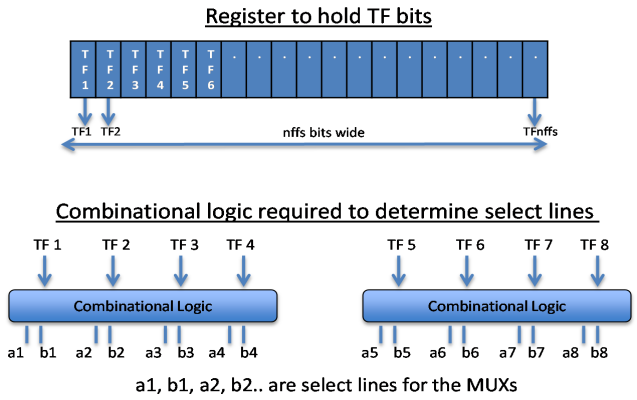


Figure 3.4: TF Bits and Select Lines Generation

FF-group size is increased, bigger MUXs are needed. This certainly gives an area overhead for this trace buffer design. However area-efficient transmission-gate based MUXs can be utilized. A transmission-gate based 2×1 MUX uses only 6 transistors [29]. Such designs can be extended to larger MUXs as well. Exploring additional area minimization techniques such as integrating the existing MUXs inside the scan FF and the trace buffer MUXs can be a potential future work.

Flexible and Configurable Tracing

With all the FFs in the circuit having a dedicated buffer space, the diagnosis engineer can configure the set of FFs to be traced according to the debugging needs. In contrast to the conventional architecture where either a small number of trace signals are hard wired to the trace buffer on chip or a semi-configurable organization via a complex routing network is required to select the desired trace signals from a given set of tapped signals [17], [18], our new architecture not only provides the flexibility of tracing any number of signals, but also provides a configurable selection of the trace signals without a huge routing overhead. An added advantage of the flexible architecture is that, by configuring the trace signals dynamically for different tests, complete restorability can be achieved across various test-sets irrespective of 0-dominant or 1-dominant test-set.

Case 1: Tracing a large number of FFs (wide-and-shallow configuration) In Fig. 3.5, a specific case of tracing 80% of the FFs in the circuit s9234 is shown. The size of the

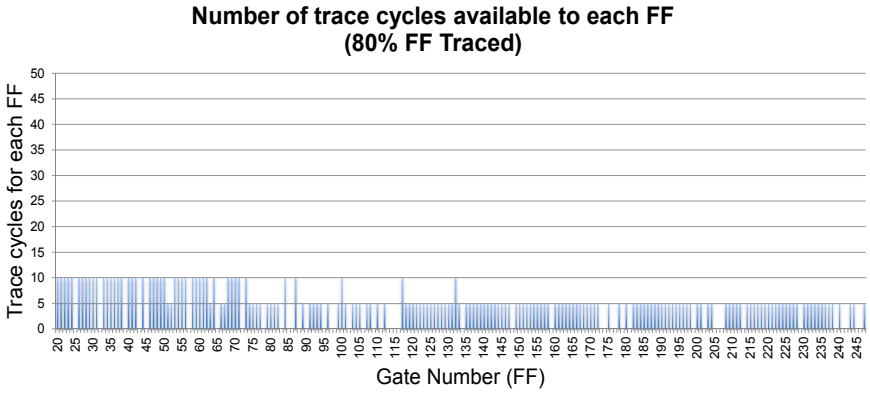


Figure 3.5: Tracing 80% FFs for s9234. Shift Register of Size 5 for each FF

shift-registers in the buffer is set to 5 and the group size is set to 16. The x-axis denotes the gate number of the FFs, and the y-axis shows the tracing depth. Note that the FFs that are not traced get 0 trace cycles and their shift registers are utilized by other FFs. Because 80% of the FFs are traced, sharing will be limited.

As expected, the number of trace cycles available to each FF with the sharing of shift registers will most likely be either 5 or 10. In a different 80% trace configuration, it might be possible that the trace depth be 15, 20, or larger for some FFs. We have not considered any trace selection strategy in this example and 80% of the FFs are selected randomly from the circuit.

Case 2: Tracing small number of FFs (narrow-and-deep configuration) In Fig. 3.6, a specific case of tracing 10% of the FFs in the circuit s9234 is shown. Again, group size of 16 and shift-register of size 5 are used. Note that the number of cycles available to certain FFs are now as high as 80 when only one FF in the group is traced. Note that there are 2 FFs in Fig. 3.6, which get 75 cycles each. This is because the the circuit has a total of 228 FFs, with 3 groups of size 16 and 12 groups of size 15. The grouping strategy used in this work is discussed in Section 3.2. If we increase the shift-register size to 10, we can get up to 160 trace cycles. Tracing deeper can help in restoration of deeper states and locating the cause of error faster.

Case 3: Tracing all the FFs (Multi-Cycle Scan) If all the FFs in the given architecture are traced, each FF will utilize the shift register associated with it. This is the multi-cycle scan configuration and can be very advantageous in scenarios as presented in [7], where successive scan dumps can now be replaced with a continuous trace in the suspect win-

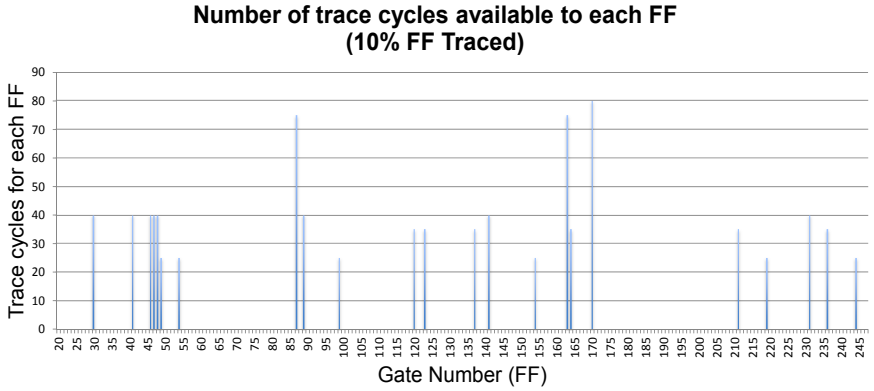


Figure 3.6: Tracing 10% FFs for s9234. Shift Register of Size 5 for each FF

dow. With the proposed architecture, we apply the test without intermittent scan-dumps, which can potentially reduce the debug time.

3.1.2 The 2×4 Decoder

The 2×4 decoder as shown in the architecture (Fig. 3.1) is used to select between various modes. The inputs to the decoder are Scan Select (SS) and Trace Select (TS) signals. If both SS and TS signals are low, the circuit works in the normal functional mode. If the Scan Select (SS) is high, then the scan enable signal from the decoder is 1 and the circuit works in the scan mode where the scan input is shifted in to the FFs for the testing purpose. If TS is high and SS is low, the trace mode is activated where the real time functional values of the FFs are stored in the dedicated buffer space for the FF. As explained in the previous subsection, the trace buffer size available to each FF is determined by which FFs are traced in a group. Finally, when both TS and SS are high, the trace dump mode is activated, where all the values from the trace buffer are offloaded using a data bus and an observable register. Finally, the offloaded values are stored separately and analyzed for debug. The network for offloading the trace buffer values is not shown in the figure due to lack of space but ideas of effective offloading are discussed in the section below.

3.1.3 The offloading network

Different designs for offloading network can be used to read the values from the trace buffer as proposed in the thesis.

Serial Offloading

One alternative can be taking the data out serially from the shift registers in a group. As all the shift registers in a group are connected, when we make the trace dump signal high, we can push the data out from all the shift registers serially while forcing the appropriate values on the select lines of the multiplexers before every shift register. The configuration should be such that, shift register 1 shifts the data to shift register 2; and shift register 2 is shifting the data forward to shift register 3 and likewise to 4. The data can be taken out from the offloading pin of shift register 4.

The various groups can be offloaded in parallel, while the serial data is offloaded from the shift registers of each group.

Parallel Offloading

Another alternative is parallel offloading of all the shift registers in a group. We can offload all the i^{th} bits of all the shift registers in a group together in parallel on a bus. The different groups can be offloaded one after another serially. However, in this approach, keeping the track of which offloaded data belongs to which FF, can be more complex than the serial offloading method.

3.2 Grouping Strategy

For the grouping strategy used in this thesis, the main idea was to distribute the traced FFs evenly across the groups. To maximize buffer sharing, one would like to group the FFs such that those likely traced signals in the narrow-and-deep configuration do not all fall into the same group. In our case, we first sort the FFs in decreasing order of their unchecked implications and then start distributing the sorted FFs in different groups and once the first

position in all the groups is occupied, we start filling the second position in the groups using the FFs with further decreasing counts of unchecked implications. For example, we consider the specific case of distributing the high implications FFs for s5378. Initially the group size is decided as 16. As number of FFs in s5378 are 179, the number of groups is set to $\left\lceil \frac{179}{16} \right\rceil = 12$. Now we start filling out first position of each group by the FFs in decreasing order based on the number of their unchecked implications. When all 179 FFs are distributed in this way, 11 groups have 15 FFs and the 12^{th} group has 14 FFs.

As stated before, the above strategy is used as we want to distribute the traced FFs (in our case, the FFs in decreasing order of number of unchecked implications) in as many different groups as possible. However, as the proposed traced architecture allows tracing of FFs based on any heuristic, the initial grouping of FFs may be of little value during the debugging. For practical design purposes, we recommend that grouping of FFs based on their proximity on chip as such, would reduce the routing overhead the most.

3.3 Restoration Strategy

In this work, signal restoration is conducted in a standard way, in which the circuit is first converted to a Boolean formula, such as the conjunctive normal form (CNF), and restoration can be readily obtained via Boolean propagation of the traced signal values. Because the implication relations are already embedded in the CNF formula, the restoration becomes straight-forward. If the depth of the trace buffer is d, the circuit is unrolled for d time-frames (termed as *Unrolling Depth*) and is converted to the corresponding Conjunctive Normal Form. The CNF is updated with the values of the selected traced signals in the trace cycles (as stored in the traced buffer) and Boolean Constraint Propagation (BCP) is performed recursively until no new signals are restored.

Note that, with the sharing of the buffer space in the proposed architecture, different trace signals can be traced for different number of trace cycles. In this sense, we unroll and create the CNF of the circuit for d_{max} cycles. Where d_{max} is the maximum trace depth that is available to any signal among all the selected trace signals. Although the unrolled depth can be high, we are interested in the number of signals restored in the final k cycles, where k is specified by the diagnosis engineer.

Table 3.1: Results for Tracing Various Percentages of FFs: ISCAS'89 Benchmark Circuits

Circuit	#FFs	% FFs Traced	Group of 4		Group of 8		Group of 16	
			AvgTC	#Unrestored	AvgTC	#Unrestored	AvgTC	#Unrestored
s5378	179	80	6.26	0	6.08	0	6.26	0
		60	8.36	0	8.36	0	8.36	0
		40	12.43	0	12.43	0	12.43	0
		20	20.00	0	24.86	0	24.86	0
s9234	228	80	6.26	0	6.15	0	6.26	0
		60	8.32	630	8.32	630	8.32	630
		40	12.53	3079	12.53	3079	12.53	3079
		20	20.00	3501	24.78	3501	24.78	3501
s13207	669	80	6.25	182	6.22	19	6.25	19
		60	8.34	359	8.34	296	8.34	296
		40	12.48	1040	12.48	1525	12.48	1150
		20	20.00	7446	24.96	7011	24.96	7011
s15850	597	80	6.24	29	6.21	0	6.24	0
		60	8.34	595	8.34	1091	8.34	595
		40	12.49	3399	12.49	3428	12.49	3428
		20	20.00	5160	25.08	5150	25.08	5150
s35932	1728	80	6.25	0	6.25	0	6.25	0
		60	8.33	0	8.33	0	8.33	0
		40	12.50	0	12.50	0	12.50	0
		20	20.00	0	24.97	0	24.97	0
s38417	1636	80	6.25	109	6.23	0	6.25	0
		60	8.33	13404	8.33	13404	8.33	13404
		40	12.51	31952	12.51	31952	12.51	31952
		20	20.00	42866	25.02	42866	25.02	42866

AvgTC: Average Traced Cycles

3.4 Experimental Results

We perform three sets of experiments. The first is on restoring 100% of the signals in the final k cycles; the second is on tracing a smaller set of signals deeper using the same architecture; finally, the third experiment is on studying the effect of multiple runs with various sets of traced signals. All the experiments are performed on the larger ISCAS'89 benchmarks and some open-core circuits. For all the experiments, the trace signal selection is based on the highest number of unchecked implications of a FF. Thus, a FF with more number of unchecked implications is picked first for tracing.

In the first experiment, in order to achieve 100% signal restoration in the last k cycles, we used group configurations of 4 FFs, 8 FFs and 16 FFs to restore the internal signals. We trace different percentage of FFs in the circuit and report the number of unrestored signals

Table 3.2: Results for Tracing Various Percentages of FFs: Open Core Circuits

Circuit	#FFs	% FFs Traced	Group of 4		Group of 8		Group of 16	
			AvgTC	#Unrestored	AvgTC	#Unrestored	AvgTC	#Unrestored
aes_core	530	80	6.25	38	6.18	32	6.25	44
		60	8.33	0	8.33	0	8.33	0
		40	12.50	0	12.50	0	12.50	0
		20	20.00	0	25.00	0	25.00	0
des_area	64	80	6.27	0	6.27	0	6.27	0
		60	8.42	0	8.42	0	8.42	0
		40	12.31	0	12.31	0	12.31	0
		20	20.00	2989	24.62	0	24.62	0
systemcaes	670	80	6.25	0	6.23	0	6.25	0
		60	8.33	0	8.33	0	8.33	0
		40	12.50	0	12.50	0	12.50	0
		20	20.00	0	25.00	0	25.00	0
wb_conmax	770	80	6.25	0	6.20	0	6.25	0
		60	8.33	0	8.33	0	8.33	0
		40	12.50	0	12.50	0	12.50	0
		20	20.00	0	25.00	0	25.00	0
wb_dma	523	80	6.26	0	6.20	0	6.26	0
		60	8.33	825	8.33	825	8.33	825
		40	12.51	2345	12.51	2345	12.51	2345
		20	20.00	2345	24.90	2345	24.90	2345

AvgTC: Average Traced Cycles

in the last 5 cycles. Each FF has a shift register of size 5 associated with it. Sharing of shift registers take place internally based on how many FFs are traced in one group of FFs. We observe that for different circuits, different percentage of FFs are required to be traced for achieving 100% restoration in the last 5 cycles. Table 3.2 reports the results. Columns 1, 2 and 3 in Table 3.2 give the circuit name, number of FFs in the circuit, and the different percentage of FFs traced. For each grouping configuration, the average trace cycles (AvgTC) available for the FFs and number of unrestored signals in the last 5 cycles (#Unrestored) are reported. Note that in the new architecture, due to the sharing of shift registers, each different traced FF can get a different number of trace cycles. The Average Trace Cycles for a given configuration is calculated as

$$AvgTC = \frac{\sum_{i=1}^{n} TC_i}{n}$$

where TC_i is the number of trace cycles available to the traced FF_i and n is number of

traced FFs. In all the cases the primary inputs are also traceable, but it is assumed that the values of primary inputs is known in the experiments. As expected, AvgTC increases with smaller percentage of traced FFs.

For some circuits, all signals are restored by tracing just 20% of the FFs as illustrated by a 0 in the #Unrestored column for those cases. For example, in circuit s5378, irrespective of the group size, all signals can be restored with just 20% of the FFs traced. One special case is des_area where tracing 20% FFs for group size of 4 does not restore everything. However, group sizes of 8 and 16 restore everything. This is most likely due to the larger values of AvgTC available in the larger group configurations.

Another interesting case is aes_core, where tracing 60% of the FFs restores everything, but if we trace 80% FFs in this case, some signals become unrestored. This can be accounted to the fact that if we trace 60% FFs, most of the traced FFs can be traced to a larger number of cycles than the case of tracing 80% FFs, thus helping the restoration.

Generally, in order to completely restore all signals in the last k cycles using a small set of trace signals, one would need to trace for more than k cycles. For this particular case, the 100% restoration is affected by the trade-off between trace-width and trace-depth (average trace depth of only 6.25 cycles) and a fine balance needs to be maintained for 100% restoration.

For all the remaining circuits such as s9234, s15850, s38417, wb_dma, tracing more than 60% FFs will restore everything in the last 5 cycles.

With the above results, it is clear that most of the circuits require a wider trace buffer configuration for 100% restoration. Moreover, in some of the cases that we discussed earlier, a flexible and configurable architecture is very critical and a restricted tracing configuration can lead to unrestored signals in the most critical cycles for debug.

The second set of experiments aims to compare our results with the conventional narrow-and-deep trace buffers, however with a 10× deeper tracing capability. In both conventional and our platforms, 20% FFs were traced. As we are just tracing 20% FFs, for a given trace buffer size, the conventional architecture with the same buffer capacity will already get 5 times as many cycles for each traced FFs as compared to the proposed architecture. Now for 10 times the buffer capacity, conventional architecture will get 50 times as many cycles. As in conventional trace buffers, the buffering is only allocated for the signals traced the number of trace cycles available is higher. Consider the case where there are 100 FFs in a circuit and

Table 3.3: Comparison with Conventional Trace Buffer with 10×Capacity. Group of 4

Circuit	#FFs	TB Size	% FFs Traced	Conventional (10xSize)			PA - Group of 4		
				AvgTC	#Unres	#Xtr	AvgTC	#Unres	#Xtr
s5378	179	895	20	250	0	53.7K	20.00	0	8.2K
s9234	228	1140	20	250	3501	68.4K	20.00	3501	10.4K
s13207	669	3345	20	250	6951	200.7K	20.00	7446	30.7K
s15850	597	2985	20	250	5150	179.1K	20.00	5160	27.4K
s35932	1728	8640	20	250	0	518.4K	20.00	0	79.4K
s38417	1636	8180	20	250	37742	490.8K	20.00	42866	75.2K
aes_core	530	2650	20	250	0	159K	20.00	0	24.3K
des_area	64	320	20	250	0	19.2K	20.00	2989	2.9K
systemcaes	670	3350	20	250	0	201K	20.00	0	30.8K
wb_conmax	770	3850	20	250	0	231K	20.00	0	35.4K
wb_dma	523	2615	20	250	2345	156.9K	20.00	2345	24K

PA: Proposed Architecture AvgTC: Average Traced Cycles
#Xtr: Number of transistors

Table 3.4: Comparison with Conventional Trace Buffer with 10×Capacity. Group of 8

Circuit	#FFs	TB Size	% FFs Traced	Conventional (10xSize)			PA - Group of 8		
				AvgTC	#Unres	#Xtr	AvgTC	#Unres	#Xtr
s5378	179	895	20	250	0	53.7K	24.86	0	11.4K
s9234	228	1140	20	250	3501	68.4K	24.78	3501	14.5K
s13207	669	3345	20	250	6951	200.7K	24.96	7011	42.8K
s15850	597	2985	20	250	5150	179.1K	25.08	5150	38.2K
s35932	1728	8640	20	250	0	518.4K	24.97	0	110.5K
s38417	1636	8180	20	250	37742	490.8K	25.02	42866	104.7K
aes_core	530	2650	20	250	0	159K	25.00	0	33.9K
des_area	64	320	20	250	0	19.2K	24.62	0	4K
systemcaes	670	3350	20	250	0	201K	25.00	0	42.8K
wb_conmax	770	3850	20	250	0	231K	25.00	0	49.2K
wb_dma	523	2615	20	250	2345	156.9K	24.90	2345	33.4K

PA: Proposed Architecture AvgTC: Average Traced Cycles
#Xtr: Number of transistors

20 FFs are traced. According to the proposed architecture, if each FF has a shift register of size 5 associated with it, the total trace buffer capacity is 500. Conventional architecture divides this buffer size of 500 among only 20 traced FFs and thus each traced FF gets 25 tracing cycles. If we allocate a trace buffer of size 5000 to conventional architecture scenario,

Table 3.5: Comparison with Conventional Trace Buffer with 10×Capacity. Group of 16

Circuit	#FFs	TB Size	% FFs Traced	Conventional (10xSize)			PA - Group of 16		
				AvgTC	#Unres	#Xtr	AvgTC	#Unres	#Xtr
s5378	179	895	20	250	0	53.7K	24.86	0	17.5K
s9234	228	1140	20	250	3501	68.4K	24.78	3501	22.3K
s13207	669	3345	20	250	6951	200.7K	24.96	7011	65.5K
s15850	597	2985	20	250	5150	179.1K	25.08	5150	58.5K
s35932	1728	8640	20	250	0	518.4K	24.97	0	169.3K
s38417	1636	8180	20	250	37742	490.8K	25.02	42866	160.3K
aes_core	530	2650	20	250	0	159K	25.00	0	51.9K
des_area	64	320	20	250	0	19.2K	24.62	0	6K
systemcaes	670	3350	20	250	0	201K	25.00	0	65.6K
wb_conmax	770	3850	20	250	0	231K	25.00	0	75.4K
wb_dma	523	2615	20	250	2345	156.9K	24.90	2345	51.2K

PA: Proposed Architecture AvgTC: Average Traced Cycles
#Xtr: Number of transistors

each traced FF will get $25 \times 10 = 250$ cycles. However, with the provision of sharing of FFs in the proposed architecture, we can obtain higher trace cycles for some of the FFs and we want to study if we can obtain comparable restoration to the conventional architecture.

The results are presented in the Table 3.3, 3.4 and 3.5 for the new architecture group sizes of 4, 8 and 16 respectively. Fig. 3.7 shows the bar graph for the number of unrestored signals in the result tables. As can be seen in Fig. 3.7, the restoration results of the proposed architecture are comparable to the conventional architecture, even when conventional architecture uses 10 times larger trace buffer. In the cases where the conventional architecture restores everything owing its ability to trace 10× deeper, the proposed architecture also restores everything by the advantage obtained by sharing the shift registers. In other cases, the restoration is comparable for both architectures.

We note that tracing deeper usually does not help in restoring more signals in the last 5 cycles. For example, in s15850, the conventional trace buffer was not able to restore 5150 signals in the last 5-cycle window. In our approach, we also were not able to restore 5150 signals, except for the group size of 4, in which 10 more signals were unrestored. Similar results for wb_dma. For s38417, the conventional trace buffer with deep tracing was able to restore more in the final 5 cycles. But even in s38417, the reduction was limited. In general, the restoration plateaus after a certain tracing depth. In order to restore more signals in the

final 5 cycles for such circuits, wider tracing is preferred over deeper tracing.

Tables 3.3, 3.4 and 3.5 also report the number of transistors that would be needed by the conventional and proposed trace buffer architectures without considering the routing network. For the proposed architecture, transistor count for MUXs used in groups of 4, 8 and 16 is also added. The conventional trace buffer architecture, is allocated 10 times the trace buffer capacity and so the number of transistors are higher in the conventional architecture as compared to the proposed architecture (shift registers + MUXs). Transistor count for for one cell of shift register is 6 (30 for the entire shift-reg). The transistor counts for 4x1MUX, 8x1MUX and 16x1 MUX are 16, 34 and 68 respectively. As clearly evident, even with a lower area overhead, the proposed architecture gives comparable restoration with the conventional narrow-and-deep architecture. With this experiment it can be inferred that our architecture also performs well in the case of narrow-and-deep trace configuration. This is only possible by the sharing of buffer space among various FFs in a group which makes the proposed architecture highly attractive.

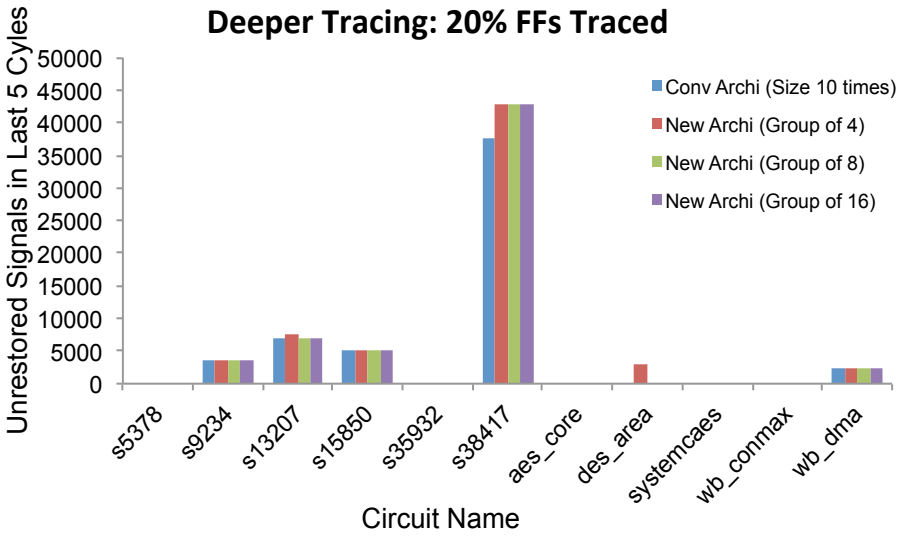


Figure 3.7: Tracing 20% FFs: Conventional Vs Proposed Architecture

A third experiment was performed in which we execute multiple debug runs with different sets of 20% traced FFs for each run, with the aim of restoring more values in a bigger window (more than the last 5 cycles). In each debug run, the already traced values from the preceding debug runs are known and can be used to help restore more values in subsequent runs. We wanted to explore how many runs of 20% tracing are needed to restore everything

in the last k cycles, where k is set of 10 and 20. In each run if we trace 20% FFs, the restored values will accumulate in each run. So, in the 2^{nd} run it is as if we are tracing 40% FFs, but at the same tracing depth as a 20% trace! We also show the comparison of the multiple run scenario with wide-and-shallow configuration with same set of FFs traced. That is, for example, we compare the 2 cumulative runs of 20% tracing with a single run of 40% wide-and-shallow tracing with same set of FFs traced.

As before, we select the trace signals based on the highest number of unchecked implications. In the first run we select top 20% FFs with highest unchecked implications; in second run we select the next 20%, and so on. Table 3.6 report the results for this experiment. Columns 1 and 2 give the circuit name and the debug run number. Column 3 shows the corresponding percentage of FFs traced for one run of wide-and-shallow configuration. The results for unrestored signals in the last 10 and 20 cycles are presented. For each case, the different group sizes of 4, 8 and 16 are used for experiment. The column "WnS" gives the number of unrestored signals in the last k cycles for the wide-and-shallow configuration. The column "Cumu" gives the number of unrestored signals in the last k cycles for the cumulative runs of 20% trace. The results for the cumulative runs of 20% FFs show that for most of the cases we are able to restore everything in the last 10 cycles after 4 debug runs. For some cases near 100% signals are restored in 3 runs itself.

Note that many signals before the last 10 cycles were also restored, just not reported here. Consider s9234 with group of 16. 7088 signals were unrestored in the last 10 cycles, but 16932 signals were unrestored when the window size increased to 20. This means that more signals (9844=16932-7088) were unrestored in the earlier 10 cycles of the 20-cycle window. This is intuitive as restoration follows in the fashion of an expansion cone - more signals restored will beget more down the road. However, this expansion plateaus to a steady state, depending on the number of FFs traced.

In s15850, everything is restored in the last 10 and 20 cycles after 3 debug runs, except for the case of unrestored signals in the last 20 cycles for a group configuration of 4 FFs.

As expected, for one complete run of the WnS configuration, the number of unrestored signals is higher (worse) than the cumulative runs scenario. This is because more trace cycles are available in individual runs of 20% FFs (more room for sharing) than for one run of 40 or 60% FFs traced. The difference in the number of unrestored signals widens as the value k is increased. For example, in circuit s9234, the total number of unrestored signals for

Table 3.6: Tracing Different Sets of 20% FFs for Multiple Runs

Circuit	Run	% FFs Traced in WnS	#Unrestored in last 10 cycles						#Unrestored in last 20 cycles					
			Group of 4		Group of 8		Group of 16		Group of 4		Group of 8		Group of 16	
			WnS	Cumu	WnS	Cumu	WnS	Cumu	WnS	Cumu	WnS	Cumu	WnS	Cumu
s9234	1	20	7088	7088	7088	7088	7088	7088	21281	21281	17353	17353	16932	16932
	2	40	6471	6161	6161	6161	6161	6161	35021	15149	43211	12876	42950	12846
	3	60	3434	410	3079	410	3033	410	62094	2558	61739	1260	61693	1260
	4	80	2814	0	2913	0	3499	0	61474	929	61573	0	62159	0
s13207	1	20	14679	14679	13840	13840	13840	13840	39174	39174	31230	31230	30038	30038
	2	40	3385	2160	4274	2070	3285	2070	40276	11318	61497	4562	57266	4340
	3	60	5118	550	4457	550	4221	550	92838	3064	92177	1154	91941	1100
	4	80	11035	0	6855	0	6524	0	98755	713	94575	0	94244	0
s15850	1	20	10328	10328	10210	10210	10210	10210	25863	25683	20292	20292	20543	20543
	2	40	6841	6608	7047	6490	6532	6490	27481	16274	72892	12852	68175	13103
	3	60	3128	0	4111	0	3242	0	107828	2149	108811	0	107942	0
	4	80	8329	0	3091	0	4376	0	113029	966	107791	0	109076	0
s38417	1	20	85760	85760	85760	85760	85760	85760	181730	181730	171138	171138	171138	171138
	2	40	63791	63785	64204	63785	63791	63785	167443	135856	230818	127044	226841	127044
	3	60	41836	29490	41314	29490	41310	29490	281326	66039	280804	58360	280800	58360
	4	80	31955	0	27425	0	25915	0	271445	5609	266915	0	265405	0
aes_core	1	20	0	0	0	0	0	0	21387	21387	10291	10291	0	0
	2	40	142	0	0	0	0	0	94833	14358	188596	7089	187995	0
	3	60	2262	0	18054	0	17546	0	338942	1167	354734	256	354226	0
	4	80	35602	0	35768	0	35340	0	372282	1077	372448	156	372020	0
wb_dma	1	20	4690	4690	4690	4690	4690	4690	12303	12303	9380	9380	9380	9380
	2	40	4690	4690	4690	4690	4690	4690	16277	11118	46246	9380	43851	9380
	3	60	3504	1650	3632	1650	3647	1650	73014	3951	73142	3300	73157	3300
	4	80	3668	0	2494	0	2602	0	73178	421	72004	0	72112	0

WnS: Wide-and-Shallow Configuration Cumu: Cumulative runs of the Narrow-and-Deep Config with different trace signals

the last 10 cycles under the 20% trace is initially the same for both the WnS and the Cumu configurations. However, as we apply successive runs of tracing different sets or 20%, the number of unrestored signals reduce in both cases. However, in the Cumu case, the reduction of the unrestored signals is faster. Again, this is due to the fact that the average number of trace cycles is higher when we trace fewer FFs, thereby allowing for more opportunity for restoration. When we attempt to increase k to 20, the number of unrestored signals actually increase in the WnS configuration, as tracing more FFs reduces the effective tracing depth.

The same trend can be seen in other circuits as well.

As shown in the third experiment, the proposed architecture can also facilitate multiple runs of tracing different sets of signals, useful to restore all internal signals in a larger window.

3.5 Conclusion

This chapter presents a novel, flexible trace buffer architecture to achieve 100% restoration of signals during post-silicon validation. The architecture provides the flexibility of selecting different sets of trace signals according to the debugging needs. With the sharing of shift registers within a group of FFs, this architecture can be configured to trace narrow-and-deep to help in locating the cause of error more quickly. Experimental results show that with the wide-and-shallow configuration, the proposed architecture is able to restore all the critical internal signals in the last k cycles. On the other hand, in deeper trace configuration, the restoration quality is not compromised and is comparable to the conventional trace architecture. Multiple runs of the narrow-and-deep configuration with varying sets of trace signals can also restore most of the signals. In the future, the observability of all FFs with the help of shift-registers gives a new "Multi-Cycle Scan" capability to the architecture which can be highly beneficial for testing and diagnosing manufacturing defects.

Chapter 4

Proposed Architecture - 2

Wide Multiplexed Tracing Architecture

4.1 Proposed Architecture

This section presents the details of the second proposed flexible trace buffer architecture. Fig. 4.1 illustrates a high-level view of the circuit focusing on a group of four FFs. The top blue shaded part is the scan chain in the circuit and the pink shaded part at the bottom shows the trace buffer configuration. The following sub-section explains the shared trace buffer configuration in this proposed architecture. The 2×4 Decoder in the figure is similar to the architecture as in the previous chapter and is not discussed here. The offloading network that can be used with this architecture is very similar to the previous architecture, the ideas are presented in the other sub-section.

4.1.1 Shared Traced Buffer Configuration

The idea of providing a small trace buffer in place with FFs is extended in this architecture as well. The difference as shown in the figure is that, in this architecture, the four FFs in a group share a single shift register. A 4×1 multiplexer is used to determine which one of the FFs gets the shift register to store the corresponding traced values. Similar to the previous architecture, the number of FFs traced in a group determines the number of trace

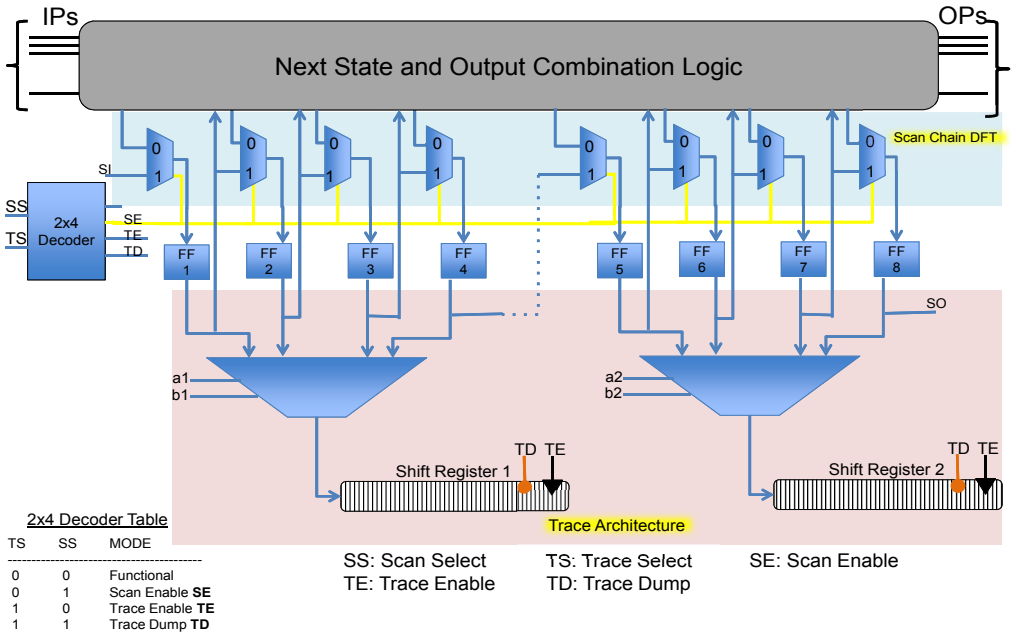


Figure 4.1: Proposed Architecture - 2

cycles available to the FFs. If only one FF is traced in the group, all bits in the shift register assigned to the group are used to store the traced values of the selected traced FF. In case more than one FF is traced in a group, the scenario is different as explained below.

Multiplexed Tracing

If more than one FF in a group of FFs is traced, the shift register is used to store the trace values of the FFs in the alternate cycles. For example, if a shift resister of size 20 is allocated to a group of 4 FFs, and all FFs in the group are traced, FFs are traced in alternate cycles, with each FF being traced, once in 4 cycles.

The multiplexer before the shift register is controlled using the select lines, which determine, which FF is traced for which cycle. For our results we have traced the FFs in alternate cycles, and followed the truth table given in the Figure 4.2.

When the TE is high, the TF bits in the table determine if the FF is selected for tracing. If TF1 is 1, this implies, FF1 is traced. As evident, this is the example for group of 4 FFs.

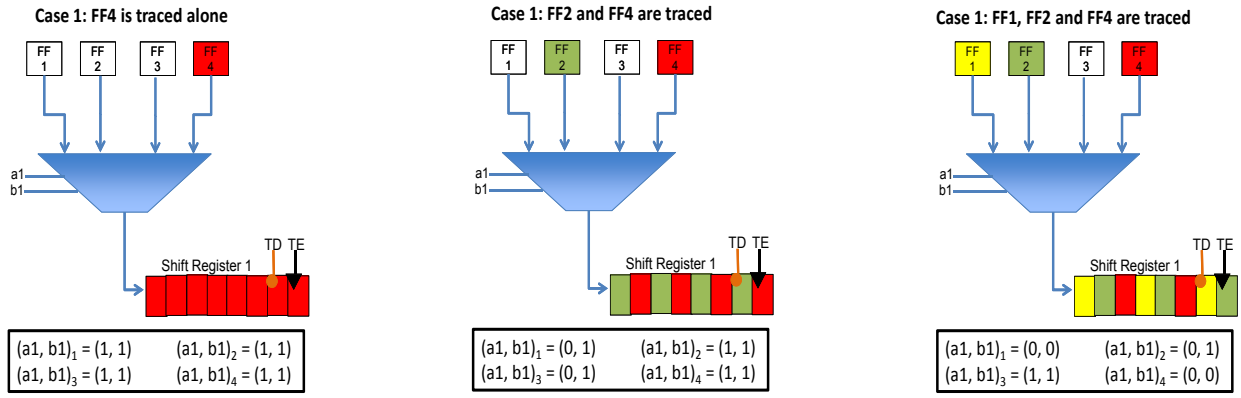
					Time Frame 1		Time Frame 2		Time Frame 3		Time Frame 4	
TE	TF1	TF2	TF3	TF4	a1	b1	a1	b1	a1	b1	a1	b1
0	0	0	0	0								
1	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	1	0	1	0	1	0	1	0	1	0
1	0	0	1	1	1	0	1	1	1	0	1	1
1	0	1	0	0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	0	0	1	1	0
1	0	1	1	1	0	1	1	0	1	1	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	1	1	0	0	1	1
1	1	0	1	0	0	0	1	0	0	0	1	0
1	1	0	1	1	0	0	1	0	1	1	0	0
1	1	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	1	0	0	0	1	1	1	0	0
1	1	1	1	0	0	0	0	1	1	0	0	0
1	1	1	1	1	0	0	0	1	1	0	1	1

Figure 4.2: Truth Table to Determine the Select Lines for Multiplexer

Consider row 2 of the truth table. The values of TF1, TF2, TF3 and TF4 are 0, 0, 0 and 1 respectively. This is the case of one FF (FF4) being traced in the group of four. The 8 right columns of the table give the values of select lines in four consecutive time frames. In the above case, as only one FF is selected, it is traced for all the 4 time frames. Thus the select lines in all time frames are (1, 1) for the multiplexer tracing the functional values of FF4 into the shift register. This case is also illustrated in Figure 4.3a with a shift register of size 8 (total 8 time frames).

The multiplexed tracing is performed when more than 1 FF in a group are traced. Say for example, TF1, TF2, TF3 and TF4 are 0, 1, 0, 1 respectively (row 6 of the truth table). This is the case where FF2 and FF4 are traced together. As illustrated in Figure 4.3b, the shift register is used to store the values of FF2 in time frames 1, 3, 5, and 7 and the values to FF4 in time frames 2, 4, 6, and 8, thus dividing the buffer space between the 2 FFs equally.

Consider the case of tracing 3 FFs in a group. In Figure 4.3c, case of tracing FF1, FF2 and FF4 is shown. This corresponds to the row 14 in the truth table. As clearly seen, the select lines a1, b1 in first time frame are (0, 0) tracing FF1, then followed by tracing FF2 in next time frame ((a1, b1) = (0, 1)), tracing FF4 in third time frame, and back to tracing FF1 in time frame 4. As the FF1 and FF2 are the earlier FFs in the group as compared to FF4, they get the extra cycles that cannot be equally divided into 3 traced FFs.



(a) (b) (c)

Figure 4.3: Buffer Sharing for Different Number of FFs Traced in a Group

Effective Width of Tracing

In this architecture, the FFs in a group share one single shift register, and thus the FFs in a group cannot be traced simultaneously. To divide the trace buffer in the group, multiplexed tracing is necessary. This means, even when we trace say 80% FFs in the circuit, in this architecture, the 80% FFs are not traced simultaneously. But while tracing the FFs in the alternate cycles, we can still achieve high restorations. Thus we introduce the concept of Effective Width of Tracing. Effective width can be defined as the percentage of FFs that when traced simultaneously match the restoration of a given percentage of FFs traced in the multiplexed manner in a given grouping configuration. In principle we investigate how much restoration loss we incur when we perform multiplexed tracing.

Grouping configuration highly affects the restoration in this architecture. Figures 4.4. 4.5 and 4.6 illustrate 3 different group sizes of 4, 8 and 16 FFs. The effective width of the different group sizes will depend on the percentage of FFs selected for tracing. The three figures depict the example of a trace buffer size of 32 distributed among different FF groups. In this example, there are 16 FFs in the circuit and 12 out 16 (75%) are traced. Let us consider the group of 4 FFs. In Figure 4.4, 16 FFs are divided in 4 groups with 4 FFs each. Let us assume FF1 to FF16 are in the decreasing order of unchecked implications.

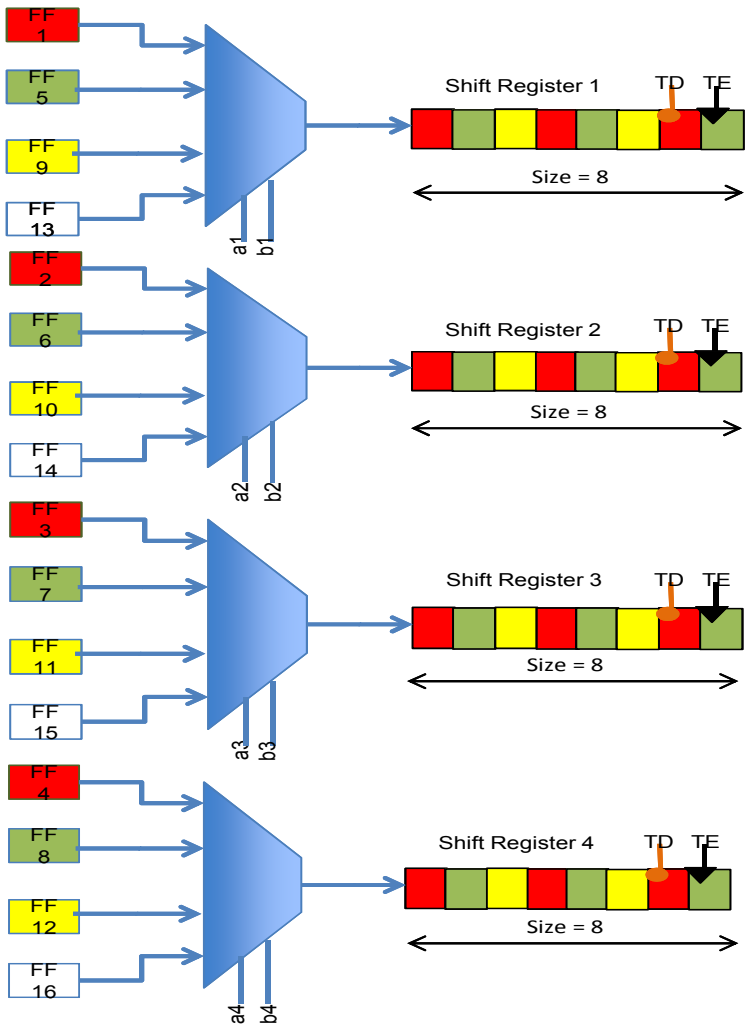


Figure 4.4: Multiplexed Tracing: Group of 4 FFs

The distribution of FFs among different groups is done such that high implication FFs fall in different groups. So the first group has FFs 1, 5, 9 and 13, the second group has FFs 2, 6, 10, 14 and similarly for groups three and four. As 12 FFs out of 16 are traced, each group has 3 traced FFs. As evident in the figure, overall 4 high implication FFs can be traced simultaneously in this configuration. Also, in one group, the frequency of tracing a FF is once in every 3 cycles. As a trace buffer of size 32 is divided into 4 groups, the maximum trace depth available for this configuration is 8.

Let us consider the case in Figure 4.5, where the case with group size of 8 is depicted.

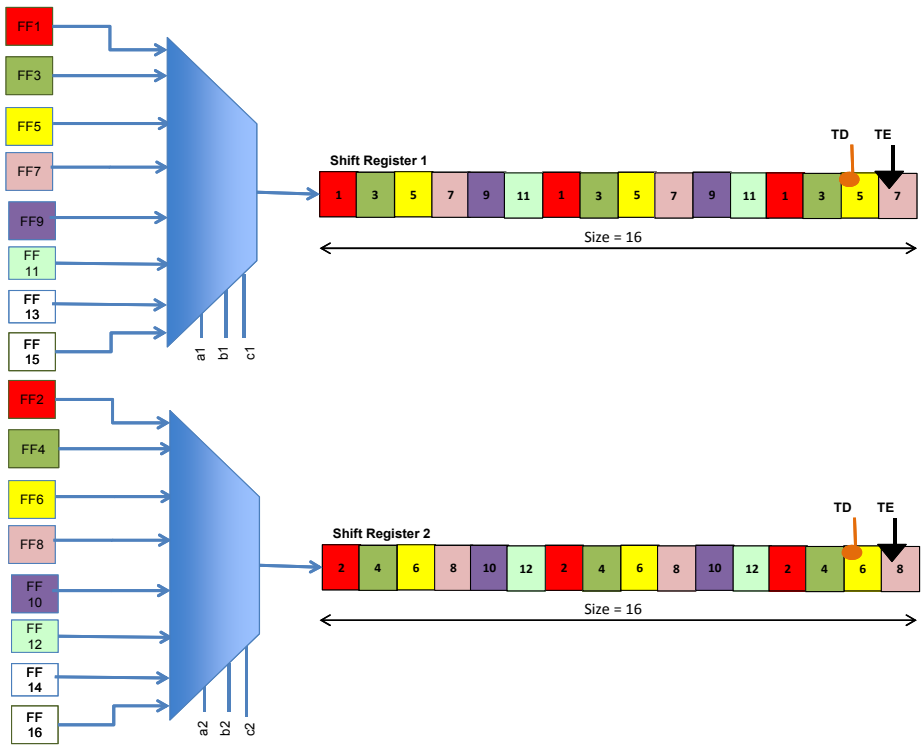


Figure 4.5: Multiplexed Tracing: Group of 8 FFs

There are 2 groups of 8 FFs each and the buffer size of 32 is divided to provide 16 bit shift register to each group. The distribution of FFs is again in accordance with allotment of high implication FFs to different groups. Again, as 12 FFs are selected for tracing, 6 FFs in each group are traced. As can be seen, 2 high implication FFs can be traced simultaneously. Also, in each group, the frequency of tracing a FF is once in 6 cycles. As the amount of FFs that can be traced simultaneously and the frequency of multiplexed tracing is less, it is intuitive, that the effective width of the tracing will be lower in this configuration. Although, a major advantage that comes with this configuration is that, the depth of tracing is higher. Each group has a shift register of size 16 and thus we can trace some FFs upto a depth of 16 (but alternating of course). We wish to study this tradeoff of tracing effective-wide and shallow and effective-narrow and deep.

Finally Figure 4.6 illustrates the group size of 16. It can be deduced that for this case, more than one FF cannot be traced simultaneously and the tracing frequency is very low, thus significantly reducing the effective width. But, the number of cycles traced overall are deeper

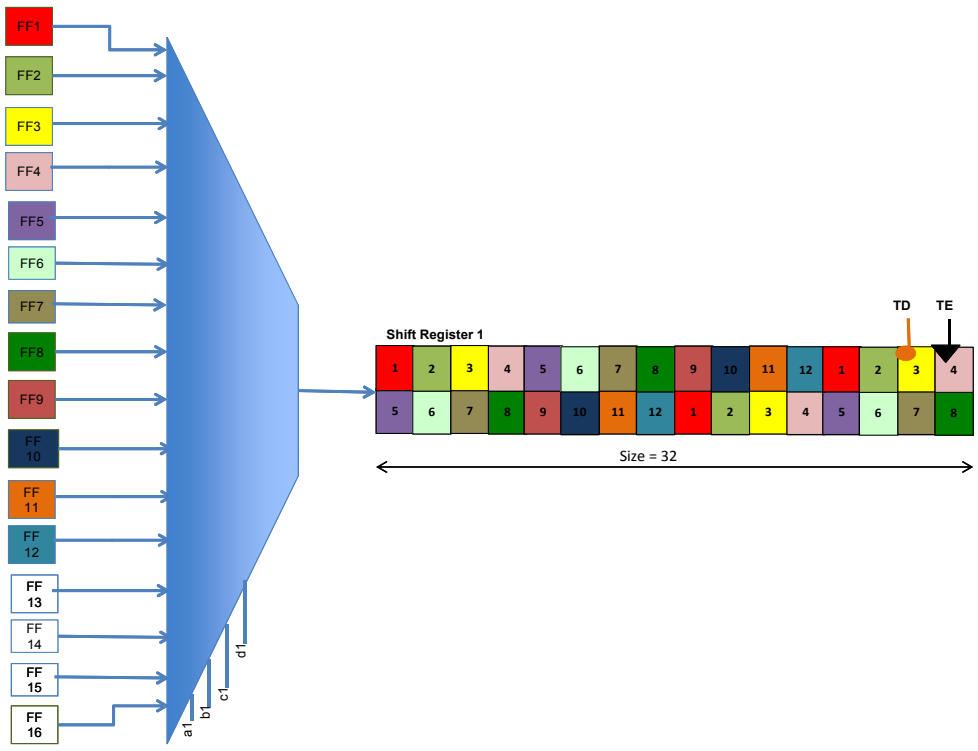


Figure 4.6: Multiplexed Tracing: Group of 16 FFs

and the results for unrestored signals in deeper cycles can get an advantage of the same.

Higher Tracing Depth Availability

As multiplexed tracing is performed in this architecture, for the same trace buffer size, the tracing depth available in wider tracing is significantly higher as compared to the first proposed architecture discussed in Chapter 3. For example, if in the first architecture, each FF has a shift register of size 5 associated with it, if all FFs are traced, the total depth of tracing is only 5. But in a second architecture, if a group of 4 FFs is used, a shift register of size 20 is available to the group. In this architecture, when 100% FFs are traced, the FFs will be traced alternately, upto 20 cycles. Thus the effective width will be lower than 100% in the second architecture, but the depth of tracing is higher.

When the group size is increased, the tracing depth availability becomes even higher, though reducing the effective-width of tracing.

4.1.2 The offloading network

The idea of parallel offloading can be used with this architecture. The shift registers of all groups can be offloaded in parallel, one bit at a time, i.e offloading all the i^{th} bits of all the shift registers in one cycle. After the bits are offloaded the traced values can be sorted for each FF and can be used for restoration.

4.2 Grouping and Restoration Strategy

The grouping strategy and restoration strategy used for this architecture are similar to that described for the first architecture in chapter 3.

A BCP implication engine is used to perform the restoration of signals after plugging the trace values in the CNF formula of the circuit.

Grouping of FFs is also based on distributing FFs with high number of unchecked implications in the different groups as discussed in previous proposed architecture. In [19] a method of grouping 2 FFs for multiplexed tracing is proposed which is based on the implication-based correlation between FFs. A similar method can be employed for better grouping strategy. But as the FFs will be grouped in place and connected to a shift register, grouping the FFs which are closer on the chip is suggested.

4.3 Investigating Tracing of Primary Inputs

In both the proposed architectures, it is assumed that all primary input (PI) values are available for the the maximum unrolling depth of the circuit (That is the max number of cycles upto which any FF can be traced). PIs play a significant role in the restoration thus knowing the values of PIs over a certain depth is very advantageous for restoration. In cases, when the circuit given for post-si validation is a part of a bigger design, the values of PIs are not readily available and have to be traced. The first architecture specifies that the all the PIs are traceable but the area overhead of tracing the PIs was not considered.

While performing the second architecture experiments, we also investigated, if there can be a subset of PIs that can be traced, and what happens if the PIs are traced in alternate

cycles. First experiment was performed where different percentage of FFs is traced for a given trace buffer size, for different grouping configurations in 1st and 2nd architecture. The total number of PIs restored in all the cycles with just tracing the FFs is noted. Could we just trace the PIs that are not restored by tracing the FFs alone? When we performed the experiments, we found that for different circuits the percentage of PIs restored by tracing FFs alone varies widely. Not only this, but different percentage of FFs traced restore different number of PIs in one circuit. In addition, a PI value is not restored in every cycle, and could not be very useful while considering overall restoration.

In the second experiment, we decided to trace all the PIs in the circuit for the max unrolling depth of the circuit on which the restoration is performed. But the PIs are grouped similar to grouping FFs in the second proposed architecture and a group of PIs is traced in alternate cycles based on the group size. Even with a group size of 2, where all the PIs are traced in every alternate cycle, the restoration results were negatively affected.

Thus we concluded that tracing of PIs in all cycles is very helpful and a shift register should be provided with each PI. The size of shift register will be the maximum tracing depth that can be available to any FF in the circuit, in first architecture, or the maximum tracing depth of any group of FFs in second architecture.

Tables 4.1 and 4.2 show the area overhead of tracing the PIs in the first proposed trace buffer architecture. The first 3 columns in the tables show the circuit name, number of FFs and the number of PIs in the circuit. For different group sizes, different maximum tracing depths are possible. If all the PIs are allotted, the shift register of the size of maximum tracing depth, the area overhead is determined. As it can be seen, for many benchmark circuits, the area overhead of tracing the PIs is not as much as compared to the trace buffer size of the FFs with multiplexer network. For example for circuits s13207 to s38584, the area overhead of tracing the PIs is no more than 10% in most of the cases. For s5378 and s9234 the overhead of adding the PIs is larger but still less than the amount of trace buffer size required by FFs. In addition, with both the trace buffers sizes combined (for FFs and PIs) the area required is still far less than required by the conventional trace buffer with 10x capacity with similar restoration performance. The overhead of tracing the PIs is added to conventional architecture case as well. For the explanation of comparison with the 10x buffer size of the conventional architecture, you can refer to the second experiment discussed in chapter 3.

For open core circuits, the scenario is different. As the number of PIs exceed the number

Table 4.1: Area Overhead of Tracing PIs: First Architecture (ISCAS'89)

Circuit	#FFs	#PIs	Shift Reg Size	Group Size	Max Unr Depth	TB Area FFs with MUXs	TB Area PIs	Percentage of FFs TB Size	Total Area	Conv. TBx10
s5378	179	35	5	4	20	8234	4200	51.00	12434	64200
				8	40	11456	8400	73.32	19856	64200
				16	80	17542	16800	95.77	34342	64200
s9234	228	19	5	4	20	10488	2280	21.74	12768	74100
				8	40	14592	4560	31.25	19152	74100
				16	80	22344	9120	40.81	31464	74100
s13207	669	31	5	4	20	30774	3720	12.088	34494	210000
				8	40	42816	7440	17.37	50256	210000
				16	80	65562	14880	22.69	80442	210000
s15850	597	14	5	4	20	27462	1680	6.11	29142	183300
				8	40	38208	3360	8.79	41568	183300
				16	80	58506	6720	11.48	65226	183300
s35932	1728	35	5	4	20	79488	4200	5.28	83688	528900
				8	40	110592	8400	7.59	118992	528900
				16	80	169344	16800	9.92	186144	528900
s38417	1636	28	5	4	20	75256	3360	4.46	78616	499200
				8	40	104704	6720	6.41	111424	499200
				16	80	160328	13440	8.38	173768	499200
s38584	1452	12	5	4	20	66792	1440	2.15	68232	439200
				8	40	92928	2880	3.09	95808	439200
				16	80	142296	5760	4.047	148056	439200

Table 4.2: Area Overhead of Tracing PIs: First Architecture (Open-Core)

Circuit	#FFs	#PIs	Shift Reg Size	Group Size	Max Unr Depth	TB Area FFs with MUXs	TB Area PIs	Percentage of FFs TB Size	Total Area	Conv. TBx10
aes_core	530	258	5	4	20	24380	30960	126.98	55340	236400
				8	40	33920	61920	182.54	95840	236400
				16	80	51940	123840	238.42	175780	236400
des_area	64	125	5	4	20	2944	15000	509.51	17944	56700
				8	40	4096	30000	732.42	34096	56700
				16	80	6272	60000	956.63	66272	56700
des3_area	128	239	5	4	20	5888	28680	487.09	34568	110100
				8	40	8192	57360	700.19	65552	110100
				16	80	12544	114720	914.54	127264	110100
systemcaes	670	259	5	4	20	30820	31080	100.84	61900	278700
				8	40	42880	62160	144.96	105040	278700
				16	80	65660	124320	189.33	189980	278700
wb_conmax	770	1129	5	4	20	35420	135480	382.49	170900	569700
				8	40	49280	270960	549.83	320240	569700
				16	80	75460	541920	718.15	617380	569700
wb_dma	523	215	5	4	20	24058	25800	107.24	49858	221400
				8	40	33472	51600	154.15	85072	221400
				16	80	49162	103200	209.91	152362	221400

Table 4.3: Area Overhead of Tracing PIs: Second Architecture (ISCAS'89)

Circuit	#FFs	#PIs	SR Size	Group Size	No. of Groups	Max Unr Depth	TB Area FFs with MUXs	TB Area PIs	Percentage of FFs TB Size	Total Area	Conv. TBx10
s5378	179	35	5	4	45	20	6090	4200	68.96	10290	64200
				8	23	40	6152	8400	136.54	14552	64200
				16	12	80	6186	16800	271.58	22986	64200
s9234	228	19	5	4	57	20	7752	2280	29.41	10032	74100
				8	29	40	7826	4560	58.26	12386	74100
				16	15	80	7860	9120	116.03	16980	74100
s13207	669	31	5	4	168	20	22758	3720	16.34	26478	210000
				8	84	40	22926	7440	32.45	30366	210000
				16	42	80	22926	14880	64.90	37806	210000
s15850	597	14	5	4	150	20	20310	1680	8.27	21990	183300
				8	75	40	20460	3360	16.42	23820	183300
				16	38	80	20494	6720	32.79	27214	183300
s35932	1728	35	5	4	432	20	58752	4200	7.14	62952	528900
				8	216	40	59184	8400	14.19	67584	528900
				16	108	80	59184	16800	28.38	75984	528900
s38417	1636	28	5	4	409	20	55624	3360	6.04	58984	499200
				8	205	40	56050	6720	11.98	62770	499200
				16	103	80	56084	13440	23.96	69524	499200
s38584	1452	12	5	4	363	20	49368	1440	2.91	50808	439200
				8	182	40	44920	2880	6.41	47800	439200
				16	91	80	49000	5760	11.75	54760	439200

of FFs by significant amount, the area required for tracing the PIs is higher than what is required for tracing the FFs but still the combined area is less than the conventional architecture with 10x capacity.

Tables 4.3 and 4.4 present the values of area overhead for FFs and PIs in the second trace buffer architecture. As the number of multiplexer required in the second architecture are considerably low, the area requirement for shared tracing of FFs is lesser and the trace buffer for PIs are larger percentage amounts of the FF trace buffers as compared to the first architecture. In certain cases the area requirement for tracing the PIs is higher than that required for tracing the FFs. But the overall area is far less than the conventional architecture with similar performance.

4.4 Experimental Results

For investigating, how the restoration is affected by multiplexed tracing using different group sizes in the second architecture, the following experiment was performed. For the ISCAS'89

Table 4.4: Area Overhead of Tracing PIs: Second Architecture (Open-Core)

Circuit	#FFs	#PIs	SR Size	Group Size	No. of Groups	Max Unr Depth	TB Area FFs with MUXs	TB Area PIs	Percentage of FFs TB Size	Total Area	Conv. TBx10
aes_core	530	258	5	4	133	20	18028	30960	171.73	48988	236400
				8	67	40	18178	61920	340.63	80098	236400
				16	34	80	18212	123840	679.99	142052	236400
des_area	64	125	5	4	16	20	2176	15000	689.33	17176	56700
				8	8	40	2192	30000	1368.61	32192	56700
				16	4	80	2192	60000	2737.22	62192	56700
des3_area	128	239	5	4	32	20	4352	28680	659.00	33032	110100
				8	16	40	4384	57360	1308.39	61744	110100
				16	8	80	4384	114720	2616.78	119104	110100
systemcaes	670	259	5	4	168	20	22788	31080	136.38	53868	278700
				8	84	40	22956	62160	270.77	85116	278700
				16	42	80	22956	124320	541.55	147276	278700
wb_conmax	770	1129	5	4	193	20	26188	135480	517.33	161668	569700
				8	97	40	26398	270960	1026.44	297358	569700
				16	49	80	26432	541920	2050.24	568352	569700
wb_dma	523	215	5	4	131	20	17786	25800	145.05	43586	221400
				8	66	40	17934	51600	287.72	69534	221400
				16	33	80	17802	103200	579.71	121002	221400

and Open-Core benchmark circuits, different percentage of FFs were traced in alternate cycles. The frequency of tracing for a selected FF depends on the group size and the number of FFs selected for tracing in the group. With the bigger group size, the overall depth of tracing increases but the frequency of tracing for a selected FF decreases. Also as discussed before, with larger groups, the effective width of tracing is low. In this experiment, every FF has a shift register of size 5 associated with it. For a group size of 4, the total tracing depth available to one group is 20, and if all the FFs in the group are traced, then each FF is traced once in 4 cycles. For this experiment, each PI is traced for 80 cycles, irrespective of the group size.

Tables 4.5 and 4.6 illustrate the results of the restoration for the above experiment. The tables list the number of unrestored signals in last 5 cycles, last 10 cycles and last 20 cycles for different cases. Table 4.5 presents the results for ISCAS'89 benchmark circuits. As can be observed in the table, the results vary for different circuits across different group sizes. Possible explanation can be given for certain result patterns. For ISCAS'89 benchmark circuits, for group size of 1 (i.e. no grouping of FFs), the performance is better than group sizes of 4, 8 and 16, for the number of unrestored signals in last 5 cycles, especially in the cases where around 40% of the FFs are traced. This is because, the wider tracing with a

depth of 5 available in group size 1, is good enough to restore almost all internal signals in last 5 cycles. The higher group sizes are not able to restore as much because the multiplexed tracing reduces the effective width and thus the performance is affected. However, difference in trend is seen when less than 40% of FFs are traced, and the higher group sizes restore more signals than the case of no grouping. A possible explanation to such result pattern can be that when tracing is narrow, the higher depth available for bigger group sizes provides the ability to restore other FFs and thus increase the restoration. As the depth of group size of 1 is just 5, the narrow tracing is not able to restore the other FFs and the restoration is lesser as compared to the bigger groups. Also, when the tracing of wider than 60% is performed, the multiplexed tracing in group of 4, 8 and 16 starts performing better because the effective width (though lower than the no-grouping case) is enough to restore most of the internal signals in last 5 cycles. For all the circuits, 100% restoration is achieved in certain configurations.

In the case of restorations in last 10 and 20 cycles, the larger group sizes perform better than no grouping, for most of the cases. The number of configurations that achieve 100% restoration in the larger group sizes is higher than that achieved by the lower group sizes (especially no grouping case). This clearly explains that though multiplexed tracing is performed, the higher depth availability in the larger groups, gives better restoration is deeper cycles (last 10 and 20 cycles). In case of s5378, when tracing is wider (more than 40%), the case of no grouping gives better restoration than the group sizes of 4, 8 and 16 for certain configurations. However, in some configurations for s5378, the higher group sizes outperform. In cases of all other circuits, higher group sizes give better restoration in last 10 and last 20 cycles for most of the configurations.

Table 4.6 presents the results for the open-core circuits. For most of the circuits, everything is restored in all group configurations and any percentage of FF traced. This can happen because in the experiment, PIs are traced upto 80 cycles and the PIs may be enough to restore all the signals in the circuit irrespective of the amount of FFs traced. Only the cases of aes_core and wb_dma have non-zero restoration values. In the case of wb_dma, for all the group sizes, the restoration is same. This can again be because of the maximum restoration being performed by the PIs. The case of aes_core is different. For the restoration in last 5 cycles, group size of 1 (no grouping) has better restoration than group sizes of 8 and 16. However, group size of 4 gives the best performance than all other grouping configurations and achieve 100% restoration in most of the cases. For the restoration in last 10 cycles,

Table 4.5: Comparison of Second Architecture Variants (ISCAS'89)

Circuit	% FFs	# Cycles	#Unres Signals in Last 5 cycles				#Unres Signals in Last 10 cycles				#Unres Signals in Last 20 cycles			
Name	Traced	PIs Traced	GOF1	GOF4	GOF8	GOF16	GOF1	GOF4	GOF8	GOF16	GOF1	GOF4	GOF8	GOF16
s5378	100	80	0	634	0	0	87	1259	0	0	274	2553	0	0
	90	80	0	488	0	152	87	958	0	310	274	1977	0	636
	80	80	0	0	357	446	87	0	729	914	274	40	1473	1843
	60	80	0	624	235	0	87	1247	484	0	274	2515	977	0
	40	80	0	454	144	483	87	897	302	981	274	1828	612	1977
	20	80	0	0	416	0	87	0	854	0	274	8	1726	0
	10	80	1194	1189	1189	1241	2502	2398	2398	2503	5180	4865	4860	5073
s9234	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	240	240	240	240	480	480	480	480	960	960	960	960
	40	80	3389	5515	8583	10159	14345	11729	16494	20053	39555	24114	32406	40745
	20	80	4319	3501	5248	8676	16056	7088	11247	17286	41746	15016	23178	34438
	10	80	7813	6050	6050	9404	24812	12105	12105	18542	59039	26003	24115	36702
s13207	100	80	0	0	0	0	8747	0	0	0	39576	460	0	0
	90	80	163	334	371	0	9130	669	729	57	40154	1698	1499	515
	80	80	163	0	0	0	9130	0	0	13	40154	408	13	239
	60	80	579	275	275	275	11286	550	550	550	42604	1586	1175	1100
	40	80	2817	1569	1603	1342	16004	3790	3274	2759	48031	13392	6594	5598
	20	80	9060	7011	13347	12690	27229	13838	27260	28088	66686	29377	54575	56681
	10	80	12385	10328	10328	16734	33735	20332	20332	33583	78194	42464	41016	67172
s15850	100	80	0	0	0	0	55	0	0	0	1114	0	0	0
	90	80	0	0	0	0	55	0	0	0	1114	0	0	0
	80	80	0	0	0	0	55	0	0	0	1114	0	0	0
	60	80	870	965	1161	1161	1685	1810	2159	2188	3320	2934	3916	4113
	40	80	3290	3290	3290	3290	6504	6490	6490	6490	12959	12845	12845	12845
	20	80	5150	5150	5150	5150	10224	10210	10210	10210	20399	20285	20285	20285
	10	80	6903	6445	6445	6506	13696	12800	12800	12918	27330	25465	25645	25705
s35932	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	0	0	0	0	0	0	0	0	0	0	0	0
	40	80	0	0	0	0	0	0	0	0	0	0	0	0
	20	80	0	0	0	0	0	0	0	0	0	0	0	0
	10	80	0	0	0	0	0	0	0	0	0	0	0	0
s38417	100	80	0	0	0	0	13656	0	0	0	70358	2052	0	0
	90	80	24	0	0	0	15103	0	0	0	73157	2052	0	0
	80	80	48	0	0	0	16598	0	0	0	76016	2052	0	0
	60	80	13452	14133	15749	17928	36330	27806	29915	33033	102893	56352	57661	64431
	40	80	32000	32762	32259	31945	67186	64681	64603	63189	155447	128162	128292	126611
	20	80	43281	42866	43630	41545	91319	85760	87123	83119	201275	171198	173428	165776
	10	80	50670	50582	49618	48578	106012	101030	99250	97216	228992	201744	198096	194082
s38584	100	80	0	0	0	0	3319	0	0	0	23929	488	0	0
	90	80	412	314	187	187	4987	888	566	566	27432	3289	1287	1287
	80	80	1284	1260	822	152	7051	3196	2393	418	31563	9884	6697	1367
	60	80	5766	4473	3629	4080	16085	11101	9014	10183	46417	28598	22883	25744
	40	80	8415	8262	6917	7393	21639	18590	15190	16729	56181	42586	33921	38113
	20	80	12412	11860	9344	9370	29931	26550	20326	20654	71136	59071	44771	45645
	10	80	16236	15195	13603	13603	37087	33109	29454	29502	82009	70583	61692	61721

again the group size of 4 preforms the best. The reason for the best restoration in the case of group size of 4 can be the best combination of the effective width and the depth available for this grouping configuration. For restoration in last 20 cycles, the group sizes of 4, 8 and 16 perform better than no grouping case. Group size of 4 has the best restoration in most of the

Table 4.6: Comparison of Second Architecture Variants (Open-Core)

Circuit Name	% FFs Traced	# Cycles PIs Traced	#Unres Signals in Last 5 cycles				#Unres Signals in Last 10 cycles				#Unres Signals in Last 20 cycles			
			GOF1	GOF4	GOF8	GOF16	GOF1	GOF4	GOF8	GOF16	GOF1	GOF4	GOF8	GOF16
aes_core	100	80	0	0	28489	28854	40751	0	59613	74678	188403	12009	163789	192505
	90	80	64	0	28267	18339	43592	0	67525	54437	191244	12009	166104	188905
	80	80	124	0	16821	29445	45140	0	58961	71584	192792	13389	175915	189436
	60	80	170	0	18103	30036	46175	0	57270	63352	193827	12247	156052	187808
	40	80	947	0	11471	29690	46964	0	42380	65047	194616	11477	124070	179872
	20	80	3051	0	0	17309	49096	0	0	52465	196748	10291	0	147200
	10	80	14928	14832	14832	22278	61017	29894	29894	46322	208669	108793	108715	138188
des_area	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	0	0	0	0	0	0	0	0	0	0	0	0
	40	80	0	0	0	0	0	0	0	0	0	0	0	0
	20	80	0	0	0	0	0	0	0	0	0	0	0	0
	10	80	0	0	0	0	0	0	0	0	0	0	0	0
des3_area	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	0	0	0	0	0	0	0	0	0	0	0	0
	40	80	0	0	0	0	0	0	0	0	0	0	0	0
	20	80	0	0	0	0	0	0	0	0	0	0	0	0
	10	80	0	0	0	0	0	0	0	0	0	0	0	0
systemcaes	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	0	0	0	0	0	0	0	0	0	0	0	0
	40	80	0	0	0	0	0	0	0	0	0	0	0	0
	20	80	0	0	0	0	0	0	0	0	0	0	0	0
	10	80	0	0	0	0	0	0	0	0	0	0	0	0
wb_conmax	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	0	0	0	0	0	0	0	0	0	0	0	0
	40	80	0	0	0	0	0	0	0	0	0	0	0	0
	20	80	0	0	0	0	0	0	0	0	0	0	0	0
	10	80	0	0	0	0	0	0	0	0	0	0	0	0
wb_dma	100	80	0	0	0	0	0	0	0	0	0	0	0	0
	90	80	0	0	0	0	0	0	0	0	0	0	0	0
	80	80	0	0	0	0	0	0	0	0	0	0	0	0
	60	80	825	825	825	825	1650	1650	1650	1650	3300	3300	3300	3300
	40	80	2345	2345	2345	2345	4690	4690	4690	4690	9380	9380	9380	9380
	20	80	2345	2345	2345	2345	4690	4690	4690	4690	9380	9380	9380	9380
	10	80	3395	3395	3395	3395	6790	6790	6790	6790	13580	13580	13580	13580

cases. However, for the case of 20% tracing, the group size of 8 achieves 100% restoration.

From the above experiment it is clear that, the grouping in second architecture performs well as compared to no grouping. The multiplexed tracing especially in group sizes of 8 and 16 reduces the restoration in the last 5 cycles. However, for restoration in last 10 and 20 cycles, the larger group sizes perform better, because of the higher depth of tracing available.

The second experiment was performed to compare the first and second architectures. For

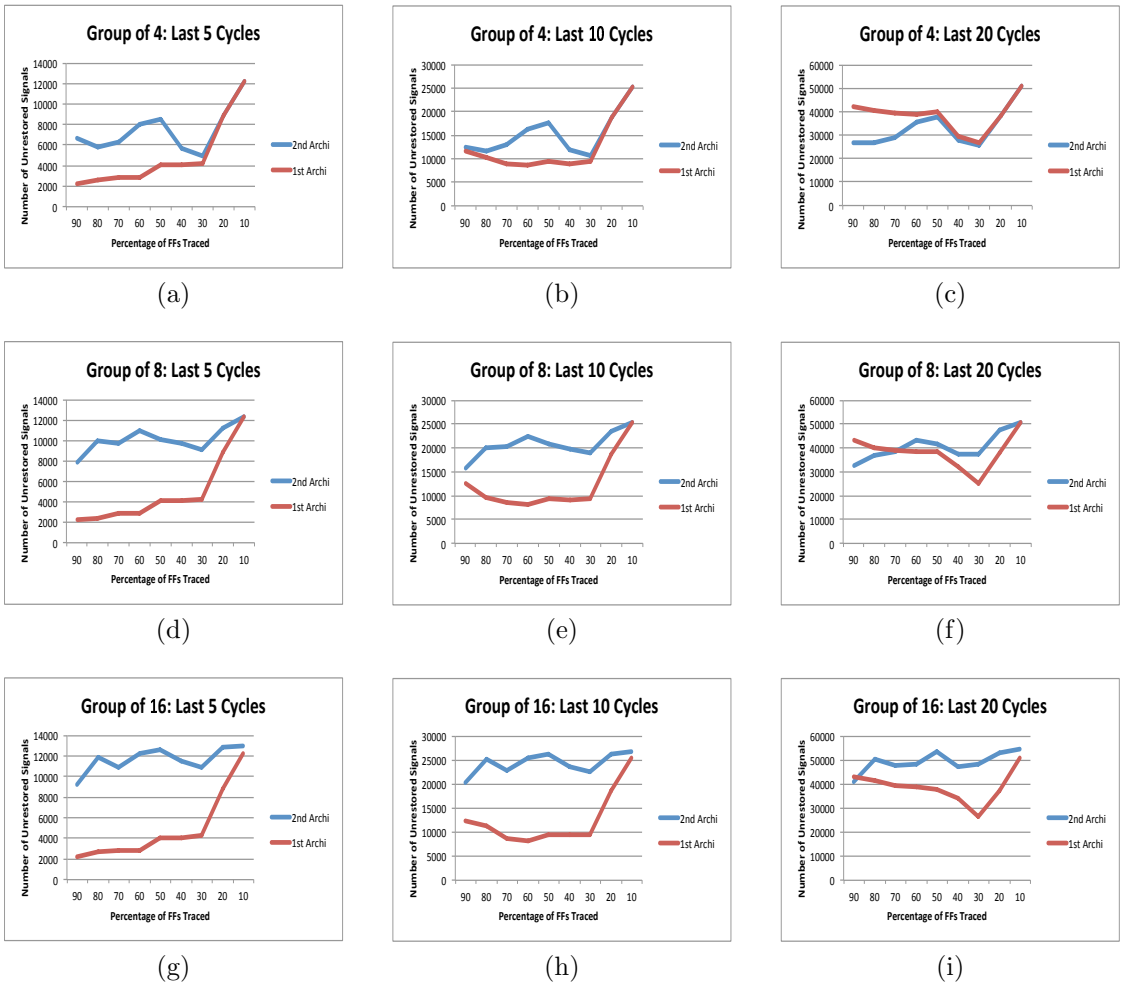


Figure 4.7: Comparison of First and Second Architecture: s5378

a given circuit, different percentage of FFs are traced for different grouping configurations (group of 4, 8 and 16) and the number of unrestored signals in last 5, 10 and 20 cycles is noted. The primary inputs are not traced in this experiment because we need to compare the two architectures purely based on the percentage of FFs traced. For each circuit, both the architectures are experimented. Note that while the first architecture provides a small dedicated shift register to each FF, the second architecture provides a dedicated shif register to the group of FFs. In first architecture, the complete wide tracing is possible but when a large percentage of FFs are traced, the depth of tracing available is smaller. On the other hand, the second architecture (wide multiplexed architecture) is effectively less wide than the first architecture and the frequency of tracing a selected signal decrease with the group

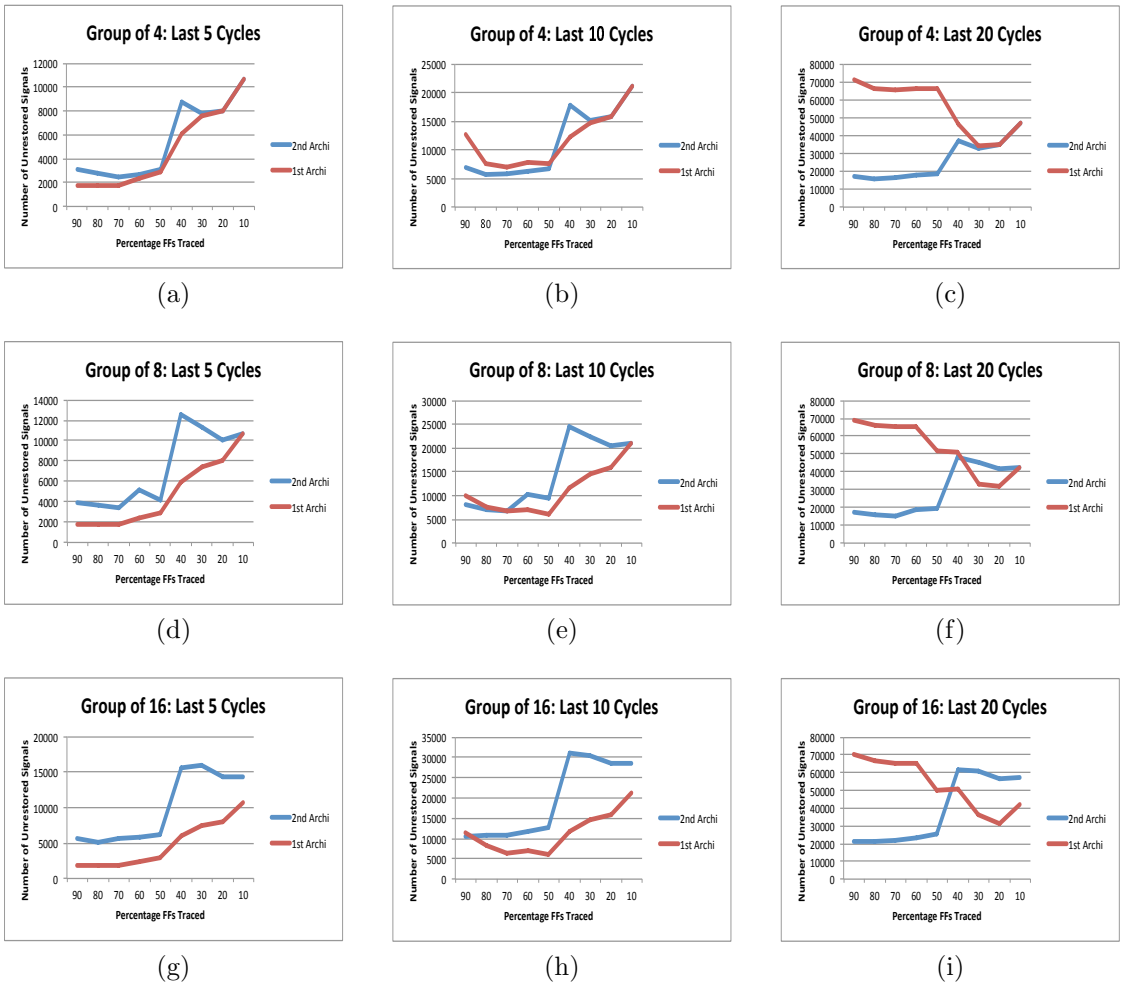


Figure 4.8: Comparison of First and Second Architecture: s9234

size and number of the FFs traced in a group. The advantage of the second architecture is a large overall tracing depth available. The group size of four FFs has tracing depth of 20 available to each group, the group size of 8 has tracing depth of 40 available to each FF and the group size of 16 has tracing depth availability of 80. This tradeoff in both the architecture is clearly reflected in the results. Figures 4.7, 4.8, 4.9 and 4.10 show the results for circuits, s5378, s9234, s13207 and s15850 respectively. The x-axis shows the percentage of FFs traced and the y-axis shows the number of unrestored signals in last k cycles, i.e. last 5, 10 and 20 cycles.

Let us consider the case of s5378 illustrated in figure 4.7. In the group size of 4 FFs,

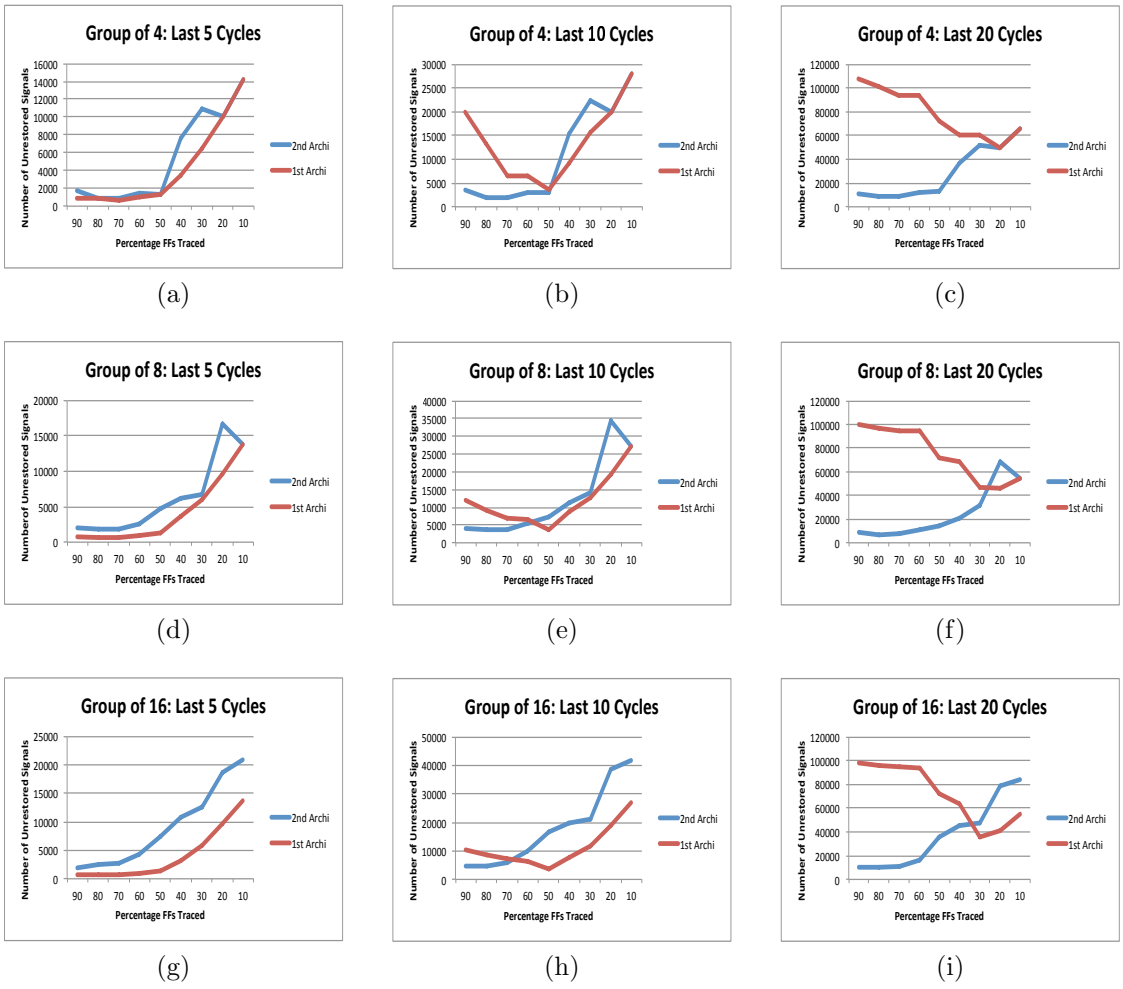


Figure 4.9: Comparison of First and Second Architecture: s13207

the second architecture has more number of unrestored signals in last 5 and 10 cycles as compared to the first architecture. However, for the case of last 20 cycles, the number of unrestored signals for second architecture is lower than first architecture case for most of the cases, especially when larger number of FFs are traced. These observation can be justified by the following reasoning. The first architecture allows wide-and-shallow tracing. In the case of tracing small percentage of FFs, like 10-30% FFs, we observe that number of unrestored signals in last 5 cycles for a group size of 4, is higher and almost matches the unrestored signals for second architecture. But, when percentage of FFs is increased, the number of unrestored signals in last 5 cycles decreases and a difference between the number of unrestored signals for first and second architecture is observed. The second architecture

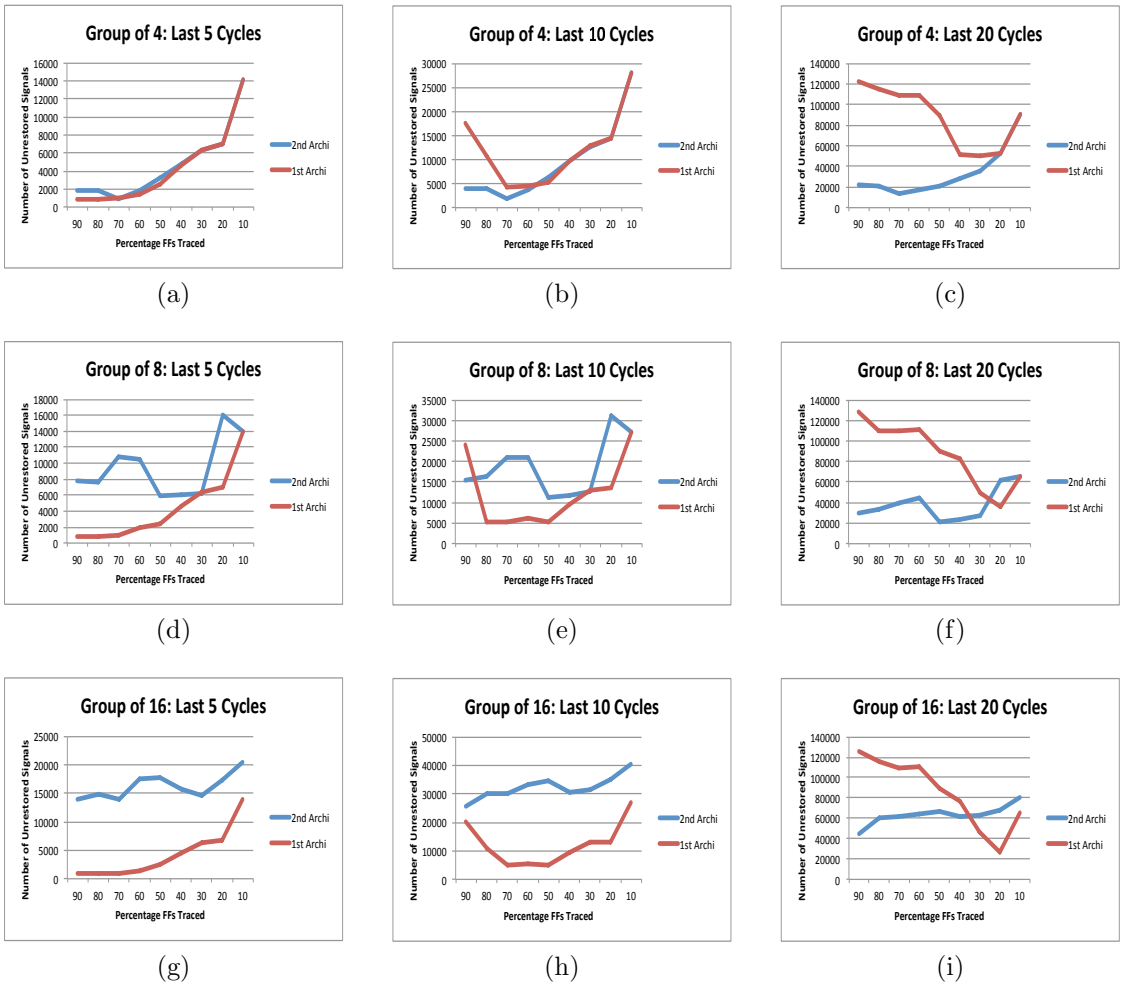


Figure 4.10: Comparison of First and Second Architecture: s15850

performance worse than the first architecture in this case because, the number of tracing isn't as wide as possible in first architecture and the depth of tracing available in second architecture doesn't help in this case as much as the width available to first architecture. Even with 100% tracing, a depth of 5 is available in the first architecture and so the number of unrestored signals is very low in last 5 cycles and is a decreased value when wider tracing is performed. For the same group size when number of unrestored signals in last 10 and last 20 cycles are noted, we observe that the difference between the first and second architecture is decreasing in the case of last 10 cycles and actually in last 20 cycles, the second architecture performs better than the first. This is because when a higher percentage of FFs are traced in first architecture, the number of FFs that can be traced to 20 cycles due to sharing are

very less thus leading to less restoration. On the other hand, in second architecture, though a multiplexed tracing is performed, the total depth available for tracing is still 20 for all groups which comes to its advantage and thus the performance is better.

When we consider bigger group sizes, the trend is similar. However, the difference between the first and second architecture widens with higher group sizes. While, the first architecture, exhibits a similar restoration pattern with nearly equal restoration values, for all group sizes, the second architecture performs worse as the group size increases. This is because the effective width and frequency of tracing a particular signal decreases with the increasing group size. The higher group size, comes with an advantage of higher tracing depth available but the effect of depth availability is not seen as much in the restoration in last 5, 10 and 20 cycles. If the restoration in large number of cycles (e.g.. 40 or 50 cycles) is observed then the second architecture will perform better given the advantage of higher depth available.

In circuits, s9234, s13207 and s15850, similar trend is observed and can be found as illustrated in figures, 4.8, 4.9 and 4.10 respectively.

4.5 Conclusion

This chapter details the second trace buffer architecture that performs multiplexed tracing. The aim is to maximize the signal restoration during post-silicon validation. This architecture also provides the flexibility of selecting different sets of trace signals according to the debugging needs. While smaller groups provide wide-and-shallow tracing configuration, the larger groups provide lower effective width but a larger depth of tracing. Experiment results compare the different variants of the second architecture based on the group sizes. The smaller group sizes perform well in the restoration in the last 5 cycles because of the higher effective width. However, the performance in larger group sizes is not compromised because the tracing is deeper and in spite of lower effective width and lower frequency of tracing, most of the untraced FFs are restored because of deeper tracing and thus the overall restoration is not too low. For restoration in deeper cycles (last 10 and last 20 cycles), the larger groups perform better because of the deeper tracing availability. In certain configurations, a good trade-off between the width and tracing depth, achieves 100% restoration. This chapter also investigates the influence of tracing the primary inputs on the restoration and presents the area overhead calculation when all the PIs are traced. Finally, the first

and second architectures are compared for four different ISCAS'89 benchmark circuits for different percentage of FFs traced in different grouping configurations. The result graphs presented illustrate the comparison of the two architectures.

Chapter 5

Invariants and Signal Restoration

This chapter details the work on finding circuit invariants and highlights their potential utility for high signal restoration. The following section explains the need for circuit invariants in the application domain of signal restoration for post-silicon validation. Sections 5.2 and 5.3 elaborate the procedure used for finding useful 2-node invariants and 3-node invariants in the circuit respectively. Section 5.4 discusses the steps followed for using invariants in the signal restoration application. Section 5.5 presents the results and Section 5.6 lays the foundation for future work.

5.1 Need of Circuit Invariants for Signal Restoration

The two novel trace buffer architectures proposed in Chapters 3 and 4 target high signal restoration in most critical cycles of debug and also offer the flexibility of deeper signal restoration and configurable trace signal selection. With all the advantages of the proposed architectures, there are certain circuit cases where complete restoration is not achieved and a few signals still remain unresolved. The restoration is also lower when PIs in the circuit are not traced. All these scenarios provide a motivation to improve the method of restoration. The enhanced restoration strategy can be combined with the proposed architectures, and much better signal restoration can be achieved.

One of the most promising candidates of improving the signal restoration method is addition of useful invariants to the circuit formula. Invariants in general and circuit invariants in particular are unique relations between different signals in the circuit that hold true for all

reachable states of the circuit. Invariants can be added as clauses to the CNF formula and the implication-based engine that is used to restore the untraced signals, can restore more with the help of the unique structural information captured by the invariants. Thus useful circuit invariants are required in view of improving the signal restoration during post-si validation.

The following sections detail the procedures used in this work for finding 2-node invariants and 3-node invariants.

5.2 Finding 2-Node Invariants

Following steps are used to find the 2-node invariants in the circuit.

5.2.1 Generating Signatures

Simulation is performed for a set of 1000 vectors for the circuit and signature for all the gates are generated. Signature for a gate i is the set of values of i for all the vectors 1 to k where k is the maximum number of vectors used for simulation. Figure 5.1 illustrates how the signature for every gate can be stored.

	Vectors				
	v1	v2	v3	v4	v5
Gate 1	1	0	1	0	1
Gate 2	1	0	0	0	1
Gate 3	0	1	1	1	0
Gate 4	0	1	0	0	1
Gate 5	1	0	1	0	0
Gate 6	0	1	0	0	1

Figure 5.1: 2D Signature Array for 6 Gates and 5 Vectors

5.2.2 Selecting Combinations of the signals

In the circuit a combination of each signal with every other signal may have a unique relation, but it is not possible to consider all the combinations within time constraints. Moreover,

some combinations may just have direct relations that are already embedded in the CNF of the circuit and those combinations may not give any improvement in restoration. To prune the number of combinations of signals selected, we first filter out the combinations of signals that exist around the same gate. This is done by a simple structural analysis of each combination. Secondly, the signals that have direct relation between them are filtered out. Out of the two signals, the signal at the lower level is set to a value say 0, and with all the other signal values as don't cares, forward simulation is performed to know if the second signal in the combination (the one at a higher level) changes its value. This process is repeated the second time with the other signal value at the first gate, in this example with value 1. All the combinations that exhibit the direct relations between the signals are not considered.

While selecting the combinations we also filter out all combinations with the gates that are constants in the circuit.

5.2.3 Finding Missing Patterns

A combination of 2 gates that is selected can have 4 different signal patterns. For example, say, a selected combination consists of gate numbers 1 and 6. The four patterns of the gates will be (-1, -6), (-1, 6), (1, -6) and (1, 6), where 1 denotes that gate 1 has a value 1 and -1 denotes gate 1 has value 0. Similarly, 6 implies gate 6 with value 1 and -6 implies gate 6 with value 0. Signatures of the gates are traversed to know if any of the patterns is missing from the entire simulation. If a pattern is present for any vector, we drop it. All the patterns that are absent are marked as "Potential Invariants" of the circuit. Say the pattern (1, -6) is not observed through the simulation, we mark it as a potential invariant candidate. This means (1, -6) may not at all be possible for the circuit. When gate 1 is 1, gate 6 cannot hold 0 value.

5.2.4 Reducing the Number of Potential Invariants

As the size of circuit increases, the number of potential invariant candidates increase to a great extent. While finding the potential invariant candidates, we filter out some of the candidates, if they are implied by the other potential candidates already considered. For e.g say (1, -6) is a potential invariant candidate. We assume that this candidate is a true

invariant (1, -6 is not possible) and so if a gate say 8 implies 1, we conclude that (8, -6) is also true invariant and just drop it because it is implied by (1, -6). Certainly a risk is incurred in the sense that, if (1, -6) was a false invariant, (8, -6) may have been a good potential invariant candidate. But we experiment limiting the number of potential invariants in this manner so that the number of candidates to be verified could be reduced.

5.2.5 Validating an Invariant

To validate a potential invariant, we plug in the signal values from the invariant in the CNF formula of the circuit. A recursive Boolean Constraint Propagation (BCP) is performed and if a conflict is caused, the potential invariant candidate is validated to be true. The true invariant is added to the original CNF of the circuit to prune the search space while verifying the following potential invariant candidates. For example, if plugging in 1 and -6 in the CNF causes a conflict, the invariant (1, -6) is verified. This implies signal values 1 (gate 1 = 1) and -6 (gate 6 = 0) cannot co-exist and a clause (-1 + 6) is added to original CNF of the circuit. Clearly, the clause will yield false when gate 1 = 1 and gate 6 = 0 are updated in this clause, thus this clause stores the information of the invariant previously found which specifies, 1 and -6 cannot coexist.

All the verified true invariants are stored in a file for their potential help in signal restoration.

5.3 Finding 3-Node Invariants

Finding 3 node variants follow similar procedure but with some differences. Following are the steps which are used to establish 3 node invariants.

5.3.1 Generating Signatures

Generating and storing signatures follow the same method as described in the section 5.2.1. The signatures are traversed for each combination to find the missing patterns which can be the potential invariants.

5.3.2 Selecting Combinations of the gates

As the number of combination of 3 gates in a circuit (especially larger benchmarks) is huge, the number of gates which are used in any combination are pruned in the first step.

As mentioned in section 5.2.2, the gates which are constant in the circuit are filtered out initially and their combinations with any other gates are not considered. The initial set of gates that are considered for creating the combinations are as follows:

1. 1^{st} node is selected from a pool of top t% of the high implication FF signal values in the time frame 0. The top t% (top 20% and top 5% used in this work) of the FF signal values with maximum number of implications are selected for this purpose. Note that the *signal value* of a FF with highest number of implications is considered first. That means, number of implications for both values of a FF are found separately (setting the FF as 0 and setting the FF as 1), and out of all the signal values across all FFs, the ones with highest number of implications are considered first. For example, if a FF number 36 with value 0 has highest number of implications, the combinations with -36 are considered.

2. 2^{nd} node is selected from a pool of all FFs in the circuit in time frame 1.

3. 3^{rd} node is selected from a pool of top t% high implication signal values across all gates in time frame 1. For our experiments, the top 20% and top 5% of signal values across all gates with highest implications are considered.

Selecting the high implication gates for two nodes out of three, can potentially give higher chances of restoring different gates, and thus the overall restoration can be higher.

After selecting 3 gates for the combination, it is analyzed if the signals in the combination are around a same gate, or direct forward simulation establishes any relation between the gates of the combination. In any such case, the combination is dropped and not considered for further analysis.

5.3.3 Finding Missing Patterns

As the signal values instead of gates are considered (for 2 nodes out of 3) while selecting the combinations, all 8 patterns in a combination of 3 gates are not possible. For example, in the first pool, FF 36 with the signal value 0 has highest implications, so our aim is to include

-36 in the invariant. Say from the 2^{nd} pool, a FF 49 is selected for the combination. As the second pool does not consider the signal values, both values of FF 49 will be used. Say from the third pool of top 20% highest implication signal values, the gate 119 with signal value 1 is selected. So 119 (signal value = 1) should occur in the invariant.

Thus from the combination of 3 gates (36, 49 and 119), we want to look for 2 missing patterns, (36, 49, -119) and (36, -49, -119). If these missing patters are truly not possible (cause a conflict and validated), the true invariants will form the clauses (-36 + -49 + 119) and (-36 + 49 + 119). Note that the signal values with high implications are present in the invariant. -36 is gate 36 with signal value 0 and 119 is gate 119 with signal value 1.

Thus, for a combinations of 3 gates, instead of 8 patterns, we only analyze 2 patterns. If these 2 patterns are missing in the entire simulation, these 2 combinations are marked as "Potential Invariant" candidates and are validated in the next step.

5.3.4 Validating an Invariant

Each potential invariant is validated by plugging in the missing pattern in the circuit formula and then performing recursive BCP. If any potential invariant causes the conflict in the formula, it is marked as a true invariant, and is added to the circuit formula permanently.

All the verified 3-node invariants are stored in a separate file for potential use in signal restoration.

5.4 Combining Invariants and Constants to the circuit formula for signal restoration

Following steps are performed to investigate the effect of constants and invariants on signal restoration:

1. The circuit is unrolled for the maximum unrolling depth for which tracing is performed for any FF. The unrolled circuit is converted to the CNF form.

2. For each instance of the circuit, the 1-node clauses, specifying constants in the circuit are added to the CNF formula of the circuit.

Table 5.1: 2-Node Invariants

Circuit Name	Potential Invariants	Potential Invariants after Reduction	True Invariants
s5378	3,067,380	430,777	1,806
s9234	34,826,895	5,124,366	5,664
s13207	65,918,684	10,001,630	16,328
s15850	134,954,298	23,015,347	8,352

Table 5.2: 3-Node Invariants

Circuit Name	Percentage of Signals in Pools 1 and 3	Potential Invariants	True Invariants
s5378	20%	1,001,497	8,206
s9234	20%	15,089,981	21,343
s13207	5%	6,941,085	26,401
s15850	5%	10,506,147	8,658

3. The effect on signal restoration is noted after adding the constants to the formula.

4. In the next step, 2-node clauses, specifying 2-node invariants are added to the formula and the effect on restoration is noted.

5. Lastly, 3-node clauses, specifying 3-node invariants are added to the circuit formula and the effect on restoration is studied.

5.5 Experimental Results

Table 5.1 shows the number of 2-node invariants mined using the procedure described in section 5.2. First column gives the circuit name, the second column shows the number of potential invariants without reduction. The third column gives the number of potential invariants after reduction procedure as discussed in section 5.2.4. The reduced number of potential invariants go through the process of validation by plugging in each invariant in the CNF of the circuit and then performing recursive BCP to know if it causes a conflict. The right-most column gives the number of 2-node invariants validated to be true invariants.

Table 5.2 shows the results for the 3-node invariants. Column 1 gives the circuit name, the second column gives the percentage of signals selected for pools 1 and 3. Column 3 gives the number of potential invariants and the fourth column gives the number of invariants that are validated to be true. Note that the number of signals used for making the combinations are far less than the total number of signals present in the circuit. The signals for the 3-node invariants are picked from the three pools of signals as discussed in section 5.3.2.

Tables 5.3, 5.4 and 5.5 give the results on restoration after successively adding the constants, 2-node invariants and 3-node invariants to the circuit formula. Table 5.3 presents the results on the number of unrestored signals in the last 5 cycles, table 5.4 presents the case of unrestored signals in the last 10 cycles and the table 5.5 presents the results for number of unrestored signals in the last 20 cycles.

[4pt][4pt]

This experiment is performed on the first architecture with a group configuration of 4 FFs in a group. The experiment is repeated for 4 different ISCAS'89 benchmark circuits. The five columns on the left give the circuit name, group size, number of constants, number of 2-node invariants and number of 3-node invariants in the circuit respectively.

For each circuit, different percentage of FFs are traced and the number of unrestored signals is noted. First case is of initial circuit formula with no constants or invariants. In the second run constants are added to the circuit formula and the restoration results are noted.

In the third run, 2-node invariants are added to the formula with the constants and the restoration is noted. In the final run, additional 3-node invariants are added to the circuit formula and the results are noted.

The results show that for many of the cases there is a significant amount of reduction in number of unrestored signals in the last 5, 10 and 20 cycles when the constants are added to the circuit formula. For the case of unrestored signals in last 5 cycles, the maximum reduction obtained is 100% for the case of the 70% tracing for circuit 13207. The number of unrestored signals become 0. Another case of high reduction is 50% tracing in s13207 where a reduction of 57.56% in number of unrestored signals is noted. For the unrestored signals in last 10 and 20 cycles, the reduction values are higher is almost all the cases as compared to unrestored signals in last 5 cycles.

When we add the 2-node invariants to the circuit formula, further more reduction is observed

Table 5.3: Restoration in Last 5 Cycles with Constants and Invariants

Ckt Name	Group Size	No. of Constants	2-Node Invariants	3-Node Invariants	Per FFs Traced	#Unsres Last 5					
						w/o I & C	w/ C	% Dec	w/ 2-N I	% Dec	w/ 3-N I
s5378	4	404	1,806	8,206	100	0	0	0	0	0	0
					90	0	0	0	0	0	0
					80	0	0	0	0	0	0
					70	0	0	0	0	0	0
					60	0	0	0	0	0	0
					50	0	0	0	0	0	0
					40	0	0	0	0	0	0
					30	0	0	0	0	0	0
					20	0	0	0	0	0	0
					10	1189	1189	0	1189	0	1189
s9234	4	74	5,664	21,343	100	0	0	0	0	0	0
					90	0	0	0	0	0	0
					80	0	0	0	0	0	0
					70	0	0	0	0	0	0
					60	630	630	0	630	0	630
					50	1155	1155	0	1155	0	1155
					40	3079	3078	3.24	2410	21.72	2410
					30	3261	3222	1.195	2554	21.68	2554
					20	3501	3462	1.11	2794	20.19	2794
					10	6050	6009	0.67	5251	13.20	5251
s13207	4	616	14,470	26,401	100	0	0	0	0	0	0
					90	178	178	0	178	0	178
					80	182	182	0	182	0	182
					70	19	0	100	0	100	0
					60	359	275	23.39	275	23.39	275
					50	648	275	57.56	275	57.56	275
					40	1040	970	6.73	970	6.73	970
					30	3439	3439	0	3199	6.97	3199
					20	7446	7336	1.47	7081	4.90	7081
					10	10763	10698	0.60	10447	2.93	10447
s15850	4	116	8,352	8,658	100	0	0	0	0	0	0
					90	27	27	0	27	0	0
					80	29	26	10.34	26	10.34	26
					70	177	177	0	177	0	177
					60	595	595	0	595	0	595
					50	1503	1503	0	1503	0	1503
					40	3399	3399	0	3399	0	3399
					30	5160	5160	0	5130	0.58	5130
					20	5160	5160	0	5130	0.58	5130
					10	6931	6931	0	6901	0.43	6901

for all the cases. However, the amount of reduction is low. Significant amount of reduction in number of unrestored signals is observed in the case of circuit s9234 for tracing 10%-40% FFs. This significant amount of reduction is observed in all the cases of unrestored signals in last 5, 10 and 20 cycles. A significant reduction is also observed in case of circuit s13207 for different percentage of FFs traced when unrestored signals in last 5, 10 and 20 cycles are noted. For example, case of last 10 cycles, 20 and 30% tracing and 60-100% tracing gives

Table 5.4: Restoration in Last 10 Cycles with Constants and Invariants

Ckt Name	Group Size	No. of Constants	2-Node Invariants	3-Node Invariants	Per FFs Traced	#Unsres Last 10					
						w/o I & C	w/ C	% Dec	w/ 2-N I	% Dec	w/ 3-N I
s5378	4	404	1,806	8,206	100	15210	1804	88.13	1804	88.13	1804
					90	2604	1804	30.72	1804	30.72	1804
					80	2440	1725	29.30	1725	29.30	1725
					70	2005	1222	39.05	1222	39.05	1222
					60	1622	877	45.93	877	45.93	877
					50	1331	0	100	0	100	0
					40	25	0	100	0	100	0
					30	25	0	100	0	100	0
					20	6	0	100	0	100	0
					10	2404	2398	0.24	2398	0.24	2398
s9234	4	74	5,664	21,343	100	29330	29330	0	29330	0	29330
					90	8848	8334	5.80	8268	6.55	8268
					80	2814	2628	6.60	2611	7.21	2611
					70	2351	2167	7.82	2153	8.42	2153
					60	3434	3276	4.60	3262	5.00	3262
					50	3900	3734	4.25	3734	4.25	3734
					40	6471	6446	0.38	4893	24.38	4893
					30	6608	6519	1.34	4966	24.84	4966
					20	7088	6999	1.25	5446	23.16	5446
					10	12105	12014	0.75	10324	14.71	10324
s13207	4	616	14,470	26,401	100	43860	14769	66.32	12184	77.22	12184
					90	17042	14160	16.91	12176	28.55	12176
					80	11035	10956	0.715	9263	16.05	9263
					70	5051	2276	54.93	1182	76.59	1182
					60	5118	1300	74.59	1131	77.90	1131
					50	2293	737	67.85	700	69.47	700
					40	3385	3245	4.13	2510	25.84	2510
					30	8039	8039	0	6492	19.24	6492
					20	14679	14457	1.51	13960	4.89	13960
					10	21173	21041	0.62	20546	2.96	20546
s15850	4	116	8,352	8,658	100	52350	15236	70.89	15025	71.29	15025
					90	9312	8589	7.76	8496	8.76	8496
					80	8329	7926	4.83	7870	5.51	7870
					70	2865	2842	0.80	2834	1.08	2834
					60	3128	3105	0.73	3097	0.99	3097
					50	3245	3241	0.12	3241	0.12	3241
					40	6841	6841	0	6840	0.01	6840
					30	10328	10328	0	10268	0.58	10268
					20	10328	10328	0	10268	0.58	10268
					10	13852	13852	0	13792	0.43	13792

a significant amount of reduction in number of unrestored signals. All these cases where adding 2-node invariants give a significant reduction in unrestored signals for circuit s9234 and s13207 are marked in pink in the result tables.

When the 3-node invariants are added to the circuit formula with the constants and the 2-node invariants, no further reduction is noted for all the cases.

Table 5.5: Restoration in Last 20 Cycles with Constants and Invariants

Ckt Name	Group Size	No. of Constants	2-Node Invariants	3-Node Invariants	Per FFs Traced	#Unsres Last 20					
						w/o I & C	w/ C	% Dec	w/ 2-N I	% Dec	w/ 3-N I
s5378	4	404	1,806	8,206	100	45630	32224	29.37	32224	29.37	32224
					90	33024	32224	2.42	32224	2.42	32224
					80	32860	32145	2.17	32145	2.17	32145
					70	32425	31642	2.41	31642	2.41	31642
					60	32042	31297	2.32	31297	2.32	31297
					50	31751	2093	93.40	2093	93.40	2093
					40	3250	2035	37.38	2035	37.38	2035
					30	3064	1935	36.84	1935	36.84	1935
					20	3501	2351	32.84	2351	32.84	2351
					10	8859	7457	15.82	7457	15.82	7457
s9234	4	74	5,664	21,343	100	87990	87990	0	87990	0	87990
					90	67508	66994	0.76	66928	0.85	66928
					80	61474	61288	0.30	61271	0.33	61271
					70	61011	60827	0.30	60813	0.32	60813
					60	62094	61936	0.25	61922	0.27	61922
					50	62560	62394	0.26	62394	0.26	62394
					40	35012	34376	1.81	31722	9.39	31722
					30	20121	19757	1.80	16408	18.45	16408
					20	21281	20898	1.79	17538	17.58	17538
					10	32331	31979	1.08	28384	12.20	28384
s13207	4	616	14,470	26,401	100	131580	102489	22.10	99904	24.07	99904
					90	104762	101880	2.75	99896	4.64	99896
					80	98755	98676	0.08	96983	1.79	96983
					70	92771	41553	55.20	24783	73.28	24783
					60	92838	38626	58.39	24567	73.53	24567
					50	67740	37228	45.042	23680	65.04	23680
					40	40276	40276	0	25402	36.93	25402
					30	42287	42074	0.50	25933	38.67	25933
					20	39174	37898	3.25	35834	8.52	35834
					10	52871	51165	3.22	49101	7.13	49101
s15850	4	116	8,352	8,658	100	157050	119936	23.63	119725	23.76	119725
					90	114012	113289	0.63	113196	0.71	113196
					80	113029	112626	0.35	112570	0.40	112570
					70	107565	107542	0.021	107534	0.028	107534
					60	107828	107805	0.021	107797	0.021	107797
					50	73097	72902	0.26	72815	0.38	72815
					40	27481	27151	1.20	26892	2.14	26892
					30	30348	30174	0.57	29958	1.28	29958
					20	25683	25650	0.12	25448	0.91	25448
					10	40062	39634	1.06	39282	1.94	39282

The results above highlight the effect of adding constants and invariants on the signal restoration. As evident in the results, the relations with less number of nodes give better reductions in number of unrestored values. This is because lesser nodes in an invariant give a high chance of a unit-clause and hence more information can be deduced and used from such invariants.

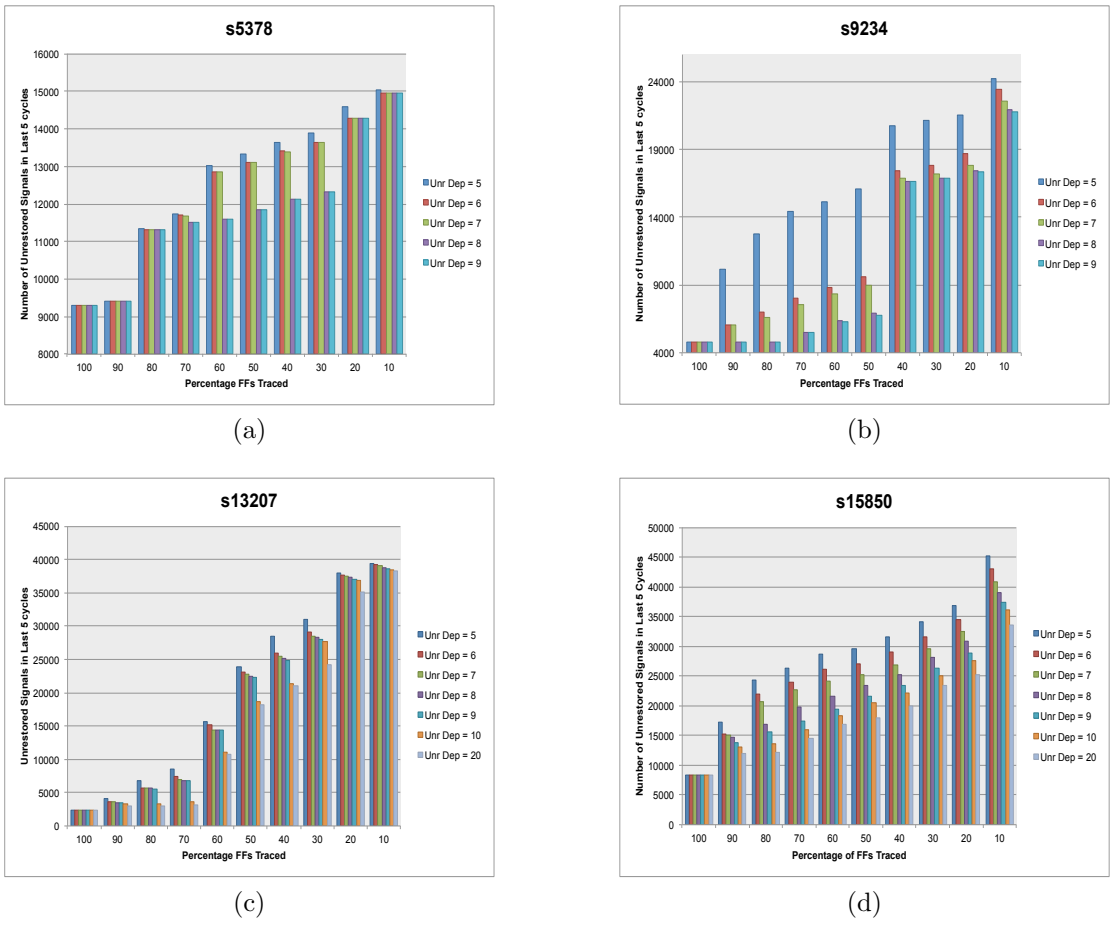


Figure 5.2: Increased Restoration with Increased Unrolling Depth

5.6 Direction for Future Research

To improve the signal restoration with the help of invariants, there is a need to mine a very good set of relations among signals in the circuit. The work done in investigating the effect of invariants on the signal restoration suggests that, the current invariants do not restore everything in the last 5, 10 and 20 cycles. In many cases, the reduction in number of unrestored signals after using the invariants is very low. This section details the potential direction for future work in mining a better set of invariants that improves restoration to a great extent.

A simple experiment was performed where a circuit is unrolled for 5 cycles in the first run.

The FFs in the first cycle are traced and the number of unrestored signals in the 5 cycles is noted. In the 2nd run of the experiment, the circuit is unrolled for 6 time frames, and the same set of FFs are traced in the same cycle as previous run. The restoration is performed again using the BCP implication engine and the number of unrestored signals in the last 5 cycles is noted. We found that, there is a reduction in the number of unrestored signals when exactly the same FFs are traced in same cycles, only by increasing the number of cycles on which the BCP is performed. When BCP is performed on the longer set of cycles, a hidden set of implications in the CNF of the circuit are deduced while doing BCP and aid the restoration. The experiment is repeated for 7, 8, 9, 10, 20, 50 and 100 cycles. We noted the results for circuits s5378, s9234, s13207, and s15850. While we noticed the successive reduction in number of unrestored signals upto unrolling depth of 9 for s5378 and s9234, the number of unrestored signals decreased upto unrolling depth of 20 for circuits s13207 and s15850. The results are shown in Figure 5.2, where successive reductions in the number of unrestored signals is seen with increased unrolling depth. The results are shown for different cases of tracing 100% FFs to 10%FFs for circuits s5378, s9234, s13207 and s15850.

This experiment clearly implies:

1. A good set of invariants should cross time frames, a long span of time frames. The invariants that bear relation among signals in the time-frames that are further apart can possibly give better restorations when added to the circuit formula.

2. A good set of invariants should be able to deduce the signals on the edge of the circuit. This means, the invariants should be able to deduce more number of primary inputs and FFs in the circuit, because if the signals on the edge are resolved they could restore more signals in the given time-frame and would keep resolving more and more signals down the road.

The inference from the experiment can be used to find better set of invariants for improved restoration in future.

Another potential application of invariants that can be employed for better restoration is trace signal selection criteria. In the previous works and in this thesis work, trace signals are selected based on the number of unchecked implications. A signal selection criteria that also uses the number of invariants in which a signal is contained, can be potentially useful. If we add the invariants to the circuit formula for the restoration, and select the trace signals based of the number of invariants they are contained in, the chances of deducing other untraced

signals in the circuit is higher and can give better restoration results.

5.7 Conclusion

This chapter explains in detail, the mining of useful 2-node and 3-node invariants in the circuit. The effect of invariants on the restoration is investigated. The results show improvement in the restoration when invariants are added to the circuit formula. For mining a better set of invariants, direction for future work is provided that aims higher restoration. The following chapter concludes the thesis and also presents the future work that is directed by the thesis.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presents two novel architectures for trace buffer design for facilitating post-silicon validation and test. Both of the architectures focus on wide tracing as opposed to narrow tracing that is popular in conventional trace buffer design. In the first architecture, a small shift register is provided to each FF of the circuit and all the FFs can be traced simultaneously up to the depth of the shift register. In the case of narrow tracing, the FFs in a group share their shift registers and deeper tracing is achieved. Experimental results show that the proposed architecture is able to achieve 100% restoration and the restoration performance is similar to the conventional architecture when narrow tracing is performed. Also, as the trace buffer is distributed across all FFs, the routing overhead of connecting the signals to a separate on-chip trace buffer is reduced.

The second architecture provides a shift register to a group of FFs and multiplexed tracing is performed if more than one FF in a group is traced. The experiments investigate and compare the restoration obtained by different group sizes. The idea of effective width is proposed and the variation in restoration is observed with the trade-off between effective width and the depth of tracing available. The area overhead of tracing PIs is computed and presented. The first and second architectures are compared for the restoration in last k cycles, where value of k varies as 5, 10 and 20. Both the architectures are compared for three different group size configurations, group sizes 4, 8 and 16.

The final part of the thesis explains, in detail, mining of useful invariants in the circuit. The

effect of adding constants (1-node invariants), 2-node invariants and 3-node invariants on restoration is investigated. A direction for future work is provided which aims at finding a good set of invariants to improve the signal restoration further.

6.2 Recommendations for Future Work

Following are some recommendations for the future work:

1. The multiplexers that are employed for sharing the shift registers in first proposed architecture and that used for multiplexed tracing in the second proposed architecture, can be combined with the multiplexers present in the existing scan chain in the circuit, thereby reducing the area overhead.

2. A hybrid architecture that combines the two proposed architecture can be investigated.

3. A better set of invariants (targeted invariants) can be mined which will facilitate higher signal restoration.

4. Better techniques on reducing the potential number of invariants can be used so that good potential invariant candidates are not dropped.

5. Invariants can be used to devise a better trace signal selection technique such that the set of trace signals lead to better restoration.

Bibliography

[1] K. Constantinides, O. Mutlu, and T. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," *in 41st IEEE/ACM International Symposium on Microarchitecture,* 2008, pp. 282-293.

[2] A. Carbine and D. Feltham, "Pentium(R) Pro processor design for test and debug," *in Proc. International Test Conf.,* 1997, pp. 294-303.

[3] G.J. Van Rootselaar and B. Vermeulen, "Silicon debug: scan chains alone are not enough," *in International Test Conf. Proc.,* 1999, pp. 892-902.

[4] B. Vermeulen and S.K. Goel, "Design for debug: catching design errors in digital chips," *IEEE Design & Test of Computers,* 2002, vol. 19, no. 3, pp. 35-43.

[5] Q. Xu and X. Liu, "On signal tracing in post-silicon validation," *in 15th Asia and South Pacific Design Automation Conf,* 2010, pp. 262-267.

[6] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," *in 43rd ACM/IEEE Design Automation Conf.,* 2006, pp. 7-12.

[7] J. Gao, Y. Han, and X. Li, "A new post-silicon debug approach based on suspect window," *in IEEE VLSI Test Symposium,* 2009, pp. 85-90.

[8] S. Prabhakar and M. Hsiao, "Using non-trivial logic implications for trace buffer-based silicon debug," *in Asian Test Symposium,* 2009, pp. 131-136.

[9] H.F. Ko and N. Nicolici, "Automated trace signals identification and state restoration for improving observability in post-silicon validation," *in Proc. of the Conf. on Design, Automation and Test in Europe,* 2008, pp. 1298-1303.

[10] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation," *in Design, Automation and Test in Europe Conf.,* 2009, pp. 1338-1343.

[11] N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *in IEEE Trans. Comput. Des. Integr. Circuits Syst.,* 2009, vol. 28, no. 2, pp. 285-297.

[12] D. Chatterjee, C. McCarter, and V. Bertacco, "Simulation-based signal selection for state restoration in silicon debug," *in International Conf. on Computer-Aided Design,* 2011, pp. 595-601.

[13] K. Basu and P. Mishra, "RATS: restoration-aware trace signal selection for post-silicon validation," *in IEEE Trans. Very Large Scale Integr. Syst.,* 2013, vol. 21, no. 4, pp. 605-613.

[14] M. Li and A. Davoodi, "A Hybrid approach for fast and accurate trace signal selection for post-silicon debug," *in Design, Automation and Test in Europe Conf.,* 2013, pp. 485-490.

[15] K. Han, J.S. Yang, and J.A. Abraham, "Dynamic trace signal selection for post-silicon validation," *in 26th International Conf. on VLSI Design,* 2013, pp. 302-307.

[16] X. Liu and Q. Xu, "On multiplexed signal tracing for post-silicon validation," *in IEEE Trans. Comput. Des. Integr. Circuits Syst.,* 2013, vol. 32, no. 5, pp. 748-759.

[17] X. Liu and Q. Xu, "Interconnection fabric design for tracing signals in post-silicon validation," *in Proc. of the 46th Annual Design Automation Conf. DAC,* 2009, pp. 352.

[18] B.R. Quinton and S.J.E.Wilton, "Concentrator access networks for programmable logic cores on SoCs." *in Proc. International Symposium on Circuits and Systems (ISCAS),* 2005, pp. 45-48.

[19] S. Prabhakar and M.S. Hsiao, "Multiplexed trace signal selection using non-trivial implication-based correlation," *in 11th International Symposium on Quality Electronic Design (ISQED),* 2010, pp. 697-704.

[20] H.F. Ko and N. Nicolici, "Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging," *in 15th IEEE European Test Symposium,* 2010, pp. 62-67.

[21] K. Basu, P. Mishra, and P. Patra, "Efficient combination of trace and scan signals for post silicon validation and debug," *in IEEE International Test Conf.,* 2011, pp. 1-8.

[22] K. Han, J.S. Yang, and J.A. Abraham, "Enhanced algorithm of combining trace and scan signals in post-silicon validation," *in IEEE 31st VLSI Test Symposium (VTS),* 2013, pp. 1-6.

[23] K. Rahmani and P. Mishra, "Efficient signal selection using fine-grained combination of scan and trace buffers," *in 26th International Conf. on VLSI Design,* 2013, pp. 308-313.

[24] E. Anis and N. Nicolici, "On using lossless compression of debug data in embedded logic analysis," *in IEEE International Test Conference,* 2007, pp. 1-10.

[25] E. Anis and N. Nicolici, "Low cost debug architecture using lossy compression for silicon debug," *in Design, Automation & Test in Europe Conference & Exhibition,* 2007, pp. 1-6.

[26] E. Anis and N. Nicolici, "On using lossy compression for repeatable experiments during silicon debug," *in IEEE Trans. Comput.,* Jul. 2011, vol. 60, no. 7, pp. 937-950.

[27] K. Basu and P. Mishra, "Efficient trace data compression using statically selected dictionary," *in 29th VLSI Test Symposium,* 2011, pp. 14-19.

[28] S. Prabhakar, R. Sethuram, and M.S. Hsiao, "Trace buffer-based silicon debug with lossless compression," *in 24th International Conference on VLSI Design,* 2011, pp. 358-363.

[29] N.H.E. Weste and K. Eshraghian, "Principles of CMOS VLSI design: a systems perspective. Addison-Wesley Pub. Co.," 1993.

[30] M. Abramovici, "In-system silicon validation and debug," *IEEE Des. Test Comput.,* 2008, vol. 25, no. 3, pp. 216-223.

[31] A. Nahir, A. Ziv, M. Abramovici, A. Camilleri, R. Galivanche, B. Bentley, H. Foster, A. Hu, V. Bertacco, S. Kapoor, "Bridging pre-silicon verification and post-silicon validation," *Design Automation Conference (DAC),* 2010 pp.94,95, 13-18.

[32] C. MacNamee and D. Heffernan, "Emerging on-chip debugging techniques for real time embedded systems," *ITE Computer & Control Engineering Journal,* 2008 vol. 11, no. 6, pp 295-303.

[33] K.H. Chang, I.L. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," *in Proc. IEEE ASP-DAC,* Jan. 2007, pp. 944-949.

[34] J. Zhao, J. Newquist and J. Patel, "A graph traversal based framework for sequential logic implication with an application to c-cycle redundancy identification," *Proc. VLSI Design Conf.,* 2001, pp. 163-169

[35] C.A.J. Van Eijk, "Sequential equivalence checking based on structural similarities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Jul 2000, vol.19, no.7, pp.814-819.

[36] S.Y. Huang; K.T. Cheng; K.C. Chen, "AQUILA: an equivalence verifier for large sequential circuits," *Design Automation Conference, Proceedings of the ASP-DAC '97 Asia and South Pacific,*, 1997, pp.455,460, 28-31.

[37] S. Hangal, S. Narayanan, N. Chandra, and S. Chakravorty, "IODINE: a tool to automatically infer dynamic invariants for hardware designs," *in Proceedings. 42nd Design Automation Conference,* 2005, pp. 775-778.

[38] F. Lu and K.T. Cheng, "IChecker: an efficient checker for inductive invariants," *in IEEE International High Level Design Validation and Test Workshop,* 2006, pp. 176-180.

[39] W. Weixin, M.S. Hsiao, "SAT-based state justification with adaptive mining of invariants," *IEEE International Test Conference,* pp.1,10, 28-30

[40] G. Cabodi, S. Nocco, and S. Quer, "Boosting the role of inductive invariants in model checking," *in Design, Automation and Test in Europe Conference and Exhibition,* pp. 1-6.

[41] F. Zheng, Y. Weng, and X. Yan, "An efficient sequential equivalence checking framework using boolean satisfiability," *in 7th International Conference on ASIC,* pp. 1174-1177.

[42] W. Hu, H. Nguyen, and M.S. Hsiao, "Sufficiency-based filtering of invariants for sequential equivalence checking," *in IEEE International High Level Design Validation and Test Workshop,* 2011, pp. 1-8.

[43] H. Nguyen and M.S. Hsiao, "Sequential equivalence checking of hard instances with targeted inductive invariants and efficient filtering strategies," *in IEEE International High Level Design Validation and Test Workshop (HLDVT),* 2012, pp. 1-8.

[44] M. Wedler, D. Stoffel, and W. Kunz, "Exploiting state encoding for invariant generation in induction-based property checking," *in ASP-DAC: Asia and South Pacific Design Automation Conference,* 2004, pp. 424-429.

[45] F. Lu, L.C. Wang, K.T. Cheng, and R.C.Y. Huang, "A circuit sat solver with signal correlation guided learning," *in Proc. Design Aut. and Test in Europe Conf.,* 2003 pp. 892-897.

[46] W. Wu and M. Hsiao, "Mining global constraints for improving bounded sequential equivalence checking," *in Proc. Design Automation Conf.,* 2006, pp. 743-748.

[47] N. Goel, M.S. Hsiao, N. Ramakrishnan, and M.J. Zaki, "Mining complex boolean expressions for sequential equivalence checking," *in Proc. Asian Test Test Symp.,* 2011, pp. 442.

[48] C.L. Chang, C.H.P. Wen, and J. Bhadra, "Speeding up bounded sequential equivalence checking with cross-time frame state-pair constraints from data learning," *in Proc. Int. Test Conf.,* 2009, pp. 1-8