EFFICIENT OBSERVABILITY ENHANCEMENT TECHNIQUES FOR POST-SILICON VALIDATION AND DEBUG

By

KANAD BASU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2012

I dedicate this to my family.

ACKNOWLEDGMENTS

I would like to sincerely thank my Ph.D. advisor Prof. Prabhat Mishra, without whose guidance, this dissertation would not have been possible. I would also like to thank my Ph.D. committee members: Prof. Sartaj Sahni, Prof. Sanjay Ranka, Prof. Greg Stitt and Prof. Ann Gordon-Ross for their valuable suggestions. I am also thankful to my labmates Weixun Wang, Xiaoke Qin, Mingsong Chen, Kartik Srivastava, Chetan Murthy, Hadi Hajimiri and Kamran Rahmani for their help and support. I would also like to thank my mentors during my two internships at Intel Corporation – Dr. Dhurbajyoti Kalita and Dr. Priyadarsan Patra.

I would also like to take this opportunity to thank those who helped me during different stages of my research. I sincerely thank Dr. Henry Ko from McMaster University and Dr. Xiao Liu from the Chinese Univeristy of Hong Kong for helping me understand various aspects of trace signal selection and signal restoration. I am grateful to Dr. Ilya Wagner from Intel Corporation for explaining me the constraints and objectives associated with test generation for post-silicon validation. I would like to thank Prof Krishnendu Chakraborty and Zhanglei Wang from Duke University, and Dr. Mehrdad Reshadi from University of California, Irvine for helpful suggestions.

I would also like to extend my gratitude towards my family to help me reach this stage. Finally, I would like to thank Mr. Sachin Tendulkar, Mr. Akira Kurosawa, Mr. Paolo Coelho and Mr. John Denver, who through their lives and immortal creations have always encouraged me to move forward, even in times of despair.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

EFFICIENT OBSERVABILITY ENHANCEMENT TECHNIQUES FOR POST-SILICON VALIDATION AND DEBUG

By

Kanad Basu

August 2012

Chair: Prabhat Mishra
Major: Computer Engineering

Post-silicon validation is widely acknowledged as a major bottleneck for complex integrated circuits. Due to increasing design complexity coupled with shrinking time-to-market constraints, it is not possible to detect all design flaws (errors) during pre-silicon verification. Post-silicon validation needs to capture these escaped functional errors as well electrical faults. A major concern during post-silicon debug is the observability of internal signals since the chip has already been manufactured. Design overhead considerations limit the number of signal states that can be traced or stored in a trace buffer. This dissertation proposes novel techniques to enhance the observability during post-silicon debug. My research has three major contributions: profitable signal-selection, efficient trace compression and observability-aware test generation. It proposes efficient signal selection techniques to enhance the observability of the circuit. Various signal selection constraints are explored including static versus dynamic, trace versus scan and general-purpose versus architecture-specific features. To improve the observability further, an efficient trace compression approach has been proposed. Extensive experimental results demonstrate significant improvement in overall signal observability. This dissertation also proposes observation-aware directed test generation techniques to drastically reduce the overall post-silicon validation effort.

CHAPTER 1
INTRODUCTION

Design complexity is increasing rapidly keeping pace with two-fold increase in number of transistors every technology cycle. Drastic increase in design complexity has led to significant increase in validation complexity. The report from International Technology Roadmap for Semiconductors (ITRS) [1] as well as other reputed agencies indicate that it is critical to develop efficient design verification techniques to secure design productivity scaling at a pace consistent with process technology cycles. There has been a plethora of research efforts in both industry and academia to develop scalable design validation approaches using a combination of simulation based techniques and formal methods. In spite of extensive efforts, it is not always possible to detect all the functional errors and electrical faults during pre-silicon validation. Post-silicon validation is used to detect design flaws including the escaped functional errors as well as electrical faults. Post-silicon validation is widely acknowledged as a major bottleneck for complex integrated circuits including modern microprocessors as well as complex System-on-Chip (SoC) designs. Various industrial studies indicate that the post-silicon validation effort consumes more than 50% of an SoCs overall design effort (measured in total cost) at 65nm technology [2]. These studies also emphasize the fact that the problem gets worse as the industry continues to move to even smaller geometries.

Figure 1-1 shows the overview of three important validation and testing phases in a typical SoC design methodology. Pre-silicon validation effort includes validation of various functional as well as timing requirements across abstraction levels including specification and implementation. Manufacturing testing is primarily used to detect physical (structural) defects in each of the manufactured ICs. On the other hand, the focus of post-silicon validation is to detect design flaws that have escaped pre-silicon validation. In reality, vast majority of functional errors are captured in the pre-silicon stage. While a small percentage of functional errors remain, the time required to find and fix them is still very expensive. It is important to note that majority of the electrical faults (including crosstalk, delay and transient faults) are captured during post-silicon validation, since it is difficult to model electrical errors during the pre-silicon verification phase.

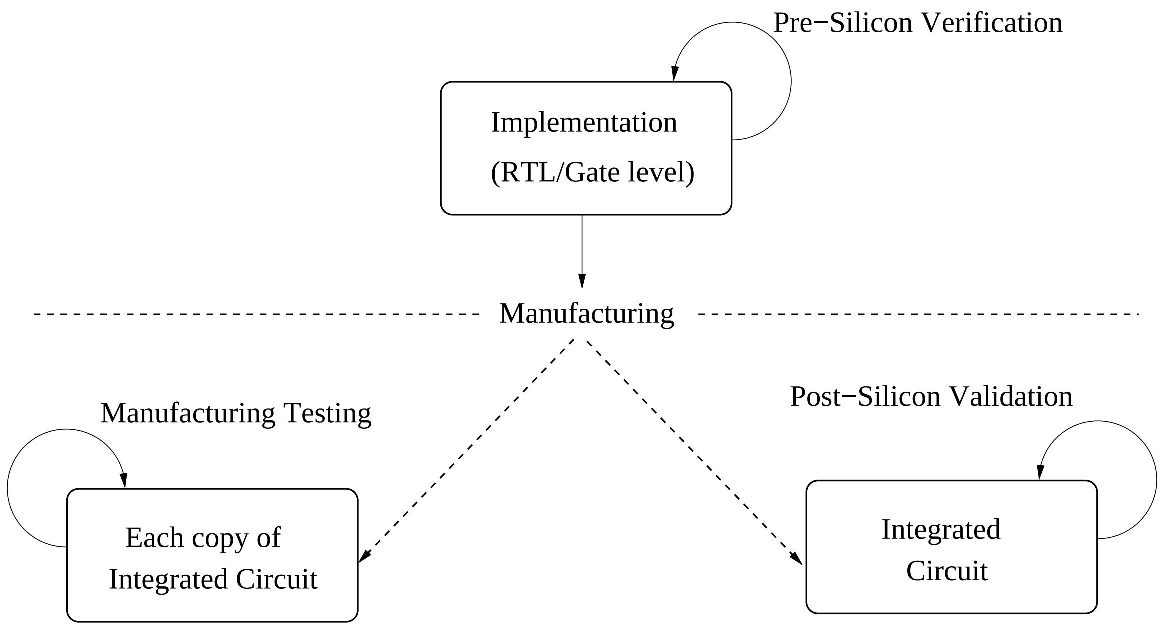


Figure 1-1. Validation and testing phases in IC design flow

Figure 1-2 presents an overview of the post-silicon validation and debug methodology. Input tests are applied to the Device Under Test (DUT). Depending of the test generation techniques, the input tests can be random, constrained-random or directed in nature. During execution (run time), states of some selected signals are traced and stored in an on-chip trace buffer. Note that the signals whose states to be stored are decided during the pre-silicon (design) phase using suitable signal selection algorithms. To increase the effective trace buffer size, various trace compression techniques can be employed. When a failure is detected, the contents of the trace buffer is dumped out to aid in post-silicon debug using an offline debugger.

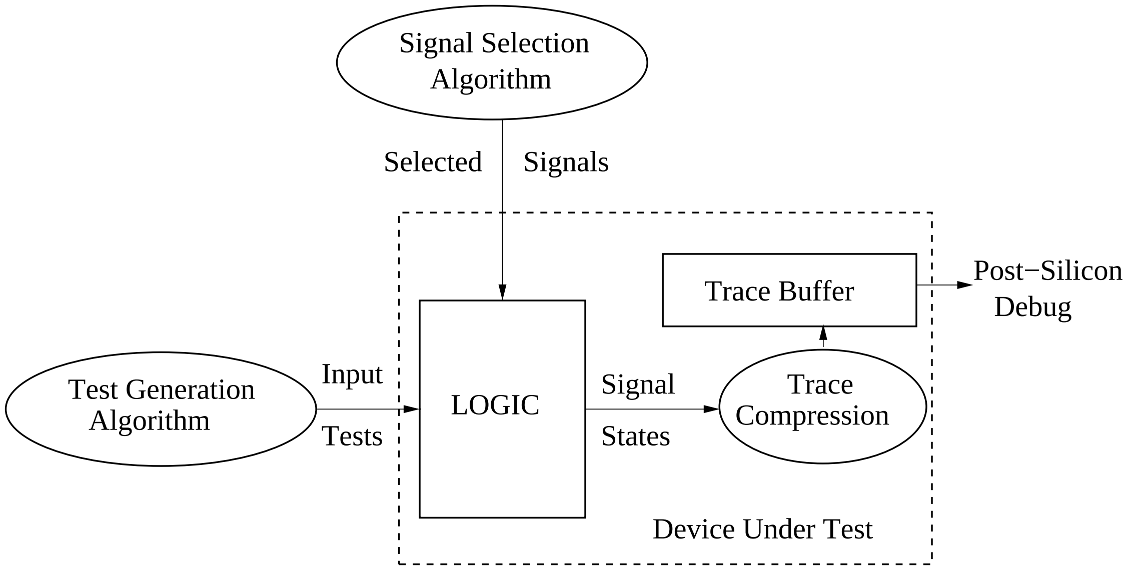


Figure 1-2. Overview of post-silicon validation

The remainder of this chapter is organized as follows. First we discuss about potential faults and defects in a System-on-Chip (SoC). Next, we describe basics of signal restoration and compare signal restoration and error detection. Finally, we discuss various challenges associated with post-silicon validation and outline our contributions to address these challenges.

1.1 Faults and Defects in Integrated Circuits

Defects and errors may get introduced at different phases of SoC design cycle - design time, synthesis, manufacturing, etc. Efficient fault modeling of these are necessary to effectively analyze, detect and fix them. Functional errors are introduced during development of specification as well as implementation. It is extremely important to detect and fix these functional errors. Some of the common defects in a chip [3] include processing defects (parasitic transistors, oxide breakdown, etc.), material defects (surface impurities), time-dependent failures (electromigration, dielectric breakdown, etc.) and packaging failures (seal break). Studies [4] reveal that of all the defects observed in a Printed Circuit Board (PCB), 51% are due to shorts, thus making electrical errors also important in any SoC validation methodology. Common fault models used to model defects and errors are:

- Single Stuck-at-Fault (represented as $s-a-0$ or $s-a-1$)
- Transistor Open and Short Fault
- Memory Faults
- Glitch Faults
- PLA Faults
- Functional Errors
- Delay Faults
- Analog Faults

Post-silicon validation deals with errors and defects corresponding to all these fault models. However, in case of manufacturing testing, errors primarily related to manufacturing defects (such as stuck-at faults and transistor opens/shorts) are considered.

Electrical defects manifest as important errors in modern SoCs. Soft errors and crosstalk faults are two important defects that can adversely affect the correct functionality of the chip. While soft errors are caused by radioactive effects on design impurities, crosstalk faults occur due to imperfect coupling capacitance between two lines in the chip. Soft errors can be modeled using single stuck-at-faults. Effects of crosstalk can be represented as glitches and delay faults. Effective directed test generation strategies need to be employed in order to detect these faults. The tests should be able to activate the faults and propagate them towards the observation points, e.g., primary outputs or internal trace signals.

Now we discuss about two important concepts related to signal selection, signal restoration and error detection. Since only a few signals can be observed during post-silicon validation, it is important to restore the unknown signal states as much as possible to enhance the observability of the chip. On the other hand, it is equally important to actually observe the errors through the trace signals.

1.2 Restoration of Unknown Signals

In post-silicon debug, unknown signal states can be reconstructed from the traced signal states in two ways - forward and backward restoration. Forward restoration deals with the restoration of signals from input to output, that is, knowledge of input states can help in restoring the value of the output. Backward restoration, on the other hand, deals with reconstructing the input from the output. Forward and backward restoration can be illustrated using the example in Figure 1-3. We use a 2-input AND gate to explain the restoration process. Forward restoration is shown in Figure 1-3(a). When one input is 0 or both inputs are 1, the output can be constructed. Figure 1-3(b) shows backward restoration. When the output is 1 or the output 0 with one input 1, the other input can be reconstructed. It is easier to reconstruct signals using forward rather than backward restoration. If all but the unknown signal values are known, forward restoration can definitely determine the unknown, while backward might fail to do so in specific cases. For example, in Figure 1-3(c), when the output is 0 and one of the input is 0, there is no way to determine the state of the other input. Although we have illustrated the signal reconstruction using a 2-input AND gate, the restoration procedure can also be described in a similar manner for other types of logic gates as well as with more inputs.

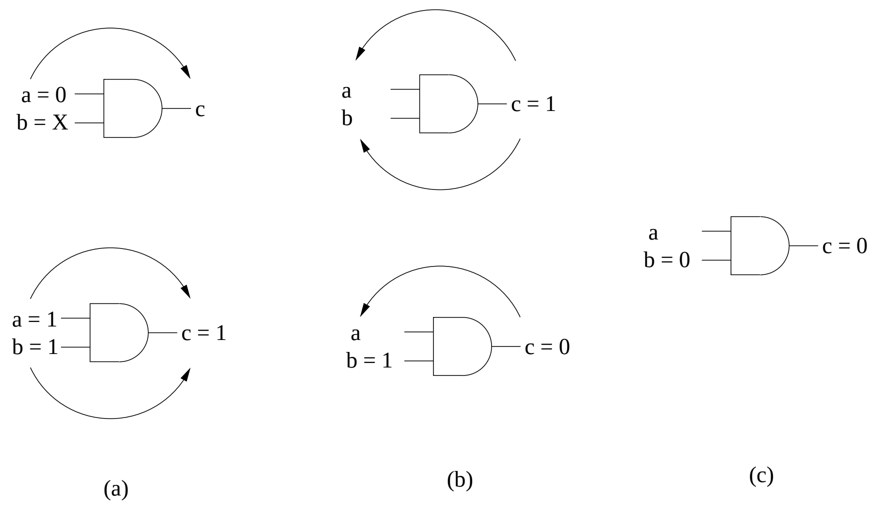


Figure 1-3. Signal restoration

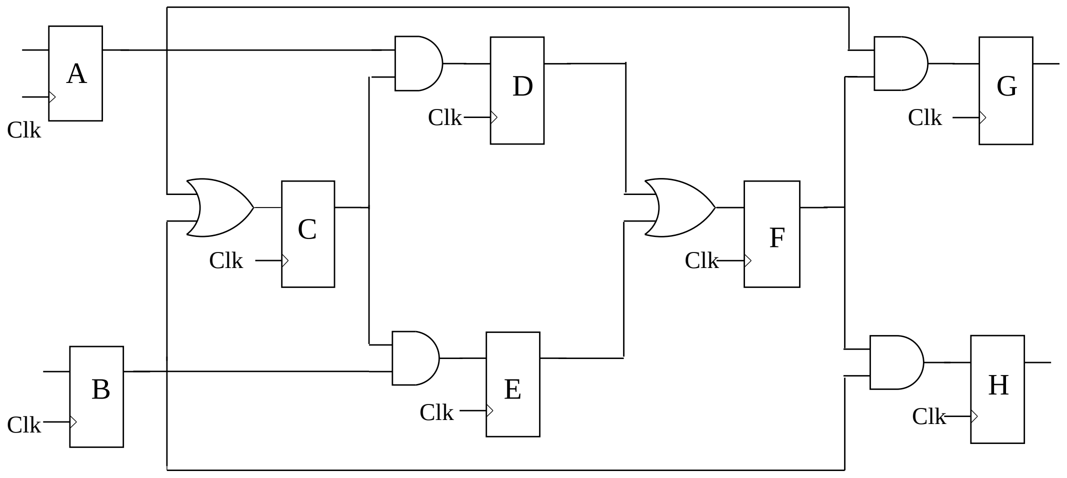


Figure 1-4. Example circuit

The signal reconstruction procedure is illustrated using a simple circuit shown in Figure 1-4. Let us assume that the trace buffer width is 2, that is, states of two signals can be recorded. We try to restore the other signal states by application of the existing signal selection methods presented in [5] and [6]. The results are shown in Table 1-1. The 'X's represent those states which cannot be determined. The selected signals are shown in shades. For both [5] and [6] the signals selected are C and F, in that order. Restoration ratio, which is a popular metric for calculation of signal restorability is defined as:

$$Restoration\ Ratio = \frac{number\ of\ states\ restored\ +\ number\ of\ states\ traced}{number\ of\ states\ traced}. \tag{1–1}$$

Let us calculate the number of restored states in Table 1-1. If we consider the row corresponding to signal A, two entries have value 0, while the rest have value X (non-restored state). Thus, two states are known. Similarly, two states are known for the row corresponding to signal B. Since signal C is traced, all the states are known (no X in the row). For signal D, three entries in the row have value 0, hence three states are reconstructed. Computing in this manner, a total of 26 states are reconstructed. Out of them, 10 entries (corresponding to signals C and F) are traced states. Therefore, Restoration Ratio in this case is 2.6.

Table 1-1. Restored signals using two selected signals

Signal	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
A	X	0	X	0	X
B	X	0	X	0	X
C	1	1	0	1	0
D	X	X	0	0	0
E	X	X	0	0	0
F	0	1	1	0	0
G	X	0	0	X	0
H	X	0	0	X	0

1.3 Signal Restoration Versus Error Detection

This section compares two widely used metric, state restoration and error detection. Majority of the existing signal selection approaches [6, 7] try to focus on restoration of unknown signals using the knowledge of known signal states. Let us consider an example circuit in Figure 1-5 comprised of 12 flip-flops. In Figure 1-5, tracing states of flip-flops A and B in cycle t, helps to restore the state of flip-flop D in cycle $t+1$, since the input to flip-flop D is an OR of the outputs of flip-flops A and B. Similarly, since flip-flops C and H are connected by a NOT gate, tracing H in cycle t provides the state of C in cycle $t-1$. Note that signal restoration can proceed in both forward (input to output) and backward (output to input) direction. For example, the restoration of D from A and B was in forward direction, while the restoration of C from H is in backward direction.

In case of error detection only backward restoration is meaningful. To explain this scenario, a part of the illustrative example of Figure 1-5 has been redrawn in Figure 1-6. When flip-flop D is in error, the bug can only propagate along a forward direction in its fan-out cone towards flip-flops E and G. Therefore, in order to detect the error in D, we have to trace either of these two flip-flops. Tracing the inputs of D, or those flip-flops that are in its fan-in cone (A and C) do not help. Given that a designer is interested in detecting an error, it is meaningful to focus directly on error detection metric instead of restoration performance.

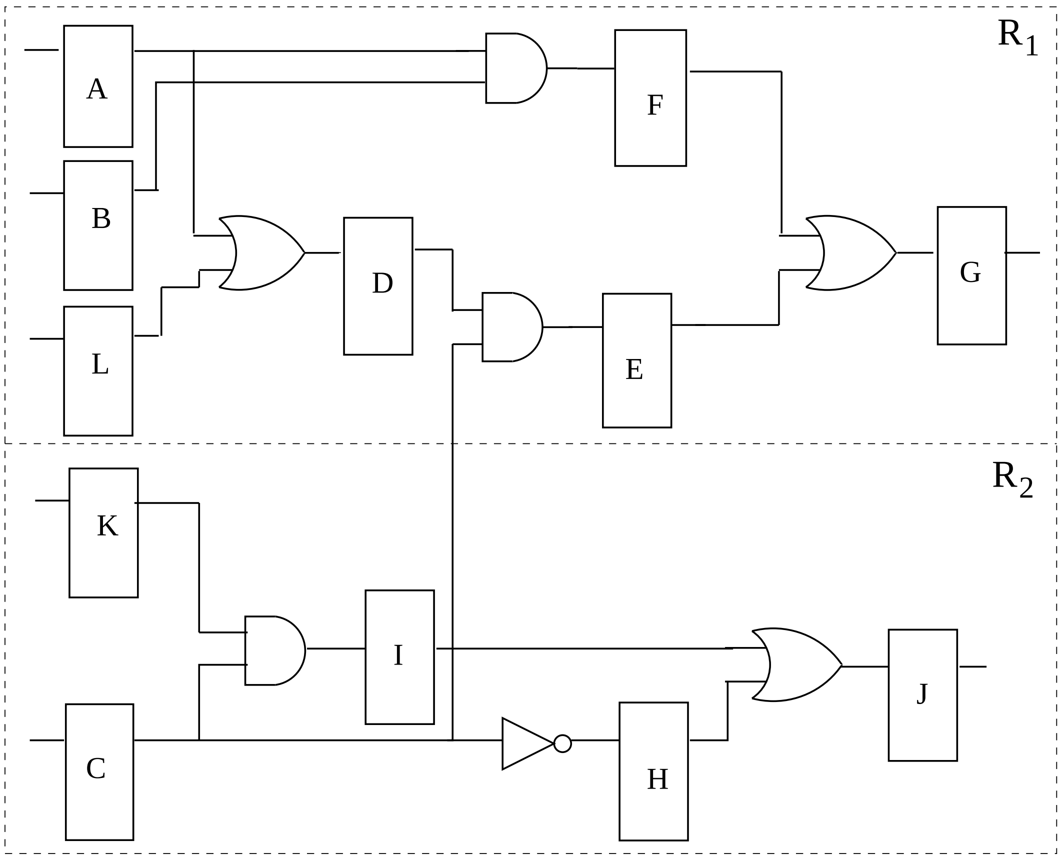


Figure 1-5. Example circuit with 12 flip-flops

1.4 Challenges

There are many challenges in developing efficient techniques for post-silicon validation and debug. This section describes five important challenges that I have addressed in this dissertation.

Challenge 1: The signals to be traced should be selected carefully in order to maximize the restoration. Signal selection techniques based on partial restoration (described in Chapter 3) were proposed by Ko et al. [5] and Liu et al. [6]. If the trace buffer width is *n*, both these approaches selected *n* signals with highest partial restoration abilities for tracing. However, signal selection based on partial restoration does not provide the best reconstruction. Also, these methods are computationally inefficient, since they require long signal selection time operating on gate-level netlists.

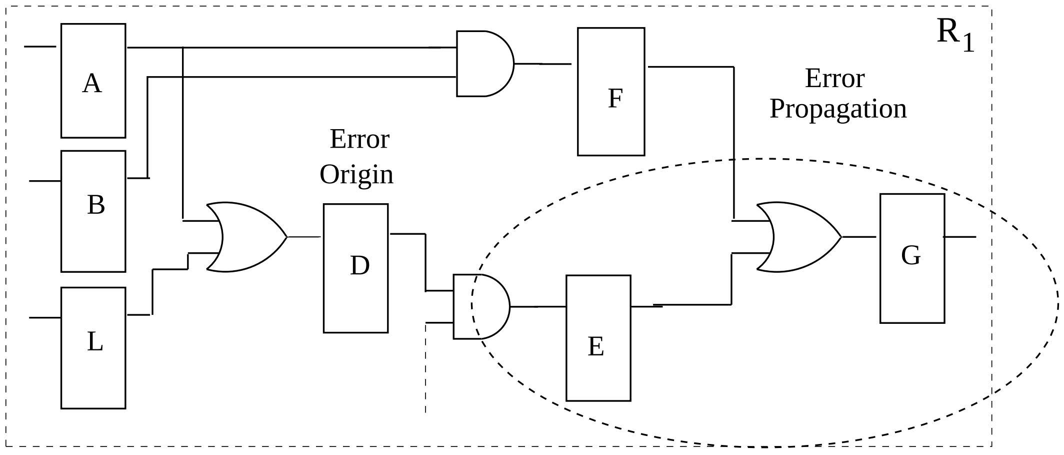


Figure 1-6. Error Propagation for the example circuit in Figure 1-5

An efficient algorithm is needed which can select trace signals providing high restoration of untraced signals with fast signal selection time.

Challenge 2: The time requirement for gate-level signal selection algorithms is high because of the excessive number of variables used to represent flip-flops and other internal signals. One promising alternative to reduce signal selection time is to explore at higher abstraction level like Register Transfer Level (RTL). Ko et al. [8] developed a signal selection approach which selects some signals from the RTL-level and some from the gate-level netlist description of the circuit, that is, their signal selection does not depend on the RTL-level design alone. It is desired that a signal selection technique be developed which solely relies on the RTL-level implementation of the design, thus reducing the memory and time requirement associated with any gate-level netlist. The primary challenge to develop an RTL-level signal selection algorithm is to ensure that this approach does not incur any restoration penalty compared to gate-level signal selection algorithms.

Challenge 3: Scan based debugging is popular in manufacturing test domain. They are primarily used to identify fabrication defects. The use of scan chains for improving signal observability during post-silicon debug has been extensively studied [9–11]. Several approaches [12, 13] have studied the combination of scan and trace signals for post silicon debug. A major challenge is that the number of trace signals, scan signals and scan dump frequency are inter-dependent. For example, selecting more trace signals implies less space for scan signals, and vice versa. Even when the space for scan signals is reserved, choosing a large scan chain (too many scan signals) implies longer scan dump frequency. In other words, there needs to be a critical balance between how many signals to observe versus how many signal states can be obtained in a specific clock cycle.

Challenge 4: Till now, we have considered how tracing some signals helps us to reconstruct the untraced signals. Debugging involves detection of errors by the traced signals. Thus, it is necessary to trace signals with an aim of maximizing the number of errors detected. During debug, some parts of the circuit may be less important for debug purposes than the rest during particular cycles. For example, the debug engineer might be interested in detecting errors in the “processor” block of an SoC instead of the “memory” block. Therefore, it is important to develop a signal selection algorithm that can dynamically select signals based on the currently active regions to enhance error detection.

Challenge 5: The circuit observability can be further enhanced by compressing the trace buffer contents. This allows more signal states to be stored in the trace buffer without increasing its size. Since the trace buffer does not contribute to the actual chip functionality except debug, its size should be as small as possible. The trace buffer has two parameters, width and depth. Width refers to the total amount of debug data that can be stored per cycle, while depth refers to the total number of cycles over which debug data can be stored. In order to keep the trace buffer size constant, while increasing the amount of trace data that can be stored, either the depth or the width has to be compressed. Different techniques of trace data compression, either by depth [14, 15] or by width [16] have been proposed. Depth compression approaches deal with selecting the cycles where the data are likely to be erroneous, and store the data for only those cycles. On the other hand, in width compression, the trace data obtained every cycle is first compressed and then stored in the trace buffer. An efficient lossless trace data compression technique is necessary which can provide both fast compression and high compression efficiency without introducing significant hardware overhead.

Challenge 6: Efficient signal selection and trace compression enhance the observability during post-silicon debug. However, it is equally important to improve controllability aspect during post-silicon validation. The input tests applied to the circuit should be carefully designed in order to maximize error detection. This involves estimating the corner case scenarios and designing tests that can activate those scenarios to allow the errors to propagate to the traced signals. Therefore, it is important for the test designer to have knowledge of the signals that are being traced. Conversely, if the tests are known a priori, the trace signals can be selected efficiently to enhance error detection capability.

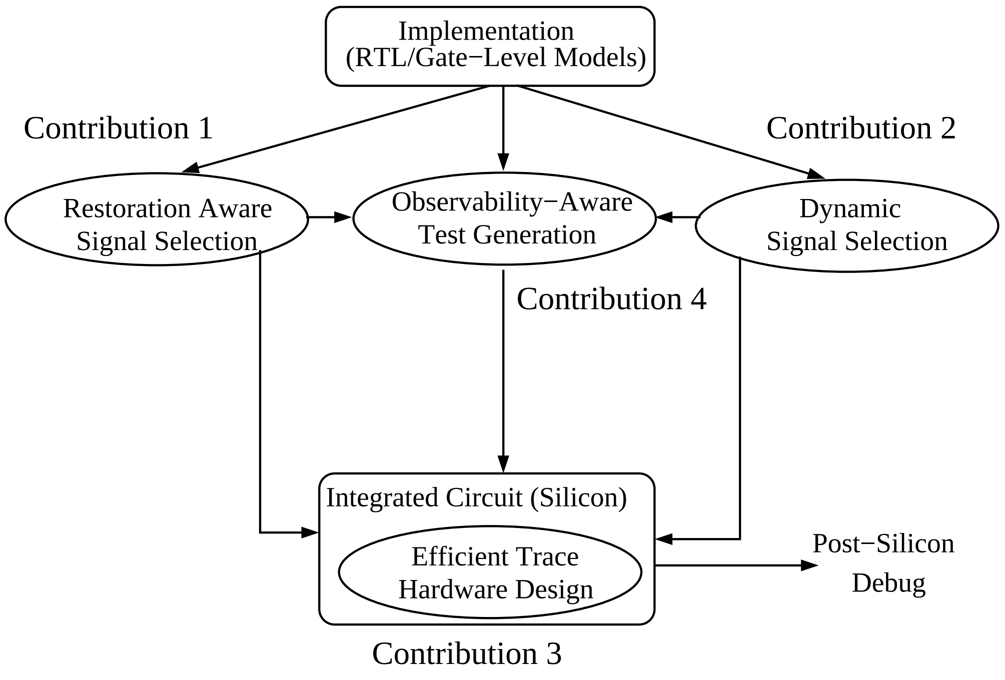

Implementation (RTL/Gate−Level Models)
Contribution 1
Contribution 2
Restoration Aware Signal Selection
Observability−Aware Test Generation
Dynamic Signal Selection
Contribution 4
Integrated Circuit (Silicon)
Efficient Trace Hardware Design
Post−Silicon Debug
Contribution 3

Figure 1-7. Research contributions

1.5 Research Contributions

My research proposes novel techniques to address challenges in enhancing the signal observability for post-silicon validation and debug. The objective of my research is to develop efficient signal selection and test generation as well as adept approaches for trace compression. The four major contributions of my research are summarized as follows. Figure 1-7 highlights these contributions in IC design methodology.

Contribution 1 [Trace Signal Selection]: This contribution addresses the first three challenges outlined in Section 1.4. Existing signal selection algorithms used partial restorability, which is not optimal for signal restoration. A total restorability[1] based signal selection algorithm has been proposed which is computationally more efficient and produces significantly better restoration performance compared to the existing approaches. We have extended the signal selection approach to RTL level, to reduce the signal selection time as well as the memory requirements, without sacrificing significantly on the restoration performance. We have also proposed an efficient technique to determine the profitable combination of trace and scan signals for post-silicon debug. The entire trace buffer width is divided to accommodate both trace and scan signals. Our approach uses a graph based representation to select three important aspects: (i) efficient trace signals to be stored every cycle, (ii) the most profitable scan signals to be included in the shadow scan chain, and (iii) the scan dump frequency based on the trace buffer width constraints.

Contribution 2:[Dynamic Signal Selection]: This contribution addresses Challenge 4. Existing trace signal selection algorithms deal with improving the restoration of untraced signals and does not focus on error detection. We have proposed a signal selection algorithm which selects profitable signals for efficient error detection. Our algorithm lays emphasis on how errors, which propagate from error origin towards signals in

[1] Partial and total restorability have been discussed in Chapter 3.

the fan-out cone can be detected. We have also proposed a region-aware signal selection algorithm (RSS) that selects profitable signals during design time based on the knowledge of functional regions and associated error zones. We have developed a low-overhead dynamic signal tracing (DST) hardware to enable designers to trace different set of signals during execution based on active (relevant) functional regions. This lays emphasis on the active error zones in the circuit that can be detected using a specifically selected set of trace signals. To the best of our knowledge, this is the first attempt in developing an efficient spatio-temporal solution for dynamic trace signal selection.

Contribution 3:[Trace Data Compression]: This contribution addresses Challenge 5. Existing trace compression algorithms choose the dictionary online (thus includes all the unique entries in the dictionary), which results in poor compression performance as well as increased hardware overhead. Studies [14, 15] reveal that the difference between the actual trace data and the ideal (error-less) trace data is very small for post-silicon debug (2-5%). This motivated us to develop a lossless dictionary based width compression scheme that operates on real-time to compress the trace data using a statically selected dictionary. This provides a better compression performance since only profitable entries are stored in the dictionary. This also provides huge reduction in compression hardware overhead. Three different compression algorithms have been proposed to trade-off between compression performance and hardware overhead.

Contribution 4:[Observability-Aware Test Generation]: This contribution addresses Challenge 6. The tests that are used during post-silicon validation are produced using random or constrained-random test generation techniques. However, error detection performance can be enhanced if the tests are designed keeping the detection objective in mind. The tests should be designed to excite the errors (specially corner case ones) and propagate them to the observation points, that is, to the traced signals. Moreover, the trace signals are chosen assuming the input tests are random in nature. However, if the test sets are known a priori, adequate trace signals can be selected to enhance error detection. We propose efficient techniques to determine the trace signals as well as the test sets that would help in detecting errors in the design.

1.6 Dissertation Organization

The dissertation is organized as follows. Chapter 2 presents related approaches for signal selection, trace compression and test generation. Chapter 3 describes our trace signal selection algorithm. Chapter 4 explores an efficient combination of trace and scan signals. Chapter 5 describes our signal selection technique that focuses on error detection. Chapter 6 presents our dynamic signal selection algorithm. A trace compression technique based on static dictionary selection is presented in Chapter 7. Chapter 8 describes observability-aware test generation techniques. Finally, Chapter 9 concludes the dissertation.

CHAPTER 2
BACKGROUND AND RELATED APPROACHES

The focus of this dissertation is to reduce the overall effort of post-silicon validation, which is one of the most critical stages in System-on-Chip (SoC) design methodology. Post-silicon validation and debug comprises of signal observation and analysis as described in Section 1.2. A primary problem for post-silicon debug is the limited observability of internal signal states since the chip has already been fabricated. Once the signal states are known, they can be analyzed using some algorithms like failure propagation tracing [17] to identify the errors in the circuit. Koushanfar et al. [18] proposed a method to obtain the internal states of a system using a "golden cut". However, their method is not applicable for post-silicon debug since it is difficult to stop execution of a process running on a chip and obtain the knowledge of all the current signal states. Formal analysis for post silicon debug proposed by De Paula [19] is not applicable to circuits with a large number of gates. Physical probing techniques were proposed by Nataraj et al. [20]. Decrease in feature size and growing complexity of IC designs have made it diffcult to implement these techniques in practice. A method for validation of memory subsystem in CMPs was proposed by DeOrio [21], which only focuses on the memory subsystem. Scan based debugging techniques such as [11] are not appropriate since they require to stop the circuit functionality when the scan data is being written. This is particularly not beneficial in cases where the functional errors are drastically apart. Double buffering [22] of scan elements helps to mitigate this problem, but with a large area penalty.

Design-for-Debug (DfD) techniques have been used extensively to increase the observability of internal signals of the silicon. Generally this is performed by sampling the data which is stored in on-chip trace buffers. Various DfD techniques like embedded logic analyzer (ELA) [23] and shadow flip flops [22] have been proposed over the years for post-silicon debug. ELA can be used to probe into the chip and record some internal logic states. The trace is then recorded in an on-chip trace buffer. During debug, the contents of trace buffer is transferred to an offline debugger via some Joint Test Action Group (JTAG) interface. In the following subsections, we discuss related approaches in the area of signal selection, trace compression and test generation.

2.1 Trace Signal Selection

Since ELA allows only a few signals to trace, they should be carefully selected in order to enhance the overall observability during post-silicon validation. A logic implication based trace signal selection method was proposed by Prabhakar et al. [24]. The authors used the primary inputs, in addition to the traced signals for restoration purposes. Ko et al. [5] and Liu et al. [6] have proposed generic trace signal selection algorithms in which a few important signals can be traced and others can be reconstructed from them. All these approaches use gate-level netlist model of a design for signal selection purposes. Both space and time complexity can be reduced if the same operation is performed at higher abstraction levels like RTL. However, care should be taken to avoid degradation of restoration performance. Considerable research has been performed over the years involving both gate and RTL-level designs in the fields of testing and validation. An RTL fault grading approach was used to ameliorate the gate-level fault coverage by Mao et al. [25]. RTL-level tests were generated and reused for detecting gate level stuck-at-faults by Yogi et al. [26]. Recently RTL-level signal selection algorithms for post-silicon validation were proposed by Ko et al. [8]. However, [8] selects some signals from the RTL-level and then the rest from the gate-level description of the circuit. Thus, both RTL-level and gate-level models of the design are necessary in [8] to select signals that adds to the memory and time overhead of signal selection.

Scan chains have been used for improving the signal observability during post-silicon debug [9–11]. A combination of scan and trace signals for post silicon debug was first proposed by [12]. In their approach, the trace buffer is used to determine the time window over which the bug might have occurred. The experiment is then re-run with the scan data concentrating on that particular time window. Combination of trace and scan data were also used by [13] for silicon debug. They used multiple runs of the same experiment to obtain the trace data. The scan data were used to select a known state. Both of these approaches assume repeatable experiments i.e., the circuit response is uniform for multiple debug runs. Ko et al. [27] proposed an approach of combining scan and trace data that works well even in non-repeatable experiments. However, their method used an exhaustive exploration of all possible trace-scan combinations to determine the best result for a particular circuit. This exhaustive exploration may not be suitable for practical purposes.

2.2 Dynamic Signal Selection

Existing trace signal selection algorithms statically selected a set of signals that would be traced every cycle. Prabhakar et al. [28] proposed an approach to alternate between two sets of signals in alternate cycles. As a result, it is a very specific case of temporal distribution of errors without any consideration for spatial distribution. A multiplexed signal selection for error detection was proposed by Liu et al. [29]. Their approach is an ad-hoc signal selection heuristic based on error visibility metric. It approach does not consider the challenges associated with dynamic signal selection in the presence of spatial and temporal distribution of errors.

2.3 Trace Data Compression

To increase the amount of data that can be stored in a trace buffer while keeping the trace buffer size constant, trace compression techniques have been proposed [14–16], which compress the trace data before storing them into trace buffer. This enables us to observe more trace data while keeping the trace buffer size constant. The trace buffer has two parameters - width and depth. Width refers to the number of signals whose states can be stored every cycle, while depth refers to the number of cycles over which the trace is stored. Existing trace compression approaches differ in terms of compression objectives - while [16] compresses the width of the trace buffer, [14, 15] compress the depth.

2.4 Observability-Aware Test Generation

Soft errors and crosstalk faults are two major electrical errors found in a fabricated SoC. Effect of soft errors on memory devices had been studied as early as in 1979 by May et al. [30]. Over the years, researchers [31–34] have studied the various aspects of soft errors. Sanyal et al. [35, 36] have proposed different methods for directed test generation for soft errors. However, these approaches are not designed for post-silicon validation purposes, that is, they assume all the output signals of a logic block are visible. However, during post-silicon validation, since the chip is fabricated, observing the output signals of every component may not be feasible since these components can be embedded in an SoC. The only observable points would be the trace signals. The test generation algorithms need to be modified to take this into account.

Crosstalk faults occur when two lines in a circuit are so near that their mutual capacitance affects their state. Effects of crosstalk faults on digital circuits [37–40] have been studied extensively. Existing test generation algorithms for crosstalk faults [41–43] suffer from the same problem as the corresponding test generation algorithms for soft errors - that is, they are not suitable for application in post-silicon validation due to limited observability through trace signals.

CHAPTER 3
RESTORATION-AWARE TRACE SIGNAL SELECTION TECHNIQUES

During post silicon debug, the signal states are stored in an on-chip trace buffer. Limited size of the trace buffer allows only some signals to be traced. The rest of the signals have to be reconstructed from them. Therefore, the signals to be traced should be carefully selected in order to enhance the overall signal restoration across the circuit. Existing signal selection approaches [5, 6], which utilize partial restorability[1] are not able to provide best possible signal reconstruction. We propose total restorability[2] based signal selection algorithms that can outperform existing approaches in both signal selection time and the quality of the selected signals.

Signal selection at higher abstraction levels is promising to reduce the overall signal selection time. Since the number of variables used to represent the registers and other signals is less at RTL-level, the overall complexity is much reduced. We have proposed an efficient signal selection approach in RTL level. Our RTL-level signal selection reduces the signal selection time as well as the memory requirements significantly without significant penalty in restoration performance. Our proposed method has similarity with the signal selection approach developed by [8] since both use a control data flow graph. However, there is a basic difference between the two approaches. While [8] selects some signals from the RTL-level and the rest from the gate-level description, our proposed approach operates only on the RTL-level description. Hence, unlike [8], it does not require the gate-level model of the circuit for signal selection.

[1] Partial Restorability of a signal refers to the probability that the signal value can be reconstructed using known values of some other traced signals

[2] Total Restorability measures whether a group of signals can definitely reconstruct a set of signal states.

3.1 Gate-level Signal Selection (GSS)

Algorithm 1 shows our gate-level signal selection procedure (GSS) that has five important steps. Edge and node values are calculated in the first two steps. Total restorability computation is then used to create region and recompute node values, accompanied by signal selection. The remainder of this section describes each of the steps in detail.

```
Algorithm 1: Gate-level Signal Selection
  Input: Circuit, Trace Buffer
  Output: List of selected signals S (initially empty)
  1: Compute the node values.
  2: Find the state element with highest value and add to S.
  3: Create Initial Region.
  while trace buffer is not full do
      4: Recompute the node values of state elements.
      5: Compute region growth by finding the state element with highest value not
      in S and add to S.
  end
  return S
```

3.1.1 Computation of Edge Values

An edge between two state elements is the path taken to reach an element from another, while passing through a number of combinational gates between them, that is, there cannot be any state elements in between them. The edge may be in the forward or backward direction. In Figure 1-4, an edge between the two flip-flops A and C passes through an OR gate. In a general case, there can be any number and type of combinational gates in an edge. To find the probability that C is influenced by the value at A (which is the value of the edge AC), there can be two cases (independent and dependent) as discussed below:[3]

[3] We are showing calculations for forward restorabilities; however, those for backward restorabilities can be derived in similar lines.

3.1.1.1 Independent signals

Consider two edges AC and BC in Figure 1-4. Here, the two input signals of the OR gate in front of flip-flop C are driven by flip-flops A and B, which are independent. Hence, the edges AC and BC are independent. To calculate the edge values for an independent scenario, we use a generic example in Figure 3-1. Later, we will show how the calculation works for the specific case in Figure 1-4.

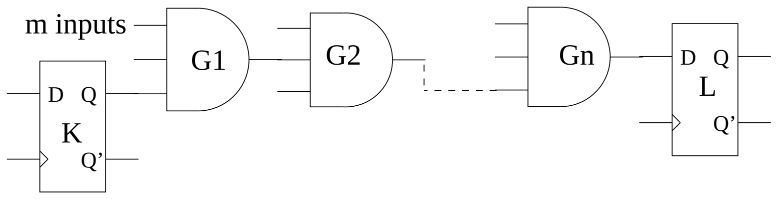


Figure 3-1. Example circuit with n gates

Figure 3-1 has two flip-flops K and L. We want to find how the input of L is sensitized by the output of K. The input of L corresponds to the output of the gate G_n. The path from K to L is independent of any other paths through which the output of K propagates. Let's consider the gate G_1. We define four probabilities: ${P^I}_{0,N}$, ${P^I}_{1,N}$, ${P^O}_{0,N}$ and ${P^O}_{1,N}$. Here, ${P^I}_{0,N}$ indicates the probability that a node n (gate or flip-flop) has an input value of '0' when another node is controlling it. Similarly, ${P^I}_{1,N}$, ${P^O}_{0,N}$ and ${P^O}_{1,N}$ indicate the cases for input value of '1', output value of '0' and '1', respectively. The output of flip-flop K can influence the output of G_1 in two cases: i) output of K is a controlling value, and ii) all the inputs to G_1 are complement of the controlling value. Let us consider G_1 to be a 2-input AND gate. We define P_{G_1} as the overall probability of K controlling G_1. According to [44],

$$P_{G_1} = {P^O}_{1,G_1} + {P^O}_{0,G_1} \tag{3–1}$$

Now, let's define ${P^O}_{0,G_1}$ and ${P^O}_{1,G_1}$. Let P_{cond0,G_1} and P_{cond1,G_1} be the probability that the output of G_1 follows the output of K, i.e., the output of G_1 is 0(1), when the output of K is 0(1). For simplicity of calculation, in this example, we assume ${P^I}_{0,G_1}$ = ${P^I}_{1,G_1}$= 0.5

(that is, occurrence of 0 or 1 follows equal probability at the input).

$$P^{O}{}_{0/1,G_1} = P_{cond0/1,G_1} \times P^{I}{}_{0/1,G_1} \tag{3–2}$$

Now, for a 2-input AND gate, P_{cond0,G_1} is 1, since 0 is the controlling input. Therefore, we obtain $P^{O}{}_{0,G_1} = 0.5$. Similarly, since 1 is the non-controlling input, P_{cond1,G_1} is 0.5, which gives $P^{O}{}_{1,G_1} = 0.25$. From Equation 3–1, it can be seen that $P_{G1} = 0.75$. Now, we return to our main goal, that is, to determine how K controls L. We first find the effect of the output from K as it propagates to the next gate G_2 and then extrapolate along the entire path to L. We use the same set of Equations 3–1 and 3–2 again, except that the input is G_1 here and the output is G_2. Obviously, the values of $P^{I}{}_{0,G_2}$ and $P^{I}{}_{1,G_2}$ would be $P^{O}{}_{0,G_1}$ and $P^{O}{}_{1,G_1}$ obtained from Equation 3–2. For example if G_2 is also a 2-input AND gate, applying Equation 3–2, we obtain, $P^{O}{}_{0,G_2}$ = 0.5, and $P^{O}{}_{1,G_2}$= 0.125. Therefore, we get P_{G2} = 0.625, where P_{G_2} is the probability for the gate G_2 defined in Equation 3–1.

In this way, the calculation continues until we reach L, to obtain the value of the edge KL. If there are n combinational gates between K and L, we get

$$P^{O}{}_{0/1,G_n} = \prod_{1 \le i \le n} (P_{cond0/1,G_i}) \times P^{I}{}_{0/1,G_1} \tag{3–3}$$

Finally, Equation 3–1 is used to compute the probability P_{G_n}, which corresponds to the value of the edge KL.

We use these computations to show how an edge value is computed in case of the circuit in Figure 1-4. Let's compute the value of edge AC. We name the OR gate in between the two as gate G and we assume that $P^{I}{}_{0,G}$ = $P^{I}{}_{1,G}$ = 0.5. Since it is an OR gate, $P_{cond0,G}$=0.5 and $P_{cond1,G}$=1. Therefore, Equation 3–2 can be used to obtain $P^{O}{}_{0,G}$=0.25 and $P^{O}{}_{1,G}$=0.5. Equation 3–1 can now be used to obtain P_G=0.75, which represents the value of the edge AC.

3.1.1.2 Dependent signals

In case of dependent signals, we need to determine the probability of a state element output influencing an m-input gate, when the output of the state element affects l inputs ($l \geq 2$) of the gate. We have used a generic example in Figure 3-2 to calculate the edge value in case of dependent signals. It should be noted that dependent signals were not considered by [5] or [6].

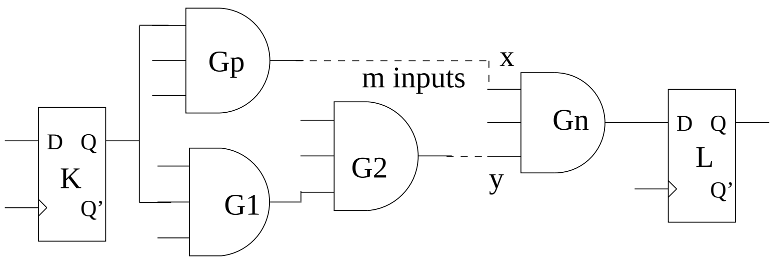


Figure 3-2. Example circuit

Let's consider Figure 3-2. It can be seen that two inputs (x, y) of the m input gate G_n are affected by flip-flop K. For this, our goal would be to combine the dependent edges so that the edge will have independent signals. We can then easily utilize the formula used in Section 3.1.1.1 to compute the edge value. We desire to find $P^O{}_{1,G_n}$ and $P^O{}_{0,G_n}$, in lines with the parameter $P^{I/O}{}_{0/1,N}$ defined in Section 3.1.1.1. Let us assume that G_n is an AND gate. For an AND gate, since 0 is the controlling value, having either of the inputs as 0 will ensure a 0 being propagated into the gate G_n. Therefore

$$P^I{}_{0,G_n} = P^O{}_{0,x} + P^O{}_{0,y} - P^O{}_{0,x\&y} \tag{3–4}$$

$P^O{}_{0,x\&y}$ subtracts the probability when both are 0, since it is being computed twice. Similarly, since 1 is the non-controlling input, we get

$$P^I{}_{1,G_n} = P^O{}_{1,x\&y} \tag{3–5}$$

where $P^O{}_{1,x\&y}$ is the probability when both x and y are '1'. Let's evaluate the terms $P^O{}_{0,x\&y}$ and $P^O{}_{1,x\&y}$. Let $P_{cond0/1,x/y}$ be the probabilities that $x(y)$ is $0(1)$ when the output of K is $0(1)$. ${P^O}_{0/1,x\&y}$ can be defined as

$${P^O}_{0/1,x\&,y} = (P_{cond0/1,x} \times P_{cond0/1,y}) \times P^O_{0/1,K}$$

With the help of Equation 3–2, this can be reduced to

$${P^O}_{0/1,x\&y} = \frac{{P^O}_{0/1,x} \times {P^O}_{0/1,y}}{{P^O}_{0/1,K}} \tag{3–6}$$

Since the paths from K to x and from K to y are assumed to be independent[4] , Equation 3–3 can be used to obtain the values ${P^O}_{0/1,x/y}$. Application of Equations 3–4 and 3–5 provide the values of ${P^I}_{0/1,G_n}$. The final P_{G_n} can be obtained using Equations 3–1 and 3–2, and the information on the number of inputs to the gate G_n. This corresponds to the value of the edge KL.

3.1.1.3 Example

We now proceed to show how the calculations described in Section 3.1.1.1 and Section 3.1.1.2 can be used to determine the edge values for the circuit in Figure 1-4. A graphical representation of the circuit is shown in Figure 3-3.

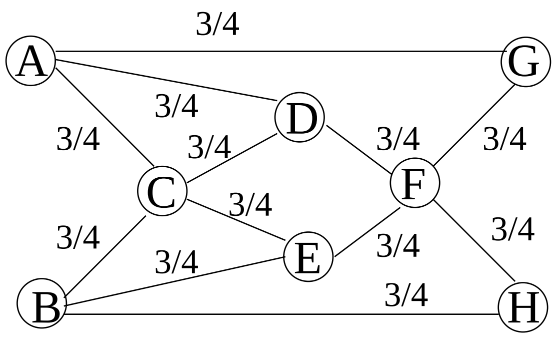


Figure 3-3. Graphical representation of example circuit

The state elements are represented by nodes and an edge between two state elements is represented by a straight line. It should be noted that there are no

[4] If any one of these paths consist of dependent signals, the above procedure can be applied in a recursive manner until it becomes an equivalent independent path.

dependent edges in this example. All the edges have one two-input gate in between them, As a result, all the edge values are $\frac{3}{4}$ (obtained from Section 3.1.1.1). We will use this graph to explain our signal selection algorithm.

3.1.2 Initial Value Computation for State Elements

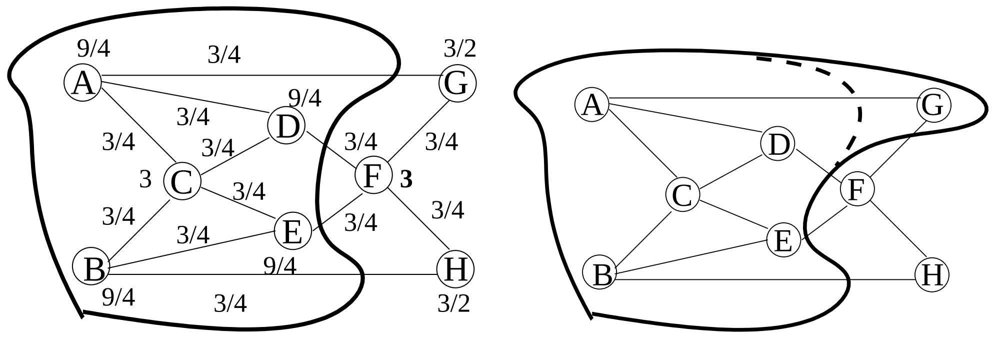
We define the value of a state element as the sum of all the edges attached with it, in both forward and backward direction. For example, in Figure 3-3, the value of flip-flop c is the sum of the weights of all edges connected with it, that is, CA, CB, CD and CE. It is important to note that we have used a "threshold" in order to prevent combinational loops inside the circuit during edge value computation. This parameter was used by [5] as well.

Our computation of the state element values are independent of the sequential loops in the circuit. In a sequential loop, the output of a state element depends on another in both the previous and the next cycle. However, both cannot be true at the same clock cycle; that is, the same state element cannot determine the output of another in the same cycle by both forward and backward restoration. While forward restoration can determine the state in at least the next cycle, backward can determine it at most the previous cycle.

3.1.3 Initial Region Creation

A region is a collection of state elements attached together. It is not necessary that all the state elements have an edge with each other in the region. However, each state element in the region must have at least one edge with another state element in the region. In Figure 3-3, the flip-flops A, B, C, D and E form a region. The first state element to be chosen is the one with the highest value, based on the calculations in Section 3.1.2. It is added to a list called "known". Now, all state elements which have an edge with the recently selected element are added to the region.

We show by an example in Figure 3-4A how this portion of our algorithm is used to perform the selection of the profitable signals. The edge values are shown along each



A Initial region creation B Region Growth

Figure 3-4. Region creation and growth

edge. The values of the flip-flops (addition of all it's edge values) are shown in bold alongside each flip-flop. For example, a has 3 edges AC, AD and AG, each having a value $\frac{3}{4}$. Therefore, the value for a is $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} = \frac{9}{4}$. The flip flop with the highest value in Figure 3-4A is C. All the nodes which have an edge from C are included in the region. The region is represented by the spline in Figure 3-4A.

3.1.4 Recomputation of Node Values

The first state element in Figure 3-3 to be traced is already known (c in the previous example). However, there are other state elements that need to be traced as well. To select the subsequent state elements, their values are recomputed. The state element whose value is being computed may have an edge to an element inside the region as well as one outside the region. Edges to state elements inside the region are given higher weight. As discussed before, many restorability computations require knowledge of more than one signal of the input/output[5] . Therefore, it is better to gain more knowledge of the signals that are already in the region, thus increasing their restorability values and therefore, aiming for total restorability of those signals. Existing approaches [5] and [6] recompute the restorability values after each iteration,

[5] For example, when all the inputs to a gate are complement of the controlling value.

which when translated to the graph in Figure 3-3, would correspond to edge value recomputation. Clearly, this is more computationally intensive.

3.1.5 Region Growth

The state element with the highest restorability and not in the list "known" is determined. If two state elements have the same value, the one with the higher forward restoration is traced. This is because, backward restoration fails in some cases whereas forward restoration does not when all the inputs are known. For example in Figure 3-4A, the next state element to be traced is A. It is included in the list "known". If the trace buffer is already full, calculations will stop, otherwise the region is continued to grow. All state elements having an edge to the recently selected node are added in the region. As shown in Figure 3-4B, in this case G is added since G is the only node connected to a and not in the region. The dotted line indicates the original region. Next, recomputation of state element values as in Section 3.1.4 is reconsidered and this process is iterated until the trace buffer is full.

3.1.6 Complexity Analysis

In this section, we compute the complexity of our algorithm. Let V be the number of state elements in the circuit and E be the number of edges in the circuit. Let n be the number of signals to be traced, that is, the size of the trace buffer is n. The first step, that is, edge value computation takes $O(E)$ time, while flip-flop value computations for each time a signal is selected takes $O(V)$ time. To select n signals, the time required is $O(NV)$. Therefore, the overall time complexity of our algorithm is $O(E+NV)$. On the other hand, the time complexity of existing algorithms is $O(NE)$. Since, $E >> V$, the time complexity of our proposed algorithm is less. The overall space complexity of our algorithm is $O(E+N+V)$. Since, $E >> N+V$, the space complexity reduces to $O(E)$.

3.2 Motivational Example

We now employ our proposed method (described in Section 3.1) for selecting signals in the circuit in Figure 1-4. The first signal that we trace is C. Note that this was the same signal that was chosen by [5] and [6]. The second signal that we choose is A, based on total restorability computations. Tracing A along with C guarantees to reconstruct D every cycle. As indicated earlier, existing methods select F as the second traced signal. Clearly, C and F together do not provide any such guarantees. The results are shown in Table 3-1. It can be seen that our method provides a restoration ratio of 3.2, which is better than the one provided by [5] and [6].

Table 3-1. Restored signals using our method

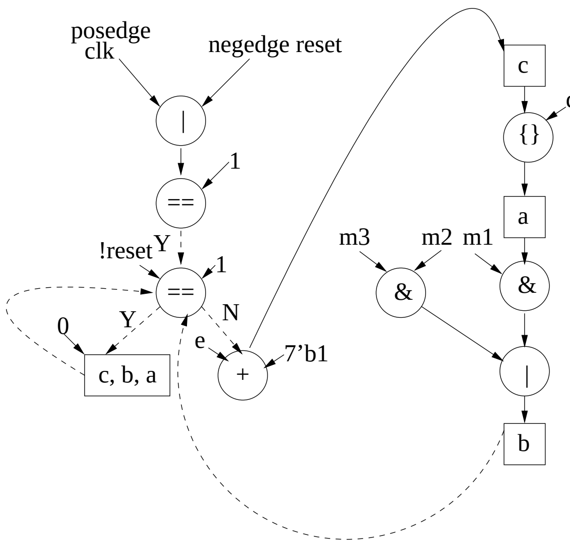
Signal	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
A	0	0	0	0	1
B	1	0	1	0	X
C	1	1	0	1	0
D	X	0	0	0	0
E	X	1	0	0	0
F	X	X	1	0	0
G	X	0	0	0	0
H	X	X	0	0	0

3.3 RTL-level Signal Selection (RSS)

To show how signal reconstruction can be efficiently performed in RTL level, let's consider the Verilog design in Figure 3-5A. The design consists of three register-variables namely a, b and c (each correspond to a set of flip-flops) as well as two input signals d and e. There are also three other signals $m1$, $m2$ and $m3$. In the example, a and b are 8 bits long, c and e are of 7 bits, while d is just a one-bit signal. To show how reconstruction is performed, let's observe how each of these flip-flops are assigned - a is the concatenated value of d and c; b is the result of logical operations between a, $m1$, $m2$ and $m3$ while c attains the sum of an arithmetic operation between e and a constant number. Let's assume that we trace the state of a. We now explain how tracing of a in cycle k helps us to reconstruct the other states. The assignment of b shows that the state of b in cycle $k+1$ can be reconstructed from the state of a by forward restoration. From the assignment of a, the states of c and d in cycle $k-1$ can be reconstructed from state of a by backward restoration. Finally, from the last statement, that is, the assignment of c, state of e can be restored in cycle $k-2$ by backward restoration. Thus, we see that tracing of only one state of a can reconstruct the states of 4 other variables in different cycles.

```
always @ (posedge clk or negedge reset)
begin
if  (!reset)
  begin
    a <= 7'b0;    b <= 7'b0; c <= 7'b0;
end
else
  begin
   a <= {d, c}; b <= (a & m1) || (m2 &
m3); c <= e + 7'b1;
  end
end
```

A RTL Verilog example



B CDFG of Verilog code

Figure 3-5. Verilog code and CDFG

Algorithm 2 shows our signal selection procedure that has six important steps. In the first step, a control data flow graph (CDFG) is generated to model the entire system. Since in the RTL-level, each register variable represents multiple state elements, we use the register variables for signal selection purpose. However, the trace buffer width refers to the total number of state elements represented by these register variables. For example, the register variable $[7:0]a$ will represent 8 state elements, and therefore selection of variable a implies that 8 trace buffer locations are needed. The relationship between the different register variables is obtained from the CDFG. These relations are used to produce the total restorability values for the variables. The register variable with the highest value is chosen for tracing. Once a variable is chosen for tracing, all the other variable values are recomputed in the same manner as in Algorithm 1. The steps

$4-6$ are continued until the trace buffer is full. The remainder of this section describes each of the steps in detail.

```
Algorithm 2: RTL-level Signal Selection
    Input: RTL description of design, No. of trace entries
    Output: List of selected signals S (initially empty)
    1: Develop the CDFG of the RTL description.
    2: Find the  relationship between the register variables.
    3: Find the initial values of the register variables.
    while trace buffer is not full do
        4: Find the register variable with the highest value.
        5: Add all corresponding state elements to the list S.
        6: Recompute values for all the register variables.
    end
    return S
```

3.3.1 CDFG Generation

The first step of RTL level signal selection is to generate the Control Data Flow Graph (CDFG) from the RTL model. CDFG can be generated using any standard HDL parser. For our use, we have generated the CDFG by modifying the open source Icarus Verilog parser [45] for the Verilog circuits. Although, our studies are based on Verilog benchmarks; our approach is also applicable for VHDL designs. The format of our CDFG representation is similar to Mohanty et al. [46].

Figure 3-5B shows the CDFG representation of the Verilog code in Figure 3-5A. The CDFG can represent both the movement of control signals as well as data values. The dotted arrows indicate the control-flow (transitions) in the CDFG, while the bold arrows represent the data flow (computations). For example, in the right hand side of Figure 3-5B, there is a bold arrow from a to the AND gate. This is because a is an input of the AND gate. The circles in the CDFG represent operational and control nodes, while the boxes represent storage nodes. For example, the circle in the top represents an OR assignment for the conditional statement always, while the square at the bottom in right hand side of Figure 3-5B represents the storage in the node b. It should be noted that direct assignments like $a <= 7b0$ are just represented as a bold arrow with value 0 entering a box for storage of value a. In this case, since three variables a, b and c are all being assigned 0 together, they are grouped in a single box.

This basic representation can be further extended to represent the CDFG of a complex design. This CDFG representation is used as input to the next step, relationship computation.

3.3.2 Relationship Computation

The relationship of a signal with others can be obtained from the CDFG. The relationship computation for a single signal in the circuit provide the effect of that signal on others. To compute the relationship of the signals, we first note that there can be two main relationship types, namely direct relationship and conditional relationship. These two classes and their respective relationship computations are explained as follows.

3.3.2.1 Direct relationship

Two signals are said to be directly dependent when they occur on the same line of a signal assignment. For example, in the RTL description shown in Figure 3-5A, the signal pairs (a, b) and (a, c) have direct relationship. This is because both the variable assignments occur inside the "if'" block. Direct relationship can be of two types, namely forward and backward relationship. Forward relationship deal with the propagation of values in the forward direction, that is from the right hand side of the assignment to the left hand side. Backward relationship on the other hand deals with the reverse, that is from left hand to right hand side of the assignment. For example, in the RTL description of the example in Figure 3-5B, a has a forward relationship on b, while b has a backward relationship on a. We will use a simple generic example to show how the direct relationship computation is performed. Later, we will consider a specific example shown in Figure 3-5B. A typical signal assignment statement looks like

$$y <= x_1 \; OP_1 \; x_2 \; OP_2 \; x_3 \; OP_3 \; .... \; x_n$$

where OP represents any operation (eg., AND, OR, etc.). We can see that there are n signals on the right hand side of the assignment statement. We want to find out the relationship of each of these signals on y. Let us assume that each of these signals are k bits long and all the x_i's are independent. We also assume that each of the OP_i's are AND gates. Therefore, the assignment statement can be rewritten as

$$y <= x_1 \ \& \ x_2 \ \&..........\& \ x_n$$

The same computations can be extended to other operations, as well as different operations for each OP_i. Let's compute the relationship of x_i $(1 < i < n)$ on y. The relationship of y on x_i is computed as a probability that y completely follows x_i. Therefore, the relationship can be found in the same way as the independent edge value computation in Section 3.1.1.1. Essentially, the relationship of y and x, ${P^y}_{0/1,x_i}$ is equivalent to $P_{cond0/1,y}$ in Equation 3–2. It should be noted that y follows x_i completely when either all the k bits of x_i are 0 (P_0), or when all the x_i's have their k bits as 1 (P_1)(Since 0 is the controlling input of AND gate). In all other cases, some of the k bits of y are different from x_i. The relationship of y on x_i is given by

$${P^y}_{0,x_i} = \frac{2^{k\times(n-1)}}{2^{k\times n}} \tag{3–7}$$

$${P^y}_{1,x_i} = \frac{1}{2^{k\times n}} \tag{3–8}$$

According to [44] and assuming for simplicity that 0 or 1 can occur with equal probability, we get,

$${P^y}_{x_i} = \frac{2^{k\times(n-1)}+1}{2^{k\times n}} \tag{3–9}$$

The numerator on the right hand side consists of two terms. The first term corresponds to the case when all the bits of x_i are 0. Therefore, each of the k bits of the other $n-1$ variables can have $2^{k\times(n-1)}$ values. The last term, 1 on the numerator, corresponds to the case when all the bits of all the x_i's are 1. The denominator denotes all the possible cases of value assignments to the x's. These calculations, as stated above, can be extended for other operations as well. For example, if OP was an OR operation, Equations 3–7 and 3–8 will be modified as

$$P^{y}{}_{1,x_i} = \frac{1}{2^{k \times n}} \tag{3–10}$$

$$P^{y}{}_{0,x_i} = \frac{2^{k \times (n-1)}}{2^{k \times n}} \tag{3–11}$$

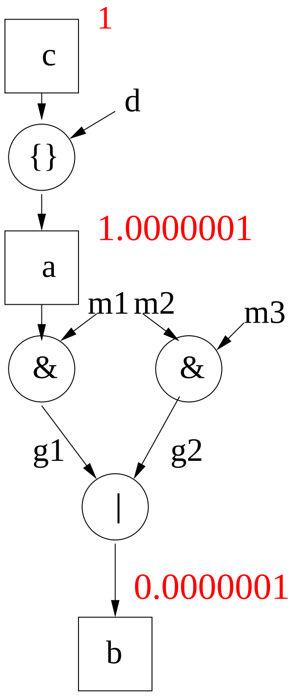


Figure 3-6. A portion of the CDFG in Figure 3-5B

We now apply these computations to the specific example in Figure 3-5B. Figure 3-6 shows a portion of Figure 3-5B that shows dependency of b and c on a. We assume that all the signals are independent here. Two new variables $g1$ and $g2$ are introduced for the ease of illustration. Clearly, we have

$$g1 = a \;\&\; m1$$

$$g2 = m2 \;\&\; m3$$

$$b = g1 \mid g2$$

To find the relationship of a on b, we first find the relationship of a on $g1$, and then $g1$ on b. It can be seen from Equation 3–7 and 3–8,

$$P^{g1}{}_{0,a} = \frac{2^8}{2^{16}}$$

$$P^{g1}{}_{1,a} = \frac{1}{2^{16}}$$

To find the second part, that is the relationship of $g1$ on b, we use Equations 3–10 and 3–11,

$$P^{b}{}_{1,g1} = \frac{1}{2^{16}}$$

$$P^{b}{}_{0,g1} = \frac{2^8}{2^{16}}$$

Combining these two sets of equations, we get the relationship of a on b as

$$P^{b}{}_{1,a} = \frac{1}{2^{16}} \times \frac{2^8}{2^{16}}$$

$$P^{b}{}_{0,a} = \frac{1}{2^{16}} \times \frac{2^8}{2^{16}}$$

Finally, using Equation 3–3, we get,

$$P^{b}{}_{a} = \frac{1}{2^{23}} = 0.0000001$$

Since c and a have a direct concatenation relationship, the value of $P^{a}{}_{c}$ is obtained as 1.0. Since there is no direct assignment relationship between b and c, there is no edge between them. Thus, we obtain the values of edges *ba* and *ac* of Figure 3-6. Figure 3-7 shows a simplified version of Figure 3-6 with the edge values (shown below the edges). The node values are computed by adding the edge values. For example, the node *c* is connected to only one edge with value 1.0, therefore, the node value of c is 1.0. Similarly, the node value for *a* is 1.0 + 0.0000001 = 1.0000001. Likewise, the node value for *b* is 0.0000001. These node values are shown in the figure (on top of nodes).

In this section, we have described the direct relationship when the signals are independent. However, similar to Section 3.1.1.2, we can have dependent signals as

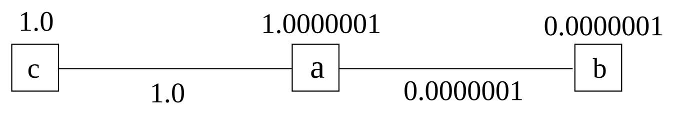


Figure 3-7. Simplified version of Figure 3-6

well. The nature of dependent signals are derived from multiple branches of the CDFG. Computations for dependent signals are similar to the computations in Section 3.1.1.2.

3.3.2.2 Conditional relationship

Conditional relationship corresponds to the non-assignment dependencies. For example, in the RTL code corresponding to Figure 3-5B, the signals a and b have conditional relationship on reset[6] . We generally do not consider backward conditional relationship, since, these are not in direct assignment statements. Conditional relationship are computed in the same way as in Equation 3–7, however the operations are checked inside the conditional block. For example, we consider the following code:

$$if(m\ or\ n)\ \ x <= y;$$

Here, the symbol x has a conditional dependence on m and n. Since there are only two variables m and n in the conditional dependency; the dependency value is $\frac{3}{4}$, as obtained from Equations 3–1 and 3–2 in Section 3.1.1.1. The conditional relationships are computed in this manner for all the signals in the circuit.

3.3.3 Signal Selection

Once the values of the variables are computed, the next step is to select the best one for tracing. The signal selection procedure is similar to the gate level signal selection. The variable with the highest value is selected and the rest of the values are recomputed using region growth. This part is same as in Algorithm 1 and hence not

[6] It should be noted that we do not consider conditional relationship of general control signals like clock (clk) or reset.

discussed here. In Figure 3-7, register variable a having the highest value is chosen for tracing. The process continues until the trace buffer is full.

3.4 Experiments

In this section, we first compare our approach with existing gate-level signal selection techniques [5, 6]. Next, we demonstrate how our proposed RTL-level signal selection can further improve signal selection time.

3.4.1 Experimental Setup

We applied our gate-level signal selection approach (GSS) on the ISCAS'89 benchmarks used by [5] and [6] to compare with their methods and hence show the effectiveness of our algorithm. We have designed a simulator in the lines of the one described by [6] for our purpose. We have implemented the simulator as an iterative process which terminates when it is not possible to restore any more states. We have fed the simulator with 10 sets of random values and noted the average restoration ratio.

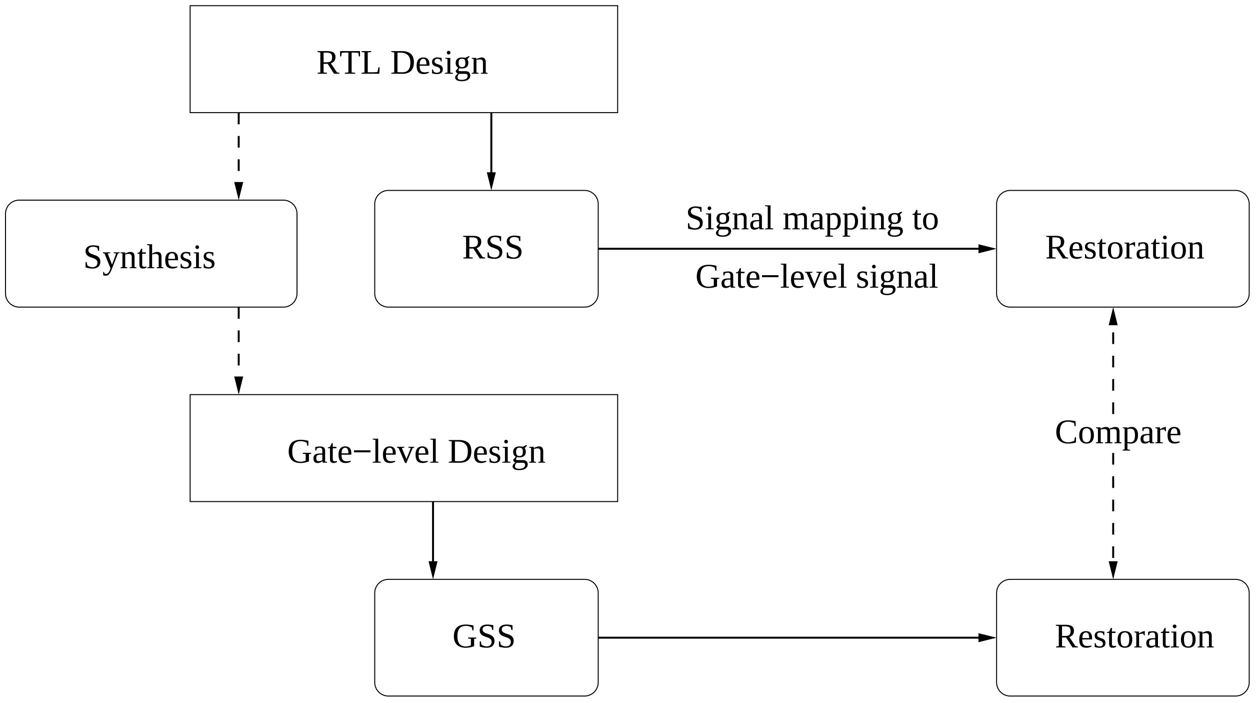


Figure 3-8. Overview of our experiments to verify RSS

Figure 3-8 gives an overview of our experimental setup to verify the RTL level signal selection algorithm (RSS). For this purpose, we have used Verilog circuits obtained from Opencores website [47]. It should be noted that we have not used the ISCAS ’89 benchmarks since an RTL description of these were not available. We have modified the Icarus Verilog parser [45] to generate the Control Data Flow Graph (CDFG). The CDFG is then parsed by another program, which provides the list of selected signals using Algorithm 2. As can be seen in Figure 3-8, we have compared the results obtained using GSS and RSS on the same circuits to compare the restoration performance of each approach. The signals selected using RSS are mapped to gate-level and the restoration performance is noted. Simultaneously, the RTL design is synthesized to gate-level netlist[7] , and GSS is applied on the netlist. A comparison of the restoration performance using the two algorithms reveal that they are almost same as discussed in Section 3.4.3. Thus, our RTL-level signal selection algorithm does not incur any significant restoration penalty compared to the gate-level signal selection algorithm.

3.4.2 Results on Gate-level Signal Selection (GSS)

Table 3-2. Comparison with Ko et al.

	Restoration Ratio with random inputs			Restoration Ratio with deterministic inputs		
Circuit	Ko et al.	Our approach	Impro-vement	Ko et al.	Our approach	Impro-vement
s38584	38	42	1.1	6	20	3.33
s38417	9	16	1.8	9	16	1.8
s35932	48	50	1.04	25	35	1.4

We would like to compare our signal selection approach with the other closely related methods. Table 3-2 compares the performance of our approach with the one proposed by Ko et al. [5] using the three largest ISCAS ’89 benchmark circuits. All the

[7] any synthesis tool can be used for this purpose; we have used Synopsys Design Compiler

experiments have been performed with a trace buffer of width 32. Table 3-2 is divided into three distinct parts. The first column indicates the circuit name. The next three columns compare the performance when random sets of inputs are used to drive the circuits. In this case, even the control signals are driven using random inputs. As a result, the circuit might fall into one of the reset states. The improvement can be defined as the ratio between the restoration ratio using our approach and that of [5]. The third part of Table 3-2 compares our approach with [5] when the gates of the circuit are driven deterministically. This means that the control signals are driven using values that prevent it from going to a reset state, while the other signals are driven with random inputs. From Table 3-2, it can be seen that the improvement obtained using random inputs is moderate (31% on average). On the other hand, considerable gain (117% on average) is obtained when we use our algorithm for deterministic inputs. As discussed earlier, random inputs to control signals might lead to reset states, which are responsible for high restoration for both the approaches. Therefore, improvement obtained is less in this case. As stated in [6] deterministic inputs are actually used in circuits during real-life applications. Hence, gain obtained with them are more significant.

Table 3-3 compares the restoration ratio of our proposed approach with the one proposed by Liu et al. [6] for the three largest ISCAS'89 benchmarks. As before, a trace buffer width of 32 is used. In this case, the inputs are deterministic in nature. An average improvement of 65% is observed. It can be seen that the improvement here is less than the one obtained in Table 3-2. This can be attributed to the fact that the algorithm proposed by [6] is more efficient than [5].

Table 3-3. Comparison with Liu et al. with deterministic inputs

		Restoration Ratio	
Circuit	Liu et al.	Our approach	Improvement
s38584	9	20	2.22
s38417	14	16	1.14
s35932	22	35	1.6

We now compare our approach with the one proposed by Prabhakar et al. [24]. [24] have used the primary inputs along with the traced signals for signal restoration. Till now, we only used the trace signals to restore the rest of the signals on the chip. However, to enable fair comparison, we have included the primary inputs for our signal restoration. The results are shown in Table 3-4 using a trace buffer of width 32. It should be noted that the improvements are moderate (on an average 10%) in this case. When we use the primary inputs for restoration, most of the states at later clock cycles can be recovered. On the other hand, the states where the input test vectors cannot reach due to state depth in early cycles can be restored using the traced data. As reported in [24], about 90-95% of the states were restored using their method. Hence, the scope for improvement is limited.

Table 3-4. Comparison of GSS with Prabhakar et al.

	Restoration Ratio		
Circuit	Prabhakar et al.	Our approach	Improvement
s5378	4.84	5.0	1.03
s9234	5.2	6.0	1.15
s15850	13.9	15.8	1.14
s38584	34.8	40.5	1.16
s35932	52.4	53.3	1.02

Figure 3-9 compares our signal selection time against the time taken by Ko et al. [5] and Liu et al. [6] for the three largest ISCAS'89 benchmark circuits. The X-axis denotes the different trace buffer widths. It can be seen that our approach takes significantly less time (up to 90%) compared to them. This is primarily due to the fact that [5] and [6] recomputes edge values in every iteration whereas we only recompute the node values.

Although our gate-level signal selection algorithm provides significant improvement over the existing approaches ([5] and [6]), it does not guarantee the maximum restoration possible using the same trace buffer size. For example, if we consider a trace buffer of width 32, *GSS* does not guarantee to choose the best 32 signals that can

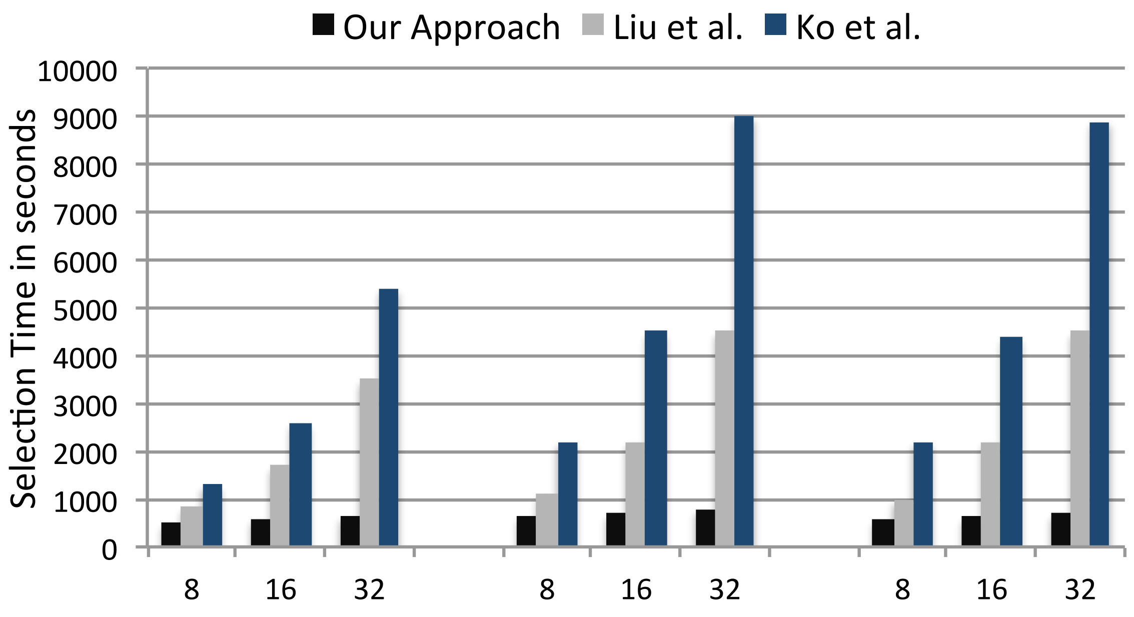


Figure 3-9. Comparison of Signal Selection Time

provide the maximum restoration possible. A detailed analysis is needed to determine the maximum restoration possible using a particular trace buffer size and the signals to be traced in order to obtain that restoration performance.

3.4.3 Results on RTL-level Signal Selection (RSS)

In this section, we discuss how our RTL level signal selection algorithm can further improve the signal selection time without compromising the restoration ratio significantly. As discussed before, we have applied our approach on the designs obtained from the Opencores benchmarks. We have compared our RSS approach with the gate-level signal selection procedure (GSS). The results are shown in Table 3-5. Similar to the previous experiments, we have assumed a trace buffer of width 32.

Table 3-5. RTL-level versus gate-level signal selection

Circuit	Memory Size Reduction	Speedup
Total CPU	8.1	697
Wishbourne LCD controller	22.81	1923
dmx512 tranceiever	191.24	733
OPB onewire	3.22	3600
Simple RS232 Uart	3.8	500

The first column in Table 3-5 provides the circuit name. The second column shows the memory size reduction which is the ratio of memory size in gate-level and RTL-level. The last column gives the speedup obtained using RSS compared to GSS. Speedup can be defined as the ratio of gate-level signal selection time to RTL-level signal selection time. As can be seen, RSS is up to 3600 times faster and requires up to 191 times less memory compared to GSS.

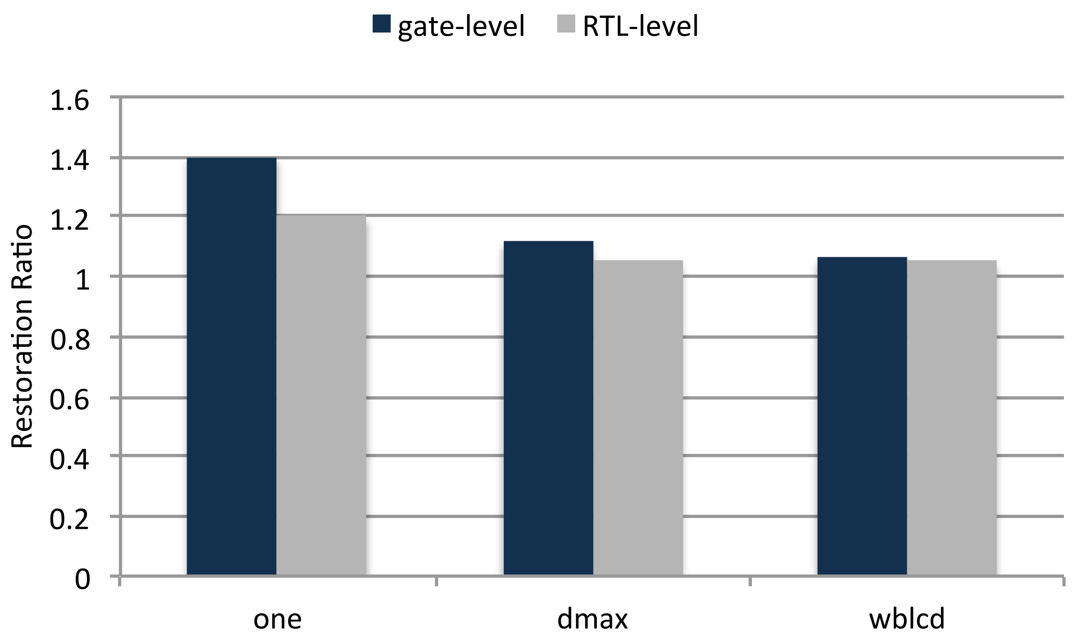


Figure 3-10. Comparison of Restoration Performance

Finally, we would like to compare the restoration performance of RSS and GSS using three Opencore benchmarks (OPB onewire, dmx512 transceiver and Wishbourne LCD controller). The results are shown in Figure 3-10 when the benchmarks are driven using deterministic inputs. As we can see in Figure 3-10, the restoration performance for gate-level and RTL-level are similar. The gate-level restoration performance is found to be slightly better than the RTL-level in some cases. The primary reason for this is the representation of state elements as arrays in RTL-level. Whenever we select a signal for tracing, we are actually tracing all the elements in the array. However, all of the signals in the array may not be equally beneficial. Some other signals could have been selected for better restoration performance.

It should be noted that our proposed trace signal selection algorithm has a high temporal observability (since the signals are traced every cycle) but a low spatial observability. If the circuit does not have many dominating signals or if the circuit is such that the overall restoration capacity of the trace signals are low, tracing a small set of signals would not help in restoring many of the untraced signal states. A possible option to improve observability is to increase the set of trace signals. However, this would mean an increase in trace buffer size which definitely adds to the debug overhead. In order to trace more signals while keeping the trace buffer size same, a possible alternative is to compromise on temporal observability but improve spatial observability. This alternative has been explained in Chapter 4.

3.5 Summary

Effeicient signal selection is important to enhance observability during post-silicon debug. We developed techniques to employ total restorability for selecting the most profitable signals that can provide better restoration compared to when signals are selected using partial restorability equations. We observed the performance of our gate-level and RTL-level signal selection algorithms using ISCAS'89 and Opencores benchmarks. Our experimental results demonstrated two major advantages - our approach can provide faster (up to 90%) signal selection as well as significantly better (up to 3 times) restoration ratio compared to existing approaches. Our RTL-level signal selection approach can further improve signal selection time by several orders-of-magnitude and also requires less memory compared to the gate-level signal selection algorithms.

CHAPTER 4
EFFICIENT COMBINATION OF TRACE AND SCAN SIGNALS

The trace signal selection algorithms described in Chapter 3 lack spatial observability since a small set of signals is being traced every cycle. To improve the spatial observability during post-silicon debug while keeping the trace buffer length fixed, scan data can be combined with trace signals. Recently, Ko et al. [27] have shown the importance of combining scan chains and trace signals. They use a part of the trace buffer input bandwidth to store selected trace signals every cycle. The remaining input bandwidth is used to dump the scan signals at a certain frequency. Although this approach produced promising results, there are several challenges to make it useful in practice. One major issue is that it used exhaustive exploration to determine the profitable combination of trace and scan signals. Such an exhaustive exploration can be infeasible for real designs. Another major concern is that the selected scan signals include almost all the flip-flops. Such an approach is neither practical nor profitable in many real scenarios, since a huge number of shadow flip-flops will be necessary. Also, the time for scan dump will increase, thus effectively decreasing the number of scan dumps (only about 20 over 1000 cycles for the ISCAS '89 benchmarks).

We have proposed an efficient technique to determine the profitable combination of trace and scan signals. Our approach uses a graph based representation to select three important aspects: (i) efficient trace signals to be stored every cycle, (ii) the most profitable scan signals to be included in the shadow scan chain, and (iii) the scan dump frequency based on the trace buffer width constraints. It is important to note that trace signal states are stored every cycle whereas scan signal states of a specific clock cycle are stored based on dump frequency[1] . A major challenge is that these three

[1] For example, if the trace buffer width is 32, and 8 trace signals are used, we have space left for only 24 scan signals. If we choose a scan chain of 48 flip-flops, the scan dump should be in every two ($48 \div 24 = 2$) cycles. In other words, in clock cycle 0, states

aspects are inter-dependent. For example, selecting more trace signals implies less space for scan signals, and vice versa. Even when the space for the scan signals is reserved, choosing a large scan chain (too many scan signals) implies longer scan dump frequency. In other words, there is a critical balance between how many signals to observe versus how many signal states can be obtained for a specific clock cycle. Our proposed approach addresses these challenges. Our experimental results show that our method can significantly improve restoration performance compared to existing methods.

4.1 Background and Motivation

To explain how combination of trace and scan signals can be used to improve signal observability, we will refer back to Figure 1-4 in Chapter 1. Using only trace signals, A and C, we were able to reconstruct 32 signal states, including 10 traced ones and 22 newly reconstructed ones.

We now show how combination of trace and scan signals can help in signal reconstruction using the same circuit. In the previous example, the trace buffer stored a total of 10 states (width 2 and depth 5). In this case, we use a trace buffer that can store 11 states. Signal C is selected for tracing every cycle. The other two important signals, A and F are used as scan signals. The scan dump is performed in alternate cycles. The modified circuit is shown in Figure 4-1[2] .

Table 4-1 shows the traced, scanned and restored signals using [27]. The state values for signal C is traced every cycle whereas the state values for scan signals (A

of 8 trace signals are stored, whereas only the states of first 24 scan signals are stored. Similarly, in the next cycle, 8 trace signal states and the last 24 scan signals (with states of cycle 0) are stored.

[2] Our method uses partial scan. Recent research by Alawadhi et al. [48] has shown that partial scan can be used without incorporating additional penalty compared to full scan.

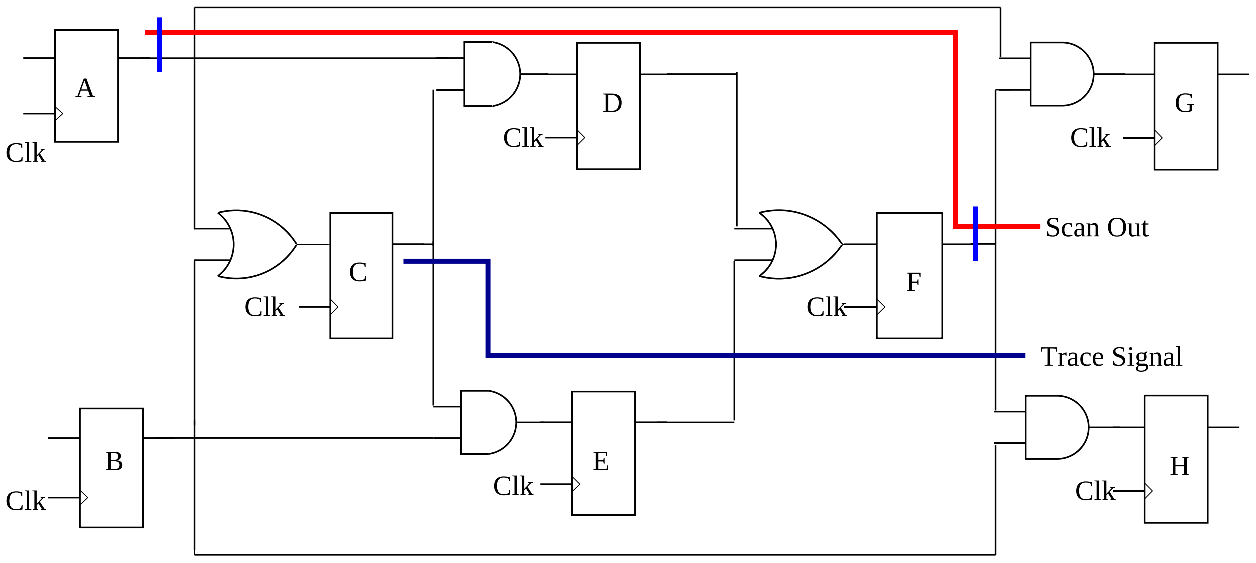


Figure 4-1. Example circuit with both scan and trace signals

and F) are dumped in alternate cycles. The scanned signal states are shown in bold. Although scan signals are dumped in alternate cycles, the table shows states for both A and F in cycle 1, cycle 3, and so on. This is because in cycle 1 the state of signal A is dumped whereas in cycle 2 the state of signal F in cycle 1 is dumped. However, the scan chain (i.e., A and F using shadow flip-flops) holds the state for the same cycle, although different parts were dumped in different cycles. In other words, the signal state of F captured at cycle 1 is dumped in cycle 2. As described by [27], the scan chains need not consist of flip-flops that are physically connected. For example, the scan chain here consists of flip-flops A and F that are connected via flip-flop D, which is not part of the scan chain. In other words, a virtual scan chain can be developed only comprising of the two flip-flops A and F. Although the restoration ratio obtained here is 3.1 (less than the trace-only method), the number of states restored is 34 which is higher than obtained earlier (**32** in case of Table 1-1, as seen in Section 1.2). Thus, more signal states provide a more detailed view of the internal state of the circuit.

The primary problem of combining scan and trace together is to determine what signals to select for tracing, and which ones to be incorporated in the scan chain. Trace signals should be chosen such that they comprise of the important signals in the circuit

Table 4-1. Restored signals using trace and scan

Signal	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
A	0	0	0	0	1
B	1	0	1	0	X
C	1	1	0	1	0
D	X	0	0	0	0
E	X	1	0	0	0
F	1	X	1	0	0
G	X	0	0	0	0
H	X	1	0	0	0

that can control significant parts of the circuit. Scan chains on the other hand should be distributed around the circuit so that the snapshot of the circuit at a particular clock cycle can be obtained during debug. Since the trace buffer is getting divided between the trace signals and the scan chains, it is also important to know how this division is done. Clearly, an equal division might not be beneficial since it would mean less importance to the scan signals and decreasing the number of scan dumps. Ko et al. [27] explored all combinations of number of trace signals and scan dump frequency to obtain a beneficial combination. We have developed an algorithm to select an efficient combination of trace and scan signals to maximize the overall signal restoration.

4.2 Trace and Scan Signal Selection

Similar to trace-only debug approaches, both the trace and scan signals are chosen during the design phase of a particular circuit. The states of the trace signals are monitored every cycle, while the scan signals are dumped at certain time intervals in a repeated fashion. We first introduce our debug architecture. Next, we describe our trace and scan signal selection algorithms.

4.2.1 Trace+Scan Debug Architecture

Our trace-scan combined architecture is motivated by the design of Ko et al. [27]. The entire space of the trace buffer is divided into two parts - one for the trace data and the other for the scan dump. The states of trace signals are offloaded into the trace buffer at every clock cycle. The trace buffer width determines the number of scan signals dumped as well as the scan dump frequency. However, since the trace buffer size is constant, the total amount of data that can be stored remains fixed. With an increase in number of flip-flops in the scan chain, the amount of data produced in each dump increases. As a result, the number of scan dumps has to be decreased in order to maintain the trace buffer constraints. The scan chain is divided into small sub chains to allow complete utilization of the total trace buffer width in the same way as [49]. These are represented as n sub chains in Figure 4-2. The partitions are shown to be numbered from 1 to n. Each of these n sub chains utilize the trace buffer inputs for dumping. To facilitate the tradeoff between scan and trace data, [27] have proposed introduction of multiplexers in front of the trace buffer inputs. This helps in dynamically reconfiguring the inputs for the trace or the scan signals. In our case, the inputs to the trace buffer are predetermined for a particular circuit. Hence, multiplexers are not needed, and this reduces the hardware overhead as well as delay associated with dynamic reconfiguration mechanism. The trace buffer has a width w and depth D. Therefore, the total number of bits that can be stored in the trace buffer are $w \times d$. Here, m of the inputs are dedicated for trace signals, while n sub-scan chains dump their values in the trace buffer. Clearly, $w = n + m$.

Our proposed algorithm comprises of two parts. First, we determine which trace signals are beneficial. Next, we determine profitable set of scan signals and scan dump frequency.

4.2.2 Trace Signal Selection Algorithm

In this section, we determine the signals that are needed to be traced during debug. The main problem that we face here are twofold. First of all, the trace signals need to be chosen efficiently in order to incorporate the advantages of using the scan signals during debug. Also, unlike the trace-only approaches ([7] and [5]), the number of signals to be traced is not fixed. Although, the maximum number of signals to be traced is equal to the

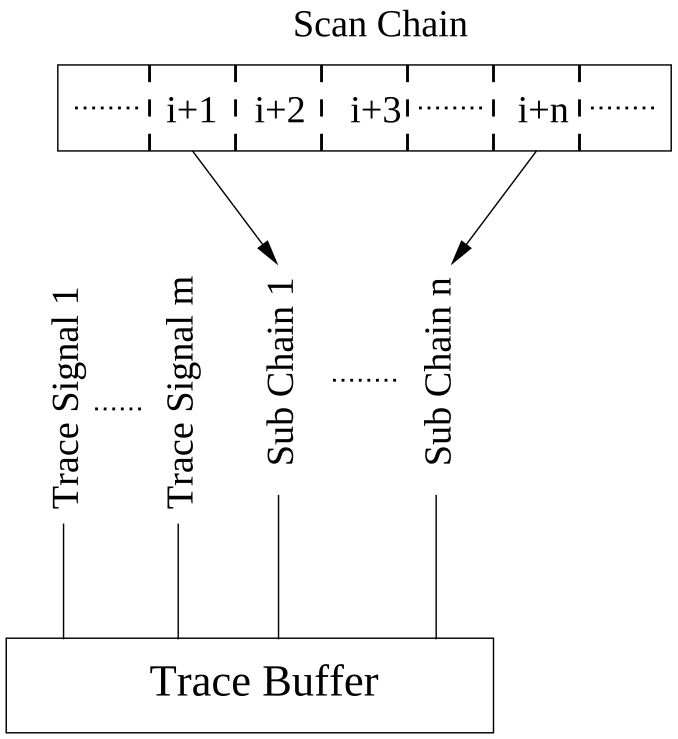


Figure 4-2. Proposed Architecture: The width w of the trace buffer is shared by m trace signals and *n* subchains of the scan chain

trace buffer width, the actual number of trace signals can be less to accommodate the scan signals.

We use two terms connectivity and threshold in our algorithm. The connectivity of a state element is defined as the number of state elements connected with it through other combinational gates (only) in both forward and backward directions, as explained in Section 4.1. The threshold is a minimum limit on the connectivity of a state element, so that it is selected for tracing. We now explain these terms using the example in Figure 1-4. The connectivity of the flip-flops can be determined using the circuit diagram. For example, in Figure 1-4, the connectivity of C is 4, since flip-flops A, B, D and E are connected to it. Similarly, connectivity of flip-flop A is 2 since only C and G are connected to it.

Algorithm 3 outlines the major steps in our trace signal selection algorithm. First we create a graph from the circuit, with each node representing a state element. The edges between the nodes represent the path taken to reach from one state element to the other. This graph construction follows the same methodology described in Figure 3-3. The graph is redrawn in Figure 4-3.

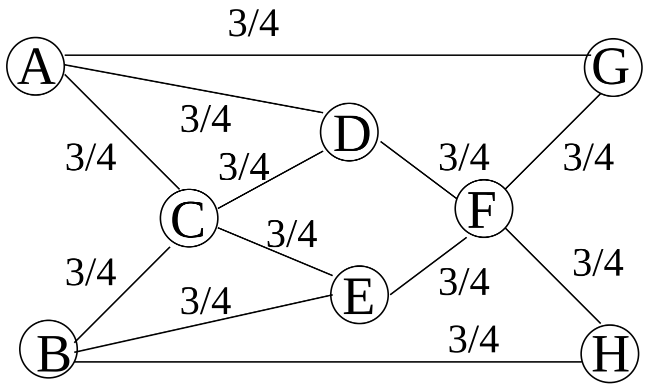


Figure 4-3. Graphical representation of example circuit

Once the graph is constructed, the node with the highest connectivity is selected as the most profitable trace signal. All the adjacent nodes and itself are deleted from the graph. The next node with highest connectivity is chosen. If the connectivity of the node is less than the threshold, the computation stops, otherwise the signal selection procedure goes on until the trace buffer width is reached.

Algorithm 3: Trace signal selection algorithm

```
Input: Circuit, threshold
Output: List of trace signals S (initially empty)
1: Create a graph GP from the circuit.
while trace buffer is not full do
    2: Find node with highest connectivity in GP.
    3: If connectivity is less than threshold return S.
    4: Otherwise, add the new node to the list S.
    5: Delete it and its adjoining nodes from GP.
    6: Re-compute the connectivities of all nodes.
end
return S
```

Let GP denote the graph model of Figure 4-3. In this example, we use 40% of the total number of flip-flops in the circuit (i.e., 3.2) as threshold. The node with the highest connectivity is C, 4, which is more than the threshold. Therefore, C is selected for tracing. Let $R\{C\}$, defined as relations of C, be the set of nodes connected with

C, including C i.e., $R\{C\} = \{A, B, C, D, E\}$. Step 5 of Algorithm 3 recalculates $GP = GP - R\{C\}$. In other words, after deletion of C and its adjoining nodes, the modified GP consists of only three nodes (F, G and H) where F is connected to both G and H. The node with the next highest connectivity in GP is F, with a connectivity of 2. Since this is less than 40%, F is not considered as a profitable trace signal. The algorithm returns C as the selected trace signal.

4.2.3 Scan Signal Selection Algorithm

In this section, we describe our proposed algorithm for selection of both scan chain and scan dump frequency. The procedure to determine the scan chain is shown in Algorithm 4. First, we create a graph from the circuit, in the same way described in Section 4.2.2. Once the graph is constructed, all the nodes that are part of the trace signals or are connected to those trace signals are removed from the graph. Then, a minimal node set is obtained from the graph. A minimal node set has two requirements. First, it is a group of nodes such that each and every other node in the graph is connected to at least one node in the set. Also, it should be minimal i.e., the set should have the least number of nodes. The procedure to obtain the minimal node set is shown in Algorithm 5. The flip-flops corresponding to the nodes in the node set constitute the scan chain. We first describe how the minimal node set is created. Next, we use an illustrative example to describe how the algorithm works.

Algorithm 4: Scan signal selection algorithm

```
Input: Circuit, already selected trace signals
Output: List of scan signals S (initially empty)
1: Create a graph from the circuit.
2: Remove the nodes related to trace
signals and its immediate neighbors.
3: Compute the node values.
4: Find the minimal node set S.
return S
```

4.2.3.1 Creation of minimal node set

The first step constructs a graph model of the circuit. Once the graph has been created, the minimal node set has to be determined. During the creation of the minimal node set, care must also be taken to ensure that the nodes having higher connectivity are selected for scanning. The algorithm for minimal node set construction is shown in Algorithm 5.

Algorithm 5: Minimal signal set creation

```
Input: Circuit as graph, Node values
Output: Minimal Node Set S (initially empty)
1: Put all the nodes in a list GPS.
while GPS is not empty do
    2: Find the node with the highest connectivity.
    3: Remove the node from GPS.
    4: Remove all nodes associated with that node from GPS along with their
    associated edges.
    5: Recompute connectivity values.
end
return S
```

4.2.3.2 Illustrative example

We now explain each of the steps in the algorithm using the graph in Figure 4-3. It should be noted that since the circuit in Figure 1-4 is small, we have not taken into consideration the effect of nodes that have been selected for tracing; that is, we have shown the scan signal approach independently of the algorithm described in Section 4.2.2. In other words, we assumed that there are no trace signals in this case. As can be seen from Figure 4-3, the node C and F have highest connectivity. We choose node C as the initial node. The nodes associated with it (i.e., A, B, D and E) are also removed along with their corresponding edges. The node connectivity information is then recomputed. After deletion of C and its adjoining nodes, the modified GP consists of only three nodes (F, G and H) where F is connected to both G and H. In other words, the connectivity values for F, G and H are 2, 1 and 1, respectively. The node with the next highest connectivity is F. Once F is selected and the adjoining nodes (G and H) are deleted, the graph becomes empty. Therefore, the computation stops. The scan chain obtained comprises of the two flip-flops C and F.

The basic idea behind this form of scan cell selection is that each node in the entire circuit is either in the minimal set or connected to at least one node in the set. Therefore, when the scan dumps (of flip-flops in the minimal set) are performed, the signal states of the nodes that are not in the minimal set can be reconstructed based on scan dumps. For example, in Figure 4-3, if the state of C is dumped in cycle i, the states of A and B may be obtained in cycle $i-1$, while the state of D and E may be obtained in cycle $i+1$.

Now we describe how to compute scan dump frequency for a set of scan signals (scan chain) based on a particular trace buffer size and number of inputs dedicated to the trace signals. Let the trace buffer depth and width be D and w respectively. Let m be the number of inputs dedicated to the trace signals. Therefore, the number of inputs of the trace buffer dedicated to the scan chain per cycle are $w-m$. Let the scan chain length be l. Therefore, number of cycles it takes to dump the entire scan chain into the trace buffer is $\frac{l}{w-m}$. This determines the scan dump frequency, since the scan chain will be dumped after each $\frac{l}{w-m}$ cycles. Since the depth of the trace buffer is D, the number of scan dumps would be $\frac{d\times(w-m)}{l}$.

4.3 Experimental Results

Table 4-2 shows the results of comparison with the existing technique [27]. We implemented both the approaches for the 3 largest ISCAS '89 benchmarks. The trace buffer is chosen with a width of 32 and a depth of 1024. In case of our approach, a threshold value of 10% is chosen for selecting the trace signals. To maintain fairness, in our implementation of the method proposed by [27], we have used the same number of trace signals as our approach and driven the inputs of the benchmarks with the same set of random values. The first column in Table 4-2 shows the circuit name. The second and third columns represent the number of states restored using our approach and the one proposed by [27]. Finally, the last column gives the improvement, which is the ratio of the number of extra states restored by our approach compared to the states restored using [27]. Our approach performed consistently better than [27] and produced up to 17.3%[3] improvement in restoration performance.

Table 4-2. Comparison with existing technique

	Restored States		
Circuit	Our Approach	Existing Technique	% Improvement
s38584	332854	283792	17.3%
s38417	601878	540603	11.3%
s35932	338090	326602	3.5%

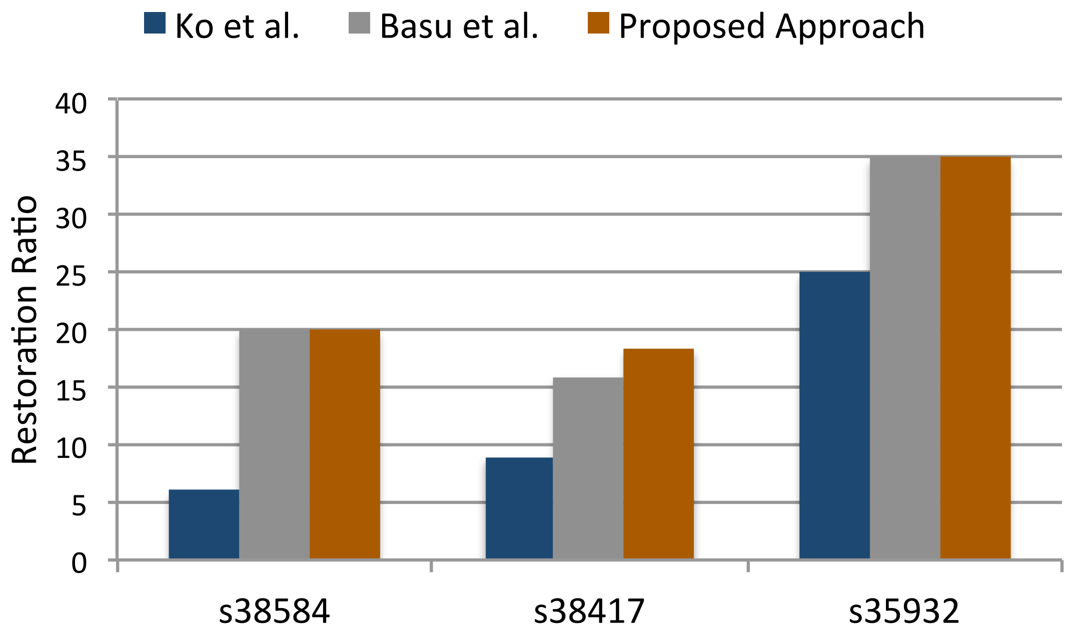


Figure 4-4. Comparison with Ko et al. and Basu et al.

We now compare our proposed approach with the existing trace-only approaches presented by Ko et al. [5] and Basu et al. [7]. Figure 4-4 shows the comparison of

[3] The maximum improvement we obtained is 44% in case of $s9234$. Since [27] did not report it in their paper, we also omitted it.

restoration ratio using the 3 largest ISCAS '89 benchmarks. As expected, our proposed approach outperforms the other two techniques for s38417. However, our proposed approach outperforms [5] for the remaining benchmarks but produces comparable results with [7][4] . The main reason is that these benchmarks have large number of dominating signals, which needs to be traced every cycle. Tracing only a few of them and performing scan dumps at regular frequencies is not helpful. On the other hand, the number of such dominating signals in s38417 is low which requires a high spatial observability of trace signals for improved restoration. Hence, a better performance is obtained with the trace-scan combined approach.

4.4 Summary

Combining trace (non-scan) and scan signals is a promising approach to enhance signal reconstruction during post-silicon debug. We developed efficient algorithms to select profitable trace signals and scan chains to maximize the restoration ratio. Our experimental results using ISCAS'89 benchmarks demonstrated that our method provides up to 17% higher restoration compared to existing approaches. We observed that it is profitable to select only trace signals if a design has large number of dominating signals, otherwise selection of both trace and scan signals is beneficial.

[4] In these cases, our approach can be considered as a trace-only approach, that is, a trace-scan combined approach with zero scan dumps, which selects the best trace signals using the method in [7].

CHAPTER 5
ERROR DETECTION AWARE TRACE SIGNAL SELECTION

Existing trace signal techniques are based on the primary objective of maximizing the restoration of untraced signals, and not detection of errors in the circuit. Error detection in a circuit can be illustrated using the example circuit in Figure 1-4. Errors in a signal are only propagated along the forward propagating path towards the output. Therefore, a signal which is only on the fan-out cone of the erroneous signal can get affected by the error. For example, in Figure 1-4, when flip-flop F is in error, tracing any flip-flop in its fan-in cone (A, B, C, D or E) would not help to reconstruct it. Instead tracing any flip-flop in its fan-out cone (G or H) has a possibility of detecting the error in F. Existing signal selection algorithms rely on both forward and backward restoration[1] for reconstructing an untraced signal. Since an error in the fan-out cone of a signal can not be detected, forward restoration is not meaningful for error detection.

Therefore, *restoration ratio*, the metric used to measure the efficiency of existing signal selection algorithms is not appropriate for error detection. Section 1.3 motivates the need for a new metric that can provide an estimate of the percentage of total errors detected in the circuit. Yang et al. [50] proposed a signal selection algorithm to facilitate early detection of errors. Their approach is focused more on latency and does not deal with the total number of errors detected. Shojaei et al. [51] developed a technique which is dedicated to detection of timing errors only. The authors considered how switching activity and power droop can be used to estimate the errors in a portion of the circuit. However, the authors did not consider the importance of functional or logical errors that might be propagating to the traced signals. We have proposed a signal selection

[1] Forward restoration deals with restoration of the output signal state when one or more of the input signal states are known. On the other hand, backward restoration obtains the input signal states when the output signal state is known. This is discussed in detail in Section 1.2.

algorithm which selects profitable signals for efficient error detection. Our algorithm lays emphasis on how errors, which propagate from an error origin towards its fan-out cone can be detected. Compared to the existing signal selection algorithms, our approach is more efficient (up to 2X) in detecting errors across the entire circuit.

Algorithm 6: Signal Selection Algorithm

```
Input: Circuit, Trace buffer width
Output: List of signals to be traced, TS
$TS = \phi$ /*Initialize to NULL*/
1: Create a graphical representation of the circuit.
2: /*Compute Edge Values*/
For each node $e_{s,d}$, which is an edge between two nodes $s$ and $d$, compute the edge value $p_{s,d}$.
3:/*Compute Node Value*/
Create a set for each node $i$: $S_i = \{(e_{j_1,i}, p_{j_1,i}), (e_{j_2,i}, p_{j_2,i}), ....(e_{j_n,i}, p_{j_n,i})\}$, where $e_{j_k,i}$ represents an edge between node $j_k$ and $i$, where $j_k$ is in the fan-in cone of $i$, and $p_{j_k,i}$ is the value of the edge.
Value for node $i$: $v_i = \sum_{k=1}^{k=n} p_{j_k,i}$, where $n = |S_i|$
4: /*Select Trace Signals*/
while trace buffer is not full do
    Select the node with the largest node value.
    Let this node be $i$.
    /*Add the node to the list*/
    $TS = TS \cup i$
    /*Remove Overlaps*/
    foreach element $t_i$ in $S_i$ do
        foreach element $t_l$ in $S_l$ do
            if they have common source $m$
              & $p_{m,i} \geq p_{m,l}$ then
                $S_l = S_l - (e_{m,l}, p_{m,l})$
            end
        end
    end
end
Return the list of selected signals, $TS$.
```

5.1 Trace Signal Selection for Error Detection

In this section, we propose a signal selection algorithm that is dedicated to error detection in the circuit. Algorithm 6 describes our signal selection algorithm, which has four important steps. The remainder of this section describes each of these steps in detail.

5.1.1 Graph based Modeling of Circuits

Figure 5-1 shows a modified version of Figure 1-4 with labeled signals. Figure 5-2 shows the graphical representation of Figure 5-1. Here, each node corresponds to a signal in the circuit. An edge represents the connectivity/flow between two nodes (signals). For example, presence of the OR gate between signals a and p is represented as an edge between the corresponding nodes. The edges have been shown using directional arrows, which indicate the propagation of error from a source to its fan-out cone. Flow of error from one node to another which is not directly connected will pass through several intermediate nodes. For example, in Figure 5-2, an error from a to c will pass through a, p, c.

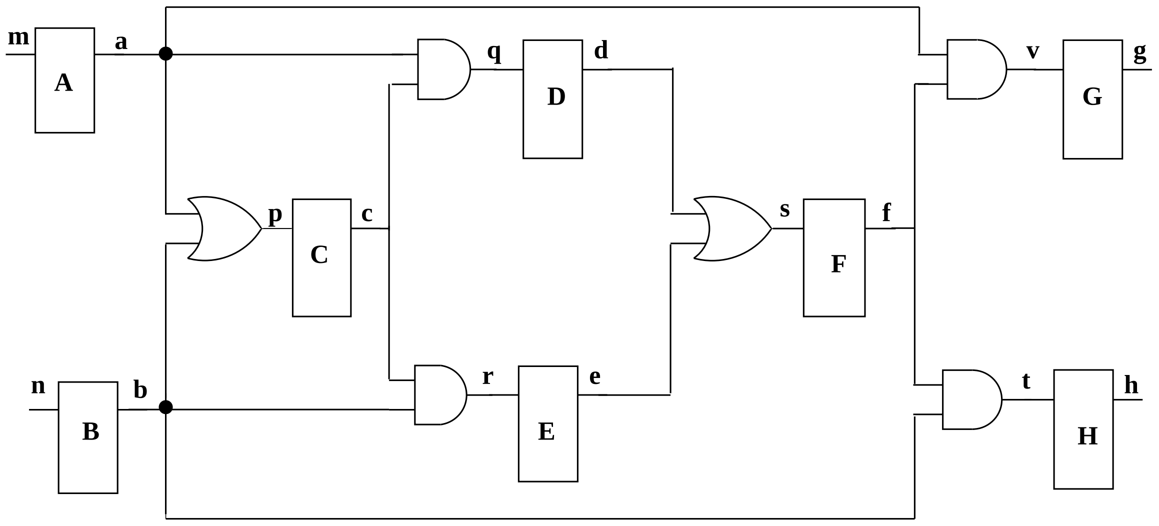


Figure 5-1. Example circuit with labeled signals

5.1.2 Edge Value Computation

We now describe how to compute individual edge values depending on the type of gate. Next, we explain the computation of compound edge values. A compound edge is one which passes through multiple nodes, that is, a compound edge comprises of two or more individual edges.

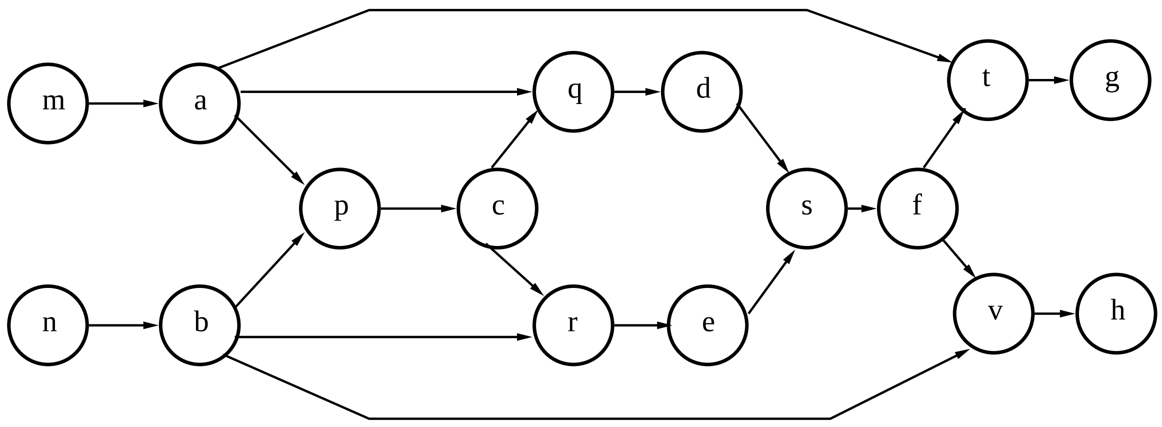


Figure 5-2. Graphical representation of Figure 5-1

AND Gate. To compute the probability of error propagation, we consider the examples in Figure 5-3A, which shows a multi-input AND gate. The graphical representation is shown in Figure 5-3B. Let the inputs be named i_1, i_2,...,i_n and the output o_1 respectively. Let's consider the error propagation from i_1 to o_1. In order for any error (0/1 or 1/0) to propagate to o_1, it is necessary that all the other inputs be tied at 1. If any of the other inputs is at state 0, the error will not propagate. If one of the inputs is at state 0, the output will always be at a state of 0, irrespective of i_1 being correct or erroneous and hence, the error gets undetected. Here, we assume all the other inputs to the AND gate are independent. We consider dependent edges later. Therefore, the probability that all of them are 1 simultaneously is the product of each of the individual probabilities. Let $P^1_{i_n}$ be the probability that input i_n is at state 1. Therefore, the probability that all the other inputs are at 1 is

$$Prob_{i_1} = \prod_{2 \leq k \leq n} P^1_{i_k} \tag{5–1}$$

which is the probability that an error at i_1 will get propagated to o_1. Similar computations can be performed if the gate had been a NAND gate.

OR Gate. The computations for an OR gate follows the approach similar to an AND gate. Let's consider a multi-input OR gate as shown in Figure 5-4A, and the corresponding graph in Figure 5-4B. As before, let the inputs be named i_1, i_2,...,i_n and the output o_2, respectively. Let's consider the error propagation from i_i to o_2. Since in an

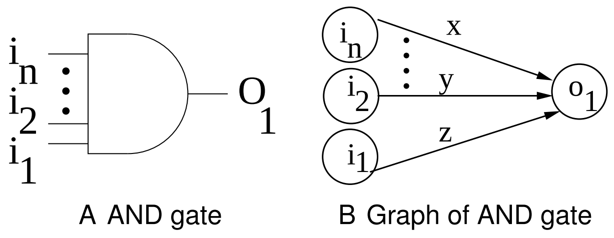


Figure 5-3. Example using AND gate

OR gate, 0 is the non-dominating input; in order to propagate an error in i_1 to o_2, all the other inputs must be held at a state of 0. The probability that an input i_k is held at 0 is $P^0_{i_k}$. The joint probability that all the inputs other than i_1 is held at 0 is

$$Prob_{i_0} = \prod_{2 \le k \le n} P^0_{i_k} \tag{5–2}$$

which is the probability that an error at i_1 gets propagated to o_2. Similar computations can be performed for any of the n inputs and also for a NOR gate.

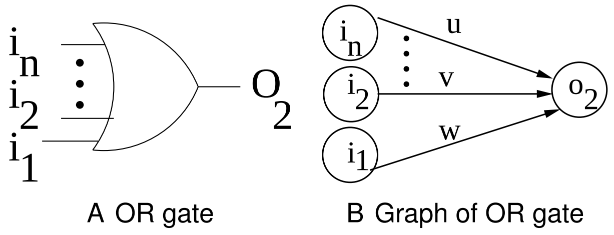


Figure 5-4. Example using OR gate

Flip-flop and NOT Gate. To show how the error propagates through flip-flops and NOT gates, let us consider the examples in Figure 5-5. Figure 5-5(a) shows a D-type flip-flop whose input is i_1 and output is o_3. Any error in i_1 will be transmitted to o_3 in the next cycle. Since there is no other signal dependency between i_1 and o_3, there's no hindrance in error propagation. Therefore, the probability of error propagation is 1 and hence, the value of the edge between the nodes i_1 and o_3 is 1.

Figure 5-5(b) shows a NOT gate whose input is i_1 and output is o_4. Similar to the D-type flip-flop, there is just one input, and hence, any error in i_1 will get propagated to o_4. Thus, the edge value between the nodes i_1 and o_4 is 1.

i_1
D
Q
o_3
clk
Q'
(a) A D Flip−flop
i_1
o_4
(b) A NOT gate

Figure 5-5. D flip-flop and NOT gate

Figure 5-6 shows the values of the edges in Figure 5-2. For simplicity, we assume that all the nodes have a 50% probability of having a state of 0 or 1[2] .

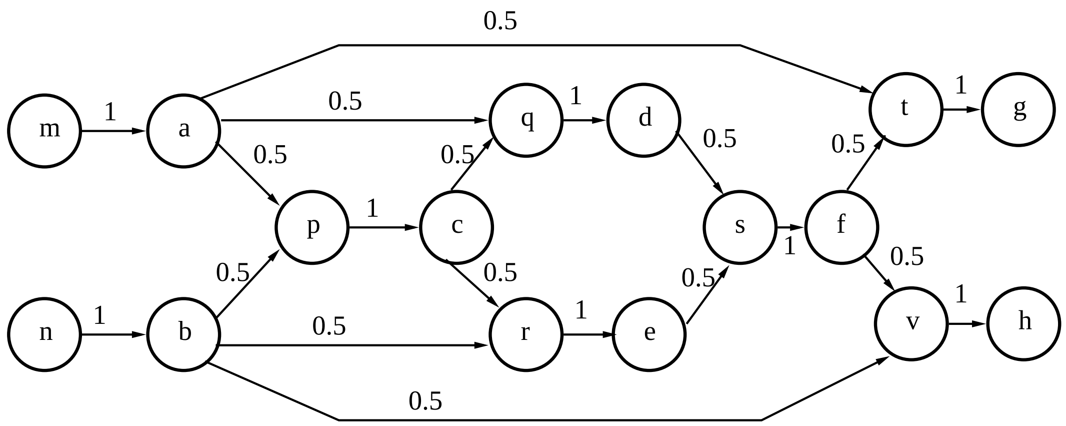


Figure 5-6. Edge values for the graph in Figure 5-2

Now we discuss how the probability of error transmission changes across multiple edges. To calculate the edge value across multiple gates, that is, probability of error propagation from one node to the others in its fan-out cone, we have to consider both independent and dependent edges.

Compound Independent Edge. An independent edge is one which passes from one node to another with each internal node along the path visited at most once. We explain the independent edges scenario using the graphical representation in Figure 5-2. The edge from q to s is an independent edge, since there exists only one path from q to s, via d. In other words, $e_{q,s}$ is traversal of $e_{q,d}$ followed by $e_{d,s}$. Since there is a

[2] In our experiments, we use the profiling information to determine the probability of inputs (nodes) staying at a particular state

flip-flop between q and d, the value of $e_{q,d}$ is 1. This is written as $p_{q,d} = 1$. To compute the value of the edge between d and s, it can be seen that the other input to s is node e. The value of the edge $e_{d,s}$ is the probability that the node e is at state 0, which in this case is 0.5. Therefore, $p_{d,s} = 0.5$. The value of edge between q and s, is the product of $p_{q,d}$ and $p_{d,s}$, that is, $p_{q,s} = 0.5$. This is intuitive because both gates are independent, and hence, the final probability will be a product of how the error gets propagated through each of them. Thus, for an independent edge, the edge value is the product of all the edges which are components of the independent edge. In general, if there are n edges, e_1, e_2,..., e_n comprising an independent edge e, value of e is

$$p_e = \prod_{1 \le k \le n} p_{e_k} \tag{5–3}$$

Compound Dependent Edge. A dependent edge is one which starts from a node, branches of, and finally combines to reach another node. As before, we explain the dependent edge computation using the graphical representation in Figure 5-2. There exists two edges between nodes n and r. One edge is n, b, r while the other is n, b, p, c, r. To compute the value of the compound edge $e_{n,r}$, we need to compute these edge values separately. The value of the edge n, b, r is product of $p_{n,b}$ and $p_{b,r}$ (follows from independent edge value computation). Thus, value of edge n, b, r, that is, $p_{n,b,r}$ is 0.5. On the other hand, $p_{n,b,p,c,r}$ is 0.25. $p_{n,r}$ is defined as $p_{n,r} = max(p_{n,b,r}, p_{n,b,p,c,r})$. This is because when the effect of two different edges are already taken into consideration, an edge with a higher error propagation probability will always dominate, and thus, can be considered an edge value from one to another. In general, if there are n edges e_1, e_2,..,e_n between two nodes s and d, then the value of the edge between s and d, $p_{s,d} = max(p_{e_1}, p_{e_2}, ..., p_{e_n})$

5.1.3 Node Value Computation

We are now ready to compute the node values for Figure 5-6. For each node, we obtain the set of nodes in its fan-in cone and the corresponding edge values. Let us consider the node p. As can be seen, it falls in the fan-out cone of four nodes, a, b, m and n. The corresponding edge values are computed and the set S_p is obtained as $S_p = \{(e_{m,p}, 0.5), (e_{a,p}, 0, 5), (e_{n,p}, 0.5), (e_{b,p}, 0.5)\}$[3] . The node value at p is defined as the sum of all these edges, that is, $v_p = 2$. In this way, the node values are computed for every node in the graph. Figure 5-7 shows the node values for the nodes in Figure 5-6.

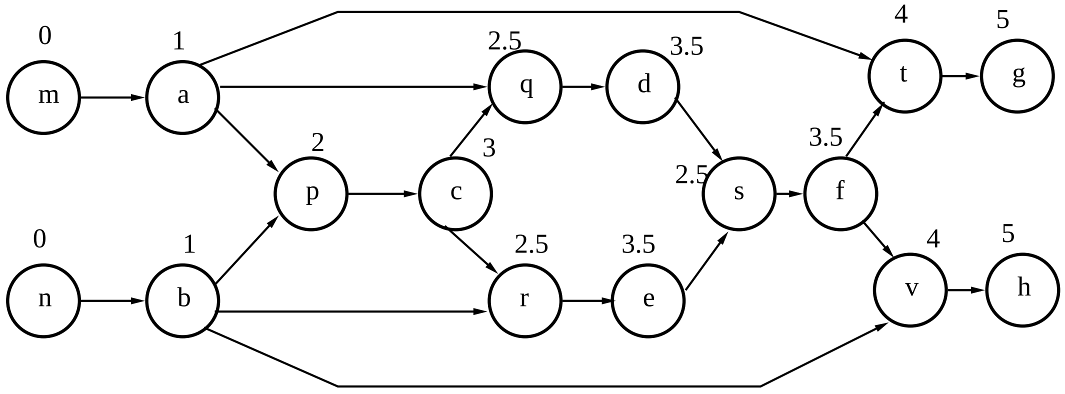


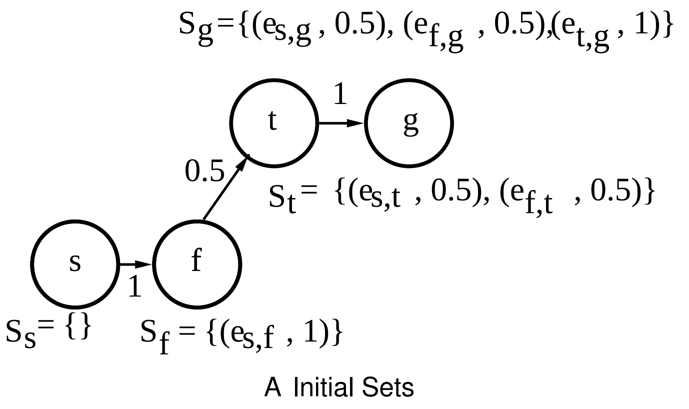
Figure 5-7. Node values for the graph in Figure 5-6

5.1.4 Signal Selection

In this section, we describe the final step in our signal selection algorithm. Once the node values are computed, the node with the highest value is selected for tracing, which is g (or h) in Figure 5-7. The subsequent signals should be carefully selected so as to enhance the error detection. We should delete contributions from signals which have a high probability of error detection from the already traced signals. Step 4 of Algorithm 1 is used for this purpose. For each signal in the circuit, we check how much of its contribution is to the already selected signal as well as to the others. If the contribution to the selected signal is larger than the contribution to some other signal, its contribution to the latter is removed. To illustrate this feature, we refer to Figure 5-8A. The edge values as well as the (edge, value) pair sets have been shown for each node. After g is selected for tracing, the node values are recomputed by removing the overlap.

[3] Each entry in the set is an (edge,value) pair

As can be seen in Figure 5-8A, $p_{s,t} = 0.5$ and $p_{s,g} = 0.5$. Since the probability of an error at s getting detected at t and g are same and g has already been selected for tracing, contribution of s is removed when recalculating the node value of t. By similar arguments, the contribution from f to t, i.e., $(e_{f,t}, 0.5)$, will be also deleted from S_t. On the other hand, since the value of $e_{s,f}$ is greater than $e_{s,g}$, it is not deleted from S_f. The recomputed sets for each of the nodes are shown in Figure 5-8B. Since f has the highest node value, it will be selected next.



Sg={(es,g , 0.5), (ef,g , 0.5),(et,g , 1)}
t
g
1
0.5
St= {(es,t , 0.5), (ef,t , 0.5)}
s
f
1
Ss= {}
Sf = {(es,f , 1)}
A Initial Sets

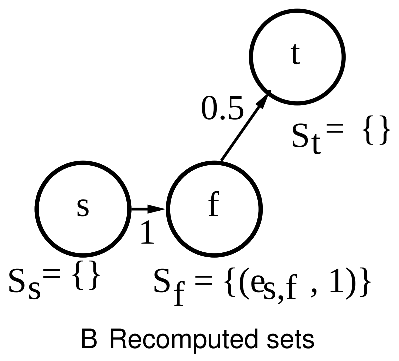


Figure 5-8. Signal Selection based on removal of overlap

5.2 Expriments

5.2.1 Experimental Setup

In this section, we discuss the performance of our algorithm using the ISCAS '89 benchmarks. For each of the benchmarks, 50 different error sites are chosen randomly. The error model is described in Section 5.2.2. A complete simulation of 1000 cycles is performed for both the correct and erroneous scenarios. The signals to be traced are monitored at each clock cycle during simulation. Their states are compared with the states obtained from the perfect simulation. Any discrepancy is reported as error. We define Error Detection Ratio (EDR) as a metric for measuring the effectiveness of a signal selection algorithm. EDR is defined as:

$$EDR = \frac{Number\ of\ Errors\ Detected}{Number\ of\ Detectable\ Errors} \tag{5–4}$$

In a circuit, all errors may not be detected by the state elements. Some of the errors get suppressed before reaching a state element (flip-flops and primary outputs). Hence, it is important to consider the number of errors that can be detected using state elements and not the total number of errors introduced in the circuit. The second and third columns in Table 5-1 show the number of flip-flops and primary outputs respectively, for the four ISCAS '89 benchmarks on which we have performed our experiment. The next two columns indicate the number of errors that are detected using flip-flops or outputs. Note that the total number of detectable errors (last column) are not the summation of values in fourth and fifth columns since there are overlap of detectable errors.

Table 5-1. Detectable Errors for the ISCAS'89 benchmarks

Circuit	#FFs	#outputs	Detectable by FFs	Detectable by outputs	Detectable (total)
s5378	179	49	30	33	38
s9234	228	22	26	2	26
s13207	669	121	38	19	38
s15850	597	87	29	9	29

5.2.2 Error Model

We have assumed a periodically recurring error model which tries to represent a real scenario. Initially, we select a set of random nodes as potential erroneous ones. A random function generator is used for this purpose. We consider each of the errors independently, that is, we consider one error at a time. Once the nodes are selected, a random time stamp is chosen which should be significantly lower than the trace buffer depth[4] . After each occurance of this time stamp, the erroneous node is assumed to malfunction. We have chosen a simple bit-flip model for our purpose. When the node

[4] We have chosen a timestamp of 100 cycles, which is less than the trace buffer depth of 1000 cycles.

is supposed to malfunction, it just flips its correct state. The erroneous state is then propagated along its fan-out cone.

5.2.3 Results

In this section, we compare our signal selection performance with restoration aware signal selection approach [7]. Though there are other signal selection methods such as [5] and [6], we compare our approach with [7], that provides the best results. We have used the same set of signals used by [7] and compared the EDR using both approaches. The results are shown in Figure 5-9. As can be seen, our method provides up to twice improvement compared to [7], when 32 state elements are traced. This is because the scheme proposed in [7] considers both forward and backward restoration when selecting signals for tracing, whereas only forward error propagation (equivalent to backward restoration) is useful for error detection.

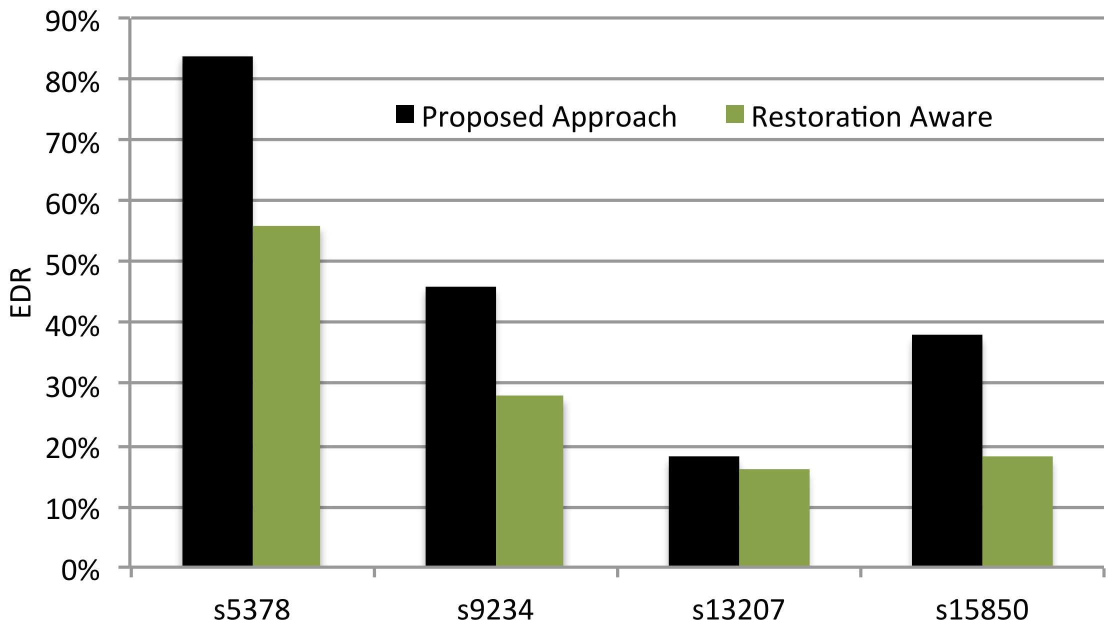


Figure 5-9. Comparison with Restoration aware signal selection

In the next set of experiments, we explore how EDR changes when we increase the number of signals (flip-flops and outputs) to be traced. We consider the number of trace signals 32, 64 and 128 and note the changes in error detection performance.

The results are shown in Figure 5-10. For *s*5378, an EDR of 84% is obtained with a trace buffer width of 32. Further increase of trace buffer width increases the number of errors detected by a small amount (up to 95%). In case of *s*9234, EDR value of 46% is obtained using a trace buffer width of 32. As expected, increase in trace buffer width increases the EDR value. Almost 77% EDR can be obtained using a trace buffer width of 128. For *s*13207, the initial error detection is small (18% EDR with a trace buffer width of 32); however, a sharp increase in error detection performance is obtained when the trace buffer width is increased. In fact, the EDR triples when the trace buffer width is 128. Although *s*15850 is a large benchmark like *s*13207, we see a considerable large EDR when the trace buffer width is 32 (38%). Since there are 673 state elements, further increase in trace buffer width to 64 and 128 does not reflect a sharp increase in EDR value.

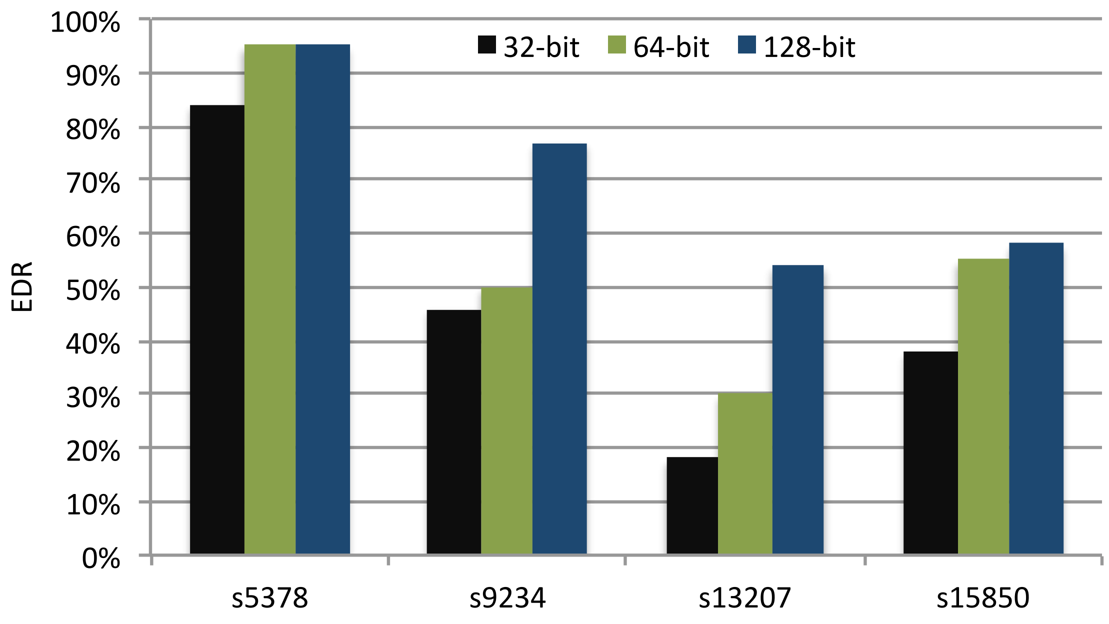


Figure 5-10. Variation of error detection with number of trace signals

Our next experiment is to see how EDR changes when we select only flip-flops for tracing. The results are shown in Figure 5-11. The variation trend is almost similar to that in Figure 5-10, since the number of flip-flops form a major portion of the number of state elements. As before, *s*5378 has most of the errors detected when the trace buffer width is 32, and hence any further increase has minor impact. On the other hand, *s*9234 and *s*13207 have sharp increase in EDR when the trace buffer width is increased.

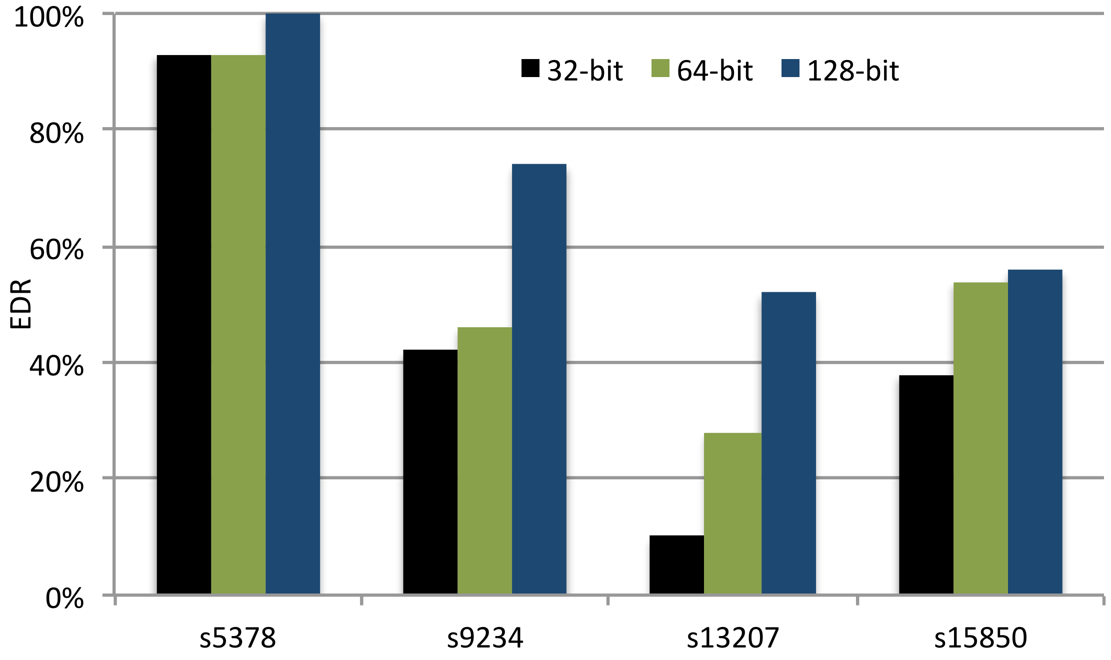


Figure 5-11. Variation of error detection with number of flip-flops traced

Finally, we would like to see how the error detection varies with variation in the number of output signals for tracing. In this case, we trace only the output signals. Since the number of output signals are relatively small, the trace buffer width is varied in steps of 4, 8, 16 and 32. Note that the denominator of EDR computation uses the values of column 5 in Table 5-1. The results are shown in Figure 5-12. For *s*5378 and *s*13207, variation of trace buffer width produces a sharp increase in the value of EDR. For *s*9234, the EDR becomes 100% with width of 4. This is because for *s*9234, only 2 errors can be detected using all the outputs, and these are detected when only 4 outputs are traced. Except *s*13207, which has 121 outputs, 16-bit trace buffer achieves 80-100% EDR for all other benchmarks.

5.3 Summary

In order to detect an error, the traced signals should remain in the fan-out cone of the error signal. We have proposed an algorithm which takes into account this fact

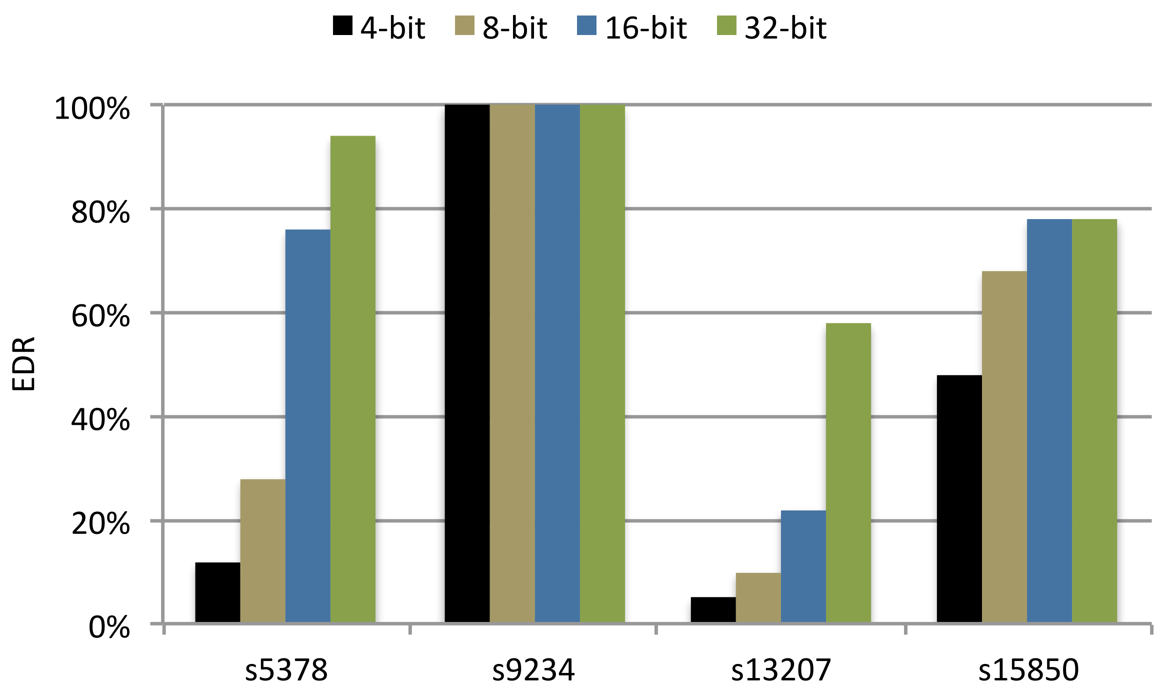


Figure 5-12. Variation of error detection with number of outputs traced

and selects signals with an objective of detecting errors. We have analyzed several cases where any signal, only flip-flops and only output signals are used for tracing. Our proposed approach is significantly better (up to 2X) in error detection compared to the state-of-the-art existing signal selection algorithms.

CHAPTER 6
DYNAMIC SIGNAL SELECTION

To improve the observability during post-silicon debug, existing techniques [5–7, 52] select a small set of profitable signals during design time. The applicability of the existing methods is limited for various reasons. First, these methods treat each component (functional regions) of the design as equally important from debug perspective and therefore select signals that are globally beneficial based on restoration capability. In other words, it assumes uniform “spatial” and “temporal” distribution of errors. In reality, certain regions may not be relevant during some cycles of operation for various reasons. For example, a set of cores in a multicore architecture may be in power saving mode (using clock gating) during certain timeframe. Therefore, no error is possible in those cores during that timeframe. Similarly, certain regions (such as well verified datapath) are less likely to have errors compared to other control-intensive regions. In general, only a small set of regions may be relevant during a particular timeframe for debugging an error. Therefore, a verification engineer would like to have knobs that allow him to trace a different set of signals at different timeframe.

Prabhakar et al. [28] proposed an approach to select between two sets of signals in alternate cycles. As a result, it is a very specific case of temporal distribution of errors without any consideration for spatial distribution. A multiplexed signal selection for error detection was proposed by Liu et al. [29]. Their approach is an ad-hoc signal selection heuristic based on error visibility metric. There is no discussion on how such a selection is beneficial for their targeted debug scenario. In other words, their approach does not consider the challenges associated with dynamic signal selection in the presence of spatial and temporal distribution of errors.

We propose an efficient signal selection algorithm and associated trace controller design that would enable verification engineers to dynamically trace different set of signals for improved error detection. We propose a region-aware signal selection algorithm (RSS) that selects profitable signals during design time (static analysis) based on the knowledge of functional regions and associated error zones. We also develop a low-overhead dynamic signal tracing (DST) hardware to enable designers to trace different set of signals during execution based on active (relevant) functional regions. This lays emphasis on the errors in active zones in the circuit that can be detected using a specifically selected set of trace signals. Although our work might seem similar to traditional test-point insertion and observability analysis [53], it is fundamentally different in two aspects. First, our approach is designed specifically for post-silicon validation and debug. Also, in case of [53], the observation points are determined by the number of signals in the fan-in cone, and not on the error propagation probability from the signals to the observation points. To the best of our knowledge, this is the first attempt in developing an efficient spatio-temporal solution for dynamic trace signal selection. Our experimental results using both ISCAS'89 benchmarks and opencores circuits demonstrate that our approach is able to detect up to 3 times more signals compared to existing state-of-the-art techniques.

6.1 Problem Formulation

The goal of this chapter is to develop an efficient dynamic signal selection technique to maximize detection of currently active errors[1] in a circuit. Various industrial studies highlight the fact that error locations are not uniformly distributed across the circuit, instead they are clustered in multiple small zones. We call them error-prone zones (or error zones, in short). We assume that error zones during post-silicon validation closely resemble those in the pre-silicon phase. We divide the circuit into multiple parts where each part contains one or more error zones. We call these parts as functional regions (or region, in short). A natural boundary for a region would be the component boundary of an SoC. For example, each core in a multicore SoC can form a region.

[1] Those located in active regions, explained later in this section.

If one component has multiple error zones, we may even divide that component into multiple regions following some functional boundary. For example, a processor core can be divided into two regions, one covering fetch and decode units and the other covering the rest. In our construction, an error zone is completely contained inside a region of the circuit, that is, we assume no overlap of an error zone between multiple regions. There is a trade-off between number of regions versus error zones. One region per error zone may create too many regions (partitions) and lead to unacceptable computational complexity and hardware overhead. On the other hand, a large region with many disjoint error zones may reduce the effectiveness of dynamic signal selection.

Let us consider a circuit represented by the rectangle in Figure 6-1. The entire circuit is divided into m regions named R_1 to R_m. Each region can have one or more disjoint error zones. For the ease of illustration, we assume one error zone per region. It does not lose any generality since one error zone can be viewed as a composition of multiple disjoint error clusters. For example, the error zone Z_{R_1} for region R_1 consists of two disjoint error clusters in Figure 6-1. These two clusters together form the error zone Z_{R_1}.

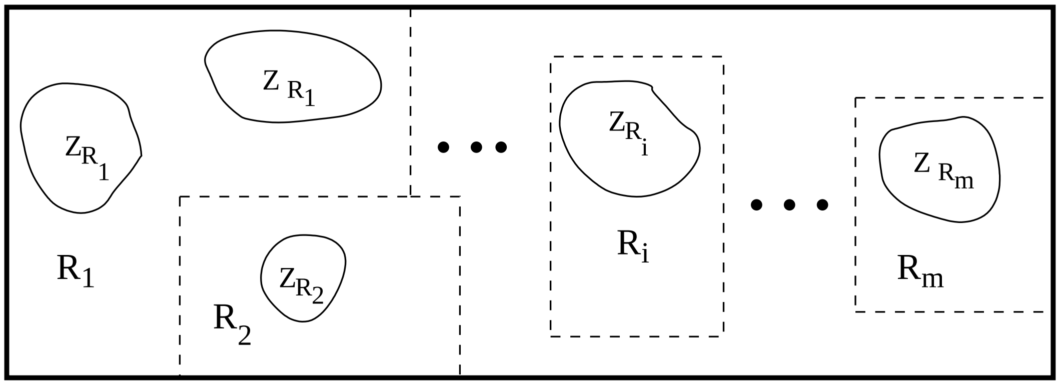


Figure 6-1. Illustrative example showing regions and error zones

We consider a trace buffer of width n, that is, n signal states can be stored in the trace buffer per cycle. During any particular cycle, some of the functional regions remain active (relevant). A region is considered active if the gates in the particular region function normally and not dormant due to power-saving mode (using clock gating) or any other reason. Regions which do not have any signal transition during certain timeframe are considered inactive and hence are not relevant. There are two extreme scenarios. When all the regions are active, our signal tracing algorithm gives proportional emphasis to every region and the associated error zones. However, when only one region is active, beneficial signals from that region need to be traced. We select n signals from each of the m regions, forming a total set of $m \times n$ signals. During execution, depending on the currently active regions, n best signals will be chosen out of $m \times n$ signals. It must be noted that the n signals from region R_i (where $1 \leq i \leq m$) used to detect errors in Z_{R_i} can be from anywhere in R_i (inside as well as outside of Z_{R_i}). Our region-based signal selection algorithm (RSS) in Section 6.2 describes how these signals are selected, while an efficient hardware implementation for dynamic signal tracing (DST) is described in Section 6.3.

6.2 Region-based Signal Selection (RSS)

Algorithm 7 describes our region based signal selection algorithm (RSS) for selecting profitable signals during design time. The first step creates a graph-based model of the circuit. Next, for each region it computes the error propagation probability (defined in Section 6.2.2) from each node in the error zone to the other nodes in the entire region. Finally, for each region the most profitable n signals are selected. The remainder of this section describes these steps in detail.

6.2.1 Graph Based Modeling of Circuits

The first step of Algorithm 7 is to construct a graphical representation of the circuit. We explain this step using our example circuit in Figure 1-5. We redraw the circuit in Figure 6-2 where the 2 regions of the circuit are shown clearly.

The graphical representation is shown in Figure 6-3. Each signal in the circuit is represented by a node and any data flow between two nodes represented by an edge. The edge is irrespective of the type of gate between two nodes. For example, flip-flops

Algorithm 7: Region-based Signal Selection (*RSS*)

```
Input: Circuit, Trace buffer width $n$, Error zones $Z_1$,...,$Z_m$
Output: m lists of selected signals, $SS_1$,...,$SS_m$
$SS_i = \phi$ /*Initialize all lists to NULL*/
1: Create a graphical representation of the circuit.
Divide the circuit into $m$ regions, $R_i$ contains the error zone $Z_i$.
2: /*Compute error propagation probability for each region $R_i$*/
For each node $s$ in $Z_i$, compute the probability of an error at $s$ getting propagated
to any node $d$ in region $R_i$.
4: /*Select $n$ trace signals for each region $R_i$*/
while $SS_i$ does not have $n$ signals or $R_i$ empty do
    4.1 For each node $d$ in $R_i$, compute the summation of the error propagation
    probability for each node $s$ at $Z_i$.
    This is the total error detection probability (EDP) at node $d$.
    4.2 Select the node $j$ with the largest $EDP$ value.
    4.3 Add the node to the list $SS_i = SS_i \cup j$
    4.4. Remove node $j$ and its overlap from $R_i$
end
Return the lists ($SS_i$,...,$SS_m$) with selected signals
```

C and H are connected by a NOT gate, hence, the two nodes representing them have an edge connecting them. Directed arrows signify the error propagation direction. R_1 and R_2 represent two different functional regions of the circuit with respective error zones, Z_{R_1} and Z_{R_2}. Let us consider the region R_1. The probable sources of error are the nodes A and B. Any errors in these nodes can propagate to the other nodes in R_1 which are in their respective fan-out cones. Therefore, the error at A can propagate to F, D, E and G. We would like to compute the possibility of an error at any of these two probable erroneous nodes (A and B) to propagate to the other nodes. We call this probability as error propagation probability, as described next.

6.2.2 Error Propagation Probability Computation

We first describe how to compute error propagation probability through single gates. Error propagation probability is defined as the probability of an error present at an input of a gate being propagated to its output. Error propagation probability over multiple

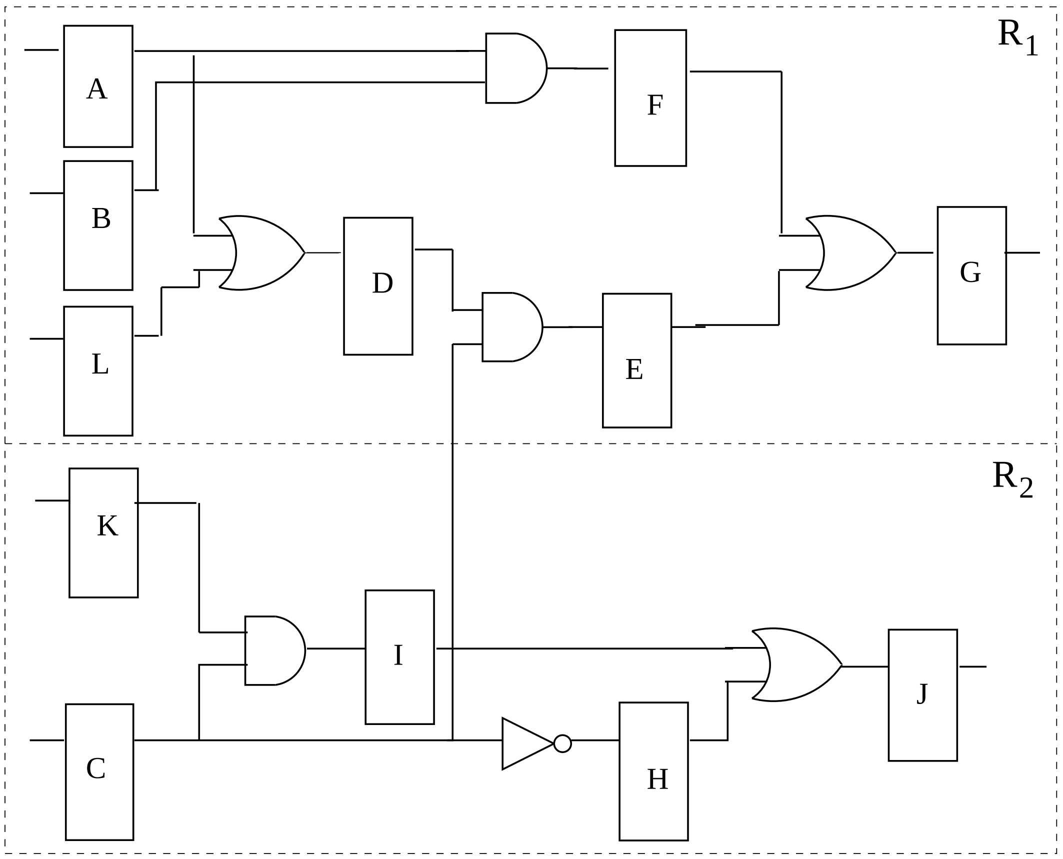


Figure 6-2. Example circuit with 2 regions and 12 flip-flops

gates will be explained later. We consider each of the single gates and compute the probability of an error at one of the inputs getting propagated to the output.

Individual Gates. To compute the probability of error propagation, we first consider a multi-input *AND* gate in Figure 6-4A. Let the inputs be named i_1, i_2,...,i_n and the output o_1 respectively. Let us assume an error occurs at one of the inputs, say, i_1. We want to compute the probability of the error to be propagated to o_1. In order for any error (0/1 or 1/0) to propagate to o_1, it is necessary that all the other inputs of the AND gate be tied at 1. If any of the other inputs is at state 0, the output will always be at a state of 0, irrespective of i_1, hence, the error gets undetected. Here, we assume all the other inputs to the AND gate are independent. Therefore, the probability that all of them are 1 simultaneously is the product of each of the individual probabilities. Let $p^1_{i_k}$ be the probability that input i_k is at state 1. Therefore, the probability that all the other inputs are

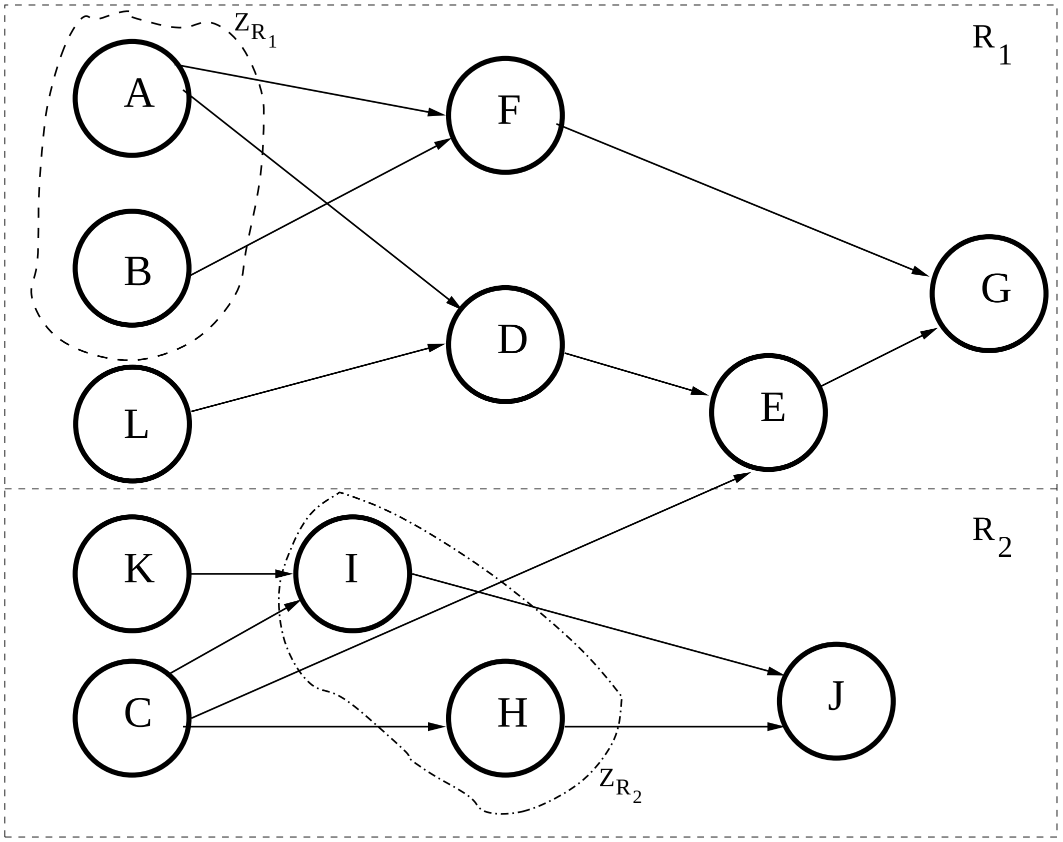


Figure 6-3. Graphical representation of Figure 1-5 with two regions

at 1 is $P_{i_1}^1 = \prod_{2\le k\le n} p_{i_k}^1$ which is the probability that an error at i_1 will get propagated to o_1, that is the error propagation probability through the AND gate. Similar computations can be performed for a NAND gate.

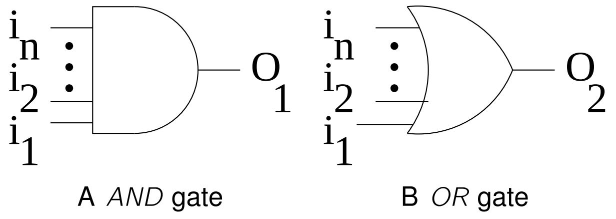


Figure 6-4. Examples using AND and OR gates

The computations for an OR gate follows the approach similar to an AND gate. A multi-input OR gate is shown in Figure 6-4B. Let's consider the error propagation from i_1 to o_2. In order to propagate an error in i_1 to o_2, all the other inputs of the OR gate must be held at a state of 0. The probability that an input i_k is held at 0 is $p_{i_k}^0$. The joint probability that all the inputs other than i_1 is held at 0 is $P^0_{i_1} = \prod_{2\le k\le n} p^0_{i_k}$ which is the error propagation probability from i_1 to o_2. Similar computations can be performed for a NOR gate. For any one input and one output node (such as flip-flop and NOT gate), the error propagation probability is always 1.

Now we discuss how the probability of error propagation changes across multiple gates. Since there are more than one gates involved, we need to consider both independent and dependent paths. A path is defined as the series of logic gates which are placed in between source (s) and destination (d) nodes. In other words, it signifies the path traversed by a potential error at node s to reach node d.

Independent Paths through Multiple Gates. An independent path is one which passes across a set of logic gates with each gate being visited at most once. We explain the independent path scenario using Figure 6-3. To keep things simple, we assume each internal signal being in a state of 0 or 1 with a probability of 50%[2] . The edge from A to E is an independent path, since there exists only one path from A to E, via D. Since there is only an OR gate between A and D, the probability of an error at A getting propagated to D is the probability of L (the other input to the OR gate) being in a state of 0, which is 0.5 in this case. Hence, the error propagation probability between A and D is 0.5. Similarly, since there is only a 2-input AND gate between D and E, the error propagation probability between D and E is 0.5. Since none of the signals are visited more than once, the overall error propagation probability between A and E is the product of these two, which is 0.25. In general, if there are $n+1$ signals in an independent path between nodes s and d, with their intermediate error propagation probabilities being p_1, p_2,..., p_n, the overall error propagation probability across the path is $P_{(s,d)} = \prod_{1\le k\le n} p_k$

[2] This assumption is for explanation purpose only; in real experiments, we gather profiling information to determine the state probability. It is explained in detail in the Section 6.4.

Dependent Paths through Multiple Gates. A dependent path is one in which while moving from a source node to a destination node, at least one of the internal nodes is visited more than once. We explain the error propagation probability computation using Figure 6-3. There exists two independent paths between nodes A and G. One edge is (A, F, G) while the other is (A, D, E, G), both branching out at A and combining at G. In order to compute the error propagation probability across the path between A and G, we need to compute these independent path values separately. For the path (A, F, G), the error propagation probability is the product of the probabilities between the paths (A, F) and (F, G), both of which, for obvious reasons are 0.5. Thus, the error propagation probability of path (A, F, G) is 0.25. On the other hand, since the path (A, D, E, G) passes through 3 independent two-input gates, the eventual error propagation probability is 0.125. The error propagation probability through path (A, G) can be computed as $p_{(A,G)} = max(p_{(A,F,G)}, p_{(A,D,E,G)})$. This is because during computation, the effect of two different paths are already taken into account, and a path with a higher probability of detecting an error will always dominate. In general, if there are n independent paths e_1, e_2,..,e_n between two nodes s and d, then the error propagation probability of the paths between s and d, $p_{(s,d)} = max(p_{e_1}, p_{e_2}, ..., p_{e_n})$

6.2.3 Signal Selection Based on Node Values

In this section, we describe the final step in our signal selection algorithm. The first node chosen for tracing in a region is the node with the highest value. The value of a node is the sum of error propagation probabilities of all paths in which the node is the destination. For example, in Figure 6-5, if we concentrate on Region R_1, the node value of E will depend on paths (A, D, E), (L, D, E) and (D, E). Since in this example only A and B are possible error locations for R_1, the relevant path would be (A, D, E); the paths (L, D, E) and (D,E) is not relevant because D and L are not in error zone. Therefore the node value of E will be the sum of error propagation probability across the path (A, D, E), that is, 0.25. We can have similar computations for other regions. The node values of all the nodes in R_1 are shown in Figure 6-5. Each node value is represented by a number beside it.

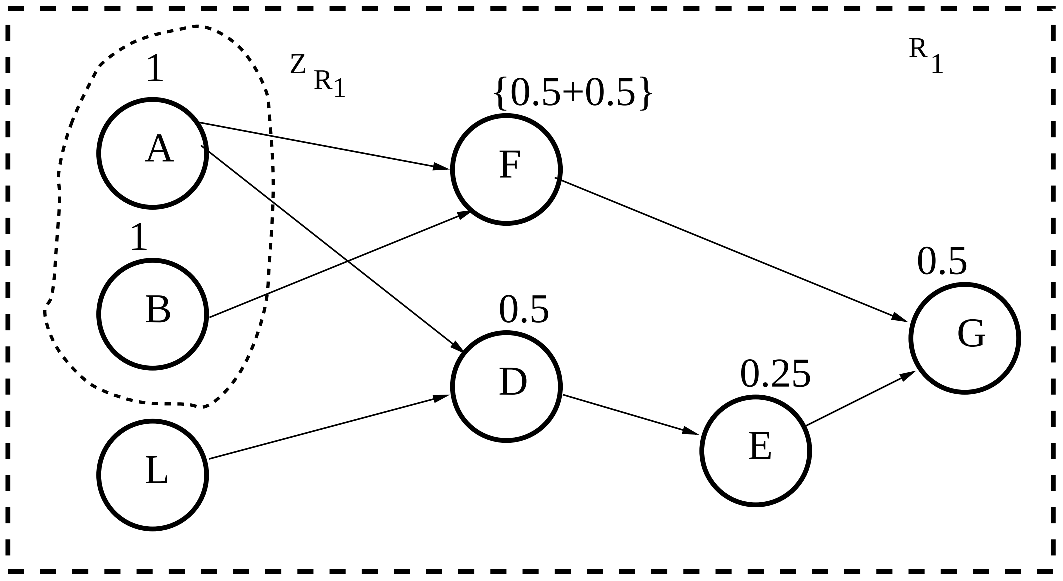


Figure 6-5. Node values for region R_1 in Figure 6-3

In Figure 6-5, three nodes (*A*, *B* and *F*) have highest node values of 1. *A* and *B* are not valid choices since any of them cannot detect the error in other node, whereas *F* can detect error in both *A* and *B* with 50% probability. Therefore, we choose *F* as the first node to trace. The subsequent signals should be carefully selected to enhance the error detection in the region. Contributions from signals which have a high error detection probability from the already selected signals should be deleted. Step 4.4 of Algorithm 2 is used for this purpose. The basic idea is that if an already selected node (e.g., *F*) can detect an error (e.g., in *A*) with equal or higher probability than another node (e.g., *D*), then the overlap from the other node should be removed. For example, since *F* can detect an error in *A* with 50% probability, the contribution claimed by *D* (also 50% for *A*) should be deleted from *D* during the next iteration. This process continues until *n* best signals are selected for each region or there are no more signals to be selected.

6.3 Dynamic Signal Tracing (DST)

Algorithm 8 describes our dynamic signal tracing (DST) procedure for improved error detection. The input to the algorithm are the chip design, trace buffer size, active regions and associated relevance and signal lists. Relevance of a region indicates how important the region is in error detection, or the possibility of finding an error in that region compared to other regions. The relevance information is provided by the pre-silicon verification engineer based on percentage of errors found in the error zone in that region (compared to other error zones) during pre-silicon validation. If no information is available, we can consider the size of the error zone in that region as relevance. If the trace buffer size is n, and there are m active regions in the circuit, during design time our RSS procedure (Algorithm 1) will select $m \times n$ signals. During execution, our DST procedure needs to choose n signals from these $m \times n$ signals that are most profitable at a certain duration depending on the k ($1 \leq k \leq m$) active regions.

```
Algorithm 8: Dynamic Signal Tracing (DST)
    Input: Circuit, Trace buffer size $n$, $k$ active regions ($R_i$), and respective relevance
           ($r_i$) and selected signal lists ($SS_i$)
    Output: List of $n$ signals to be traced, $TS$
    $TS = \phi$ /*Initialize to NULL*/
    1: Here, $r_i$ denote the relevance of region $R_i$, and $SS_i$ is the most profitable $n$
    signals selected for region $R_i$, where $1 \leq i \leq m$.
    Let $r = \sum_{i=1}^{i=m} r_i$
    2: Find the contribution from $R_i$, $C_i = \frac{n \times r_i}{r}$
    3: Select the best $C_i$ signals from $SS_i$
    4: Put the selected signals in $TS$.
    5: Repeat steps 3-4 for all k regions $1 \leq i \leq k$.
    Return the selected signals $TS$
```

Since we have to select n out of $m \times n$ signals for tracing, it is reasonable to adopt n multiplexers, each of which will provide a signal corresponding to the trace buffer output. The main problem is to divide the $m \times n$ signals among the n multiplexers so that all possible combinations of trace signals can be achieved. An obvious but expensive

Table 6-1. Selected Signals for each MUX for $n = 4$ and $m = 4$

Mux name	Input Signals for each MUX
MUX 1	A_1, B_2, B_3, B_4, C_2, C_3, C_4, D_2, D_3, D_4
MUX 2	B_1, A_2, A_3, A_4, C_2, C_3, C_4, D_2, D_3, D_4
MUX 3	C_1, A_2, A_3, A_4, B_2, B_3, B_4, D_2, D_3, D_4
MUX 4	D_1, A_2, A_3, A_4, B_2, B_3, B_4, C_2, C_3, C_4

solution would be to use n multiplexers each having all the $m \times n$ signals as input as 1 output.

One optimization can be achieved by the following observation. Let us consider a circuit with 4 regions, R_A, R_B, R_C, R_D. Suppose the signals responsible for detecting errors in region R_A are named A_1, A_2,...,A_n, in the order of priority. If signal A_1 is not selected for tracing, subsequent signals, that is, A_2, A_3,....A_n will not be selected for tracing. Thus, it is not necessary to keep the signals under the same multiplexer input as A_1. The number of initial signals selected from each region to feed into the n multiplexers are $\frac{n}{m}$. A total of n signals will fill in the first stage of each multiplexer. Now, number of signals remaining for each region is given by

$$n\_remain = n - \frac{n}{m}$$

Under each multiplexer, all these signals except the one from the same region will be stored. For example, if multiplexer 1 has signal A_1, the signals A_2, A_3 and A_4 need not be part of its input. Therefore size of each multiplexer is $size \times 1$ where

$$size = 1 + (m-1) \times (n - \frac{n}{m})$$

Thus savings obtained is $\frac{m \times n}{size}$ which reduces to $\frac{m \times n}{1 + m \times n - 2 \times n + \frac{n}{m}}$. For this example, $m = 4$ and $n = 4$; therefore the value for $size$ is 10. Our initial multiplexer design was of size $(m \times n) \times 1$, which in this case, means 16×1. Thus, we could reduce the multiplexer size by $\frac{16}{10}$, that is, 1.6. Table 6-1 shows the configurations of each multiplexers indicating the signals entering each of them.

Table 6-2. Table for $n = 2$ and $m = 2$

Current State		Selected
R_A	R_B	Signals
0	1	(B_0, B_1)
1	0	(A_0, A_1)
1	1	(A_0, B_0)

Now, we would like to explore the design for our dynamic signal tracing algorithm. The total possible number of states is $2^m - 1$ since at least one of m regions will be active at a time. This is independent of n, that is, the trace buffer width. However each of the states will be defined by n signals, signifying the n signals to be traced at that time. Table 6-2 shows a simple example controller illustrating different signal selections depending on the state of currently active regions when $m = 2$ and $n = 2$. Let the two regions be R_A and R_B. The two signals selected from each region being A_0, A_1 and B_0, B_1 respectively. At any point, only two of the signals are chosen for tracing. When only region R_A is active, the signals to be traced are the two signals from region R_A, indicated by A_0, A_1. Similarly, when only region R_B is active the two signals to be traced are B_0, B_1. When both regions are active, the trace signals to be selected are A_0, B_0.

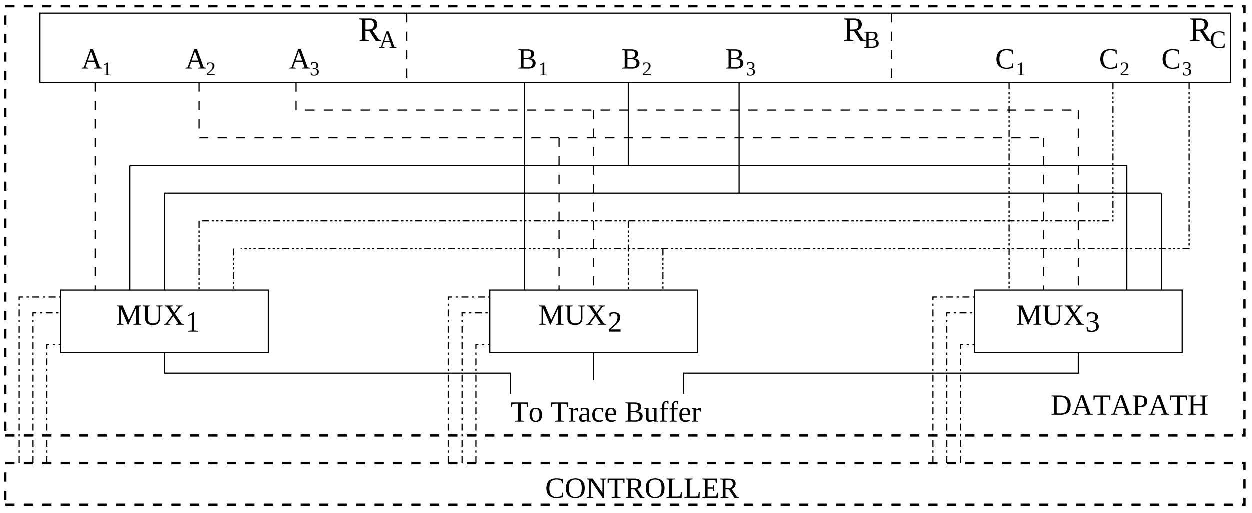


Figure 6-6. Datapath and controller design for $m = 3$ and $n = 3$

Similarly, when $m = 3$ and $n = 3$, it is evident that there will be 7 different states for this configuration. Let the 3 error regions be R_A, R_B and R_C. Signals in R_A are named

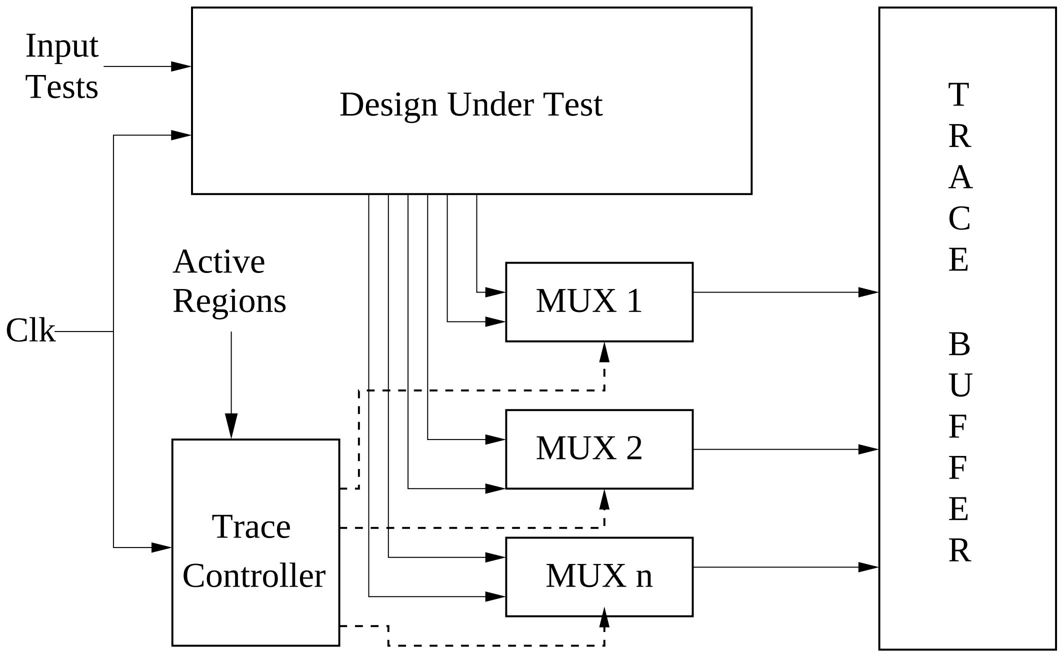


Figure 6-7. Proposed Design

as A_0, A_1, A_2 and similarly for all other regions. Figure 6-6 shows the controller and datapath design for such a configuration.

The overall structure of our proposed design is shown in Figure 6-7. Here we consider a design with n multiplexers that would produce n trace signals. The output of the multiplexers are fed to a trace buffer. The trace controller provides the control signals to the multiplexers based on the logic mentioned above. The trace controller operates under the same clock as the Design Under Test (DUT). An external knob is applied on the trace controller (generally by the validation engineer) which contains information on the currently active error zones in the circuit.

6.4 Experiments

6.4.1 Experimental Setup

We verified the effectiveness of our region-based signal selection (RSS) and dynamic signal tracing (DST) algorithms using some of the largest ISCAS'89 benchmarks as well as opencores circuits. In each of the subsequent experiments, we consider a number of regions with each region having one error zone. Each error zone comprises of about 5% of the respective region. We inserted 50 random errors in the error zones of the active regions, with the error density proportional to the region size. We assume a simple bit-flip model for error, that is, at particular cycle, the error signal will just flip its state. Profiling information is obtained by running an ideal simulation of 1000 cycles with random input vectors and noting the percentage of each signal state. We perform two simulations, one for the ideal case, when all the signals are assumed to be error free, and one with the erroneous signals included. It should be noted that we consider the errors individually, in order to prevent each error's effect from suppressing another. The error model is assumed to be sporadic, that is, errors do not kick off every cycle, but after certain intervals. For our case, we assume the errors to be manifested after a hiatus of 100 cycles. The simulation performed is of total 1000 cycles, that is, a total of 50000 cycles for the 50 errors. Any discrepancy in the traced signal states is reported as error. The metric used to measure error detection performance is Error Detection Ratio (EDR), as defined in Equation 5–4.

We have applied our algorithms using a wide variety of total regions (m) and active regions (k, $k \leq m$). In this section, we summarize the results for three scenarios (each having several subcases): 2 regions (both active and only one active), 3 regions (all active, two active, and only one active), 4 regions (all active, three active, two active, and only one active). In each of these subcases, we present the average of all possible scenarios. For example, in case of $k = 1$ and $m = 2$, the results show the average of two possible scenarios: R_1 is active or R_2 is active. We compare the following three approaches:

- **GSS.** This approach represents the existing techniques that focus on global signal selection (GSS) without any knowledge of error zones or active regions, assumes that the errors are uniformly distributed across the circuit. The signals are selected in an approach similar to [7][3] ; the only difference being that we have considered error detection and not restoration during signal selection.

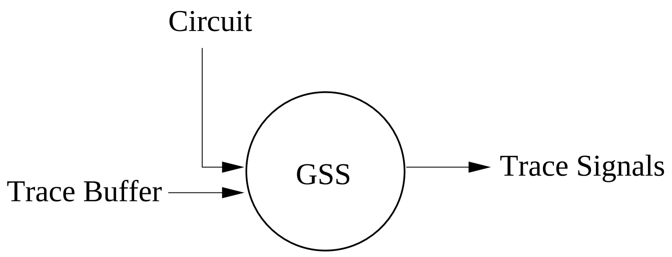


Figure 6-8. GSS

- **EZ-GSS.** We extend the existing methods with the knowledge of the error zones to evaluate their effectiveness in handling error zones. We call this approach as error-zone aware global signal selection (EZ-GSS). This is a static signal selection assuming all zones are active.

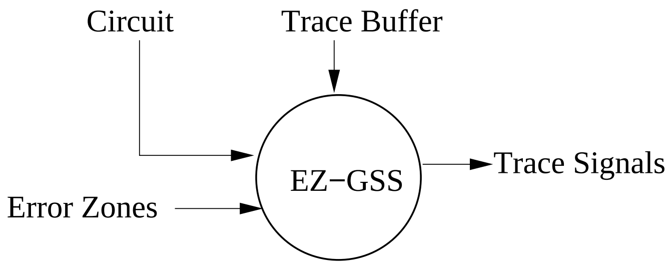


Figure 6-9. EZ-GSS

- **RSS+DST.** Our approach is essentially a combination of region-aware signal selection (RSS) and dynamic signal tracing (DST).

6.4.2 Results for Two Regions

For each of our experimental circuits, we created two regions each having one error zone. In the first set of experiment, we assume both zones are active. In this case,

[3] Although there are other similar techniques like [6, 24, 52] which can be considered as GSS; none of them considers error detection (instead focuses on restoration ratio). Moreover, these approaches do not take into account the presence of error zones and the ones currently active. Therefore, none of these approaches are expected to provide a significant performance. We have chosen one ([7]) from all these approaches as a representative of GSS.

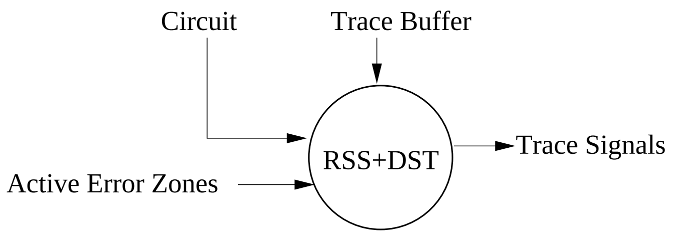


Figure 6-10. RSS+DST

EZ-GSS and RSS+DST are same, since we have to consider both zones for signal selection even during DST. The results are shown in Figure 6-11. As expected, our approach performs better than GSS, with the maximum improvement being 1.75 times, since our approach lays more emphasis on the error zones.

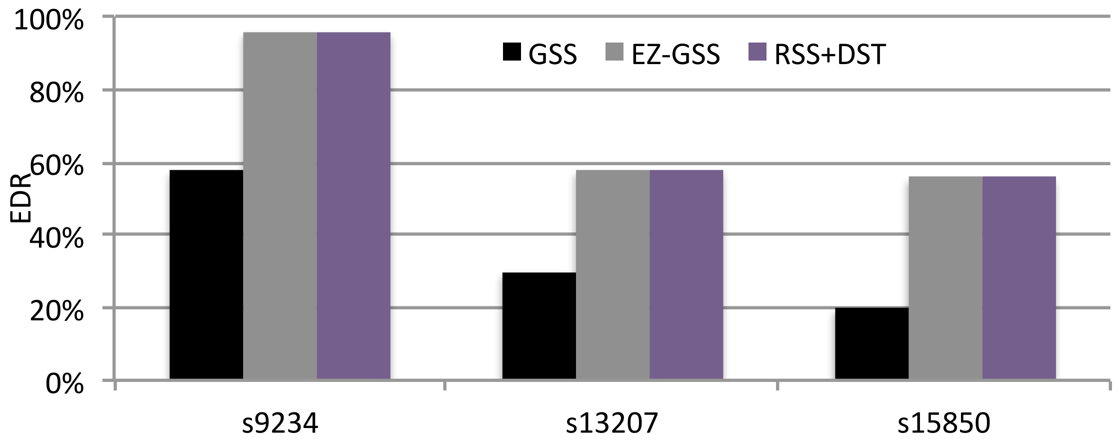


Figure 6-11. Comparison of EDR performance when both regions are active

In the next experiment, we assume one of the two error zones are active at a particular time. Now, we would like to compare the EDR performance of our three approaches, GSS, EZ-GSS and RSS+DST. The results are shown in Figure 6-12. GSS performs the worst among the three since it has no knowledge of where the error is located or which region is active. EZ-GSS performs better than GSS but has no knowledge of active regions. RSS+DST performs the best since it dynamically selects signals with the complete knowledge of currently active error zones. The maximum improvement obtained by our approach against GSS is almost 3 times.

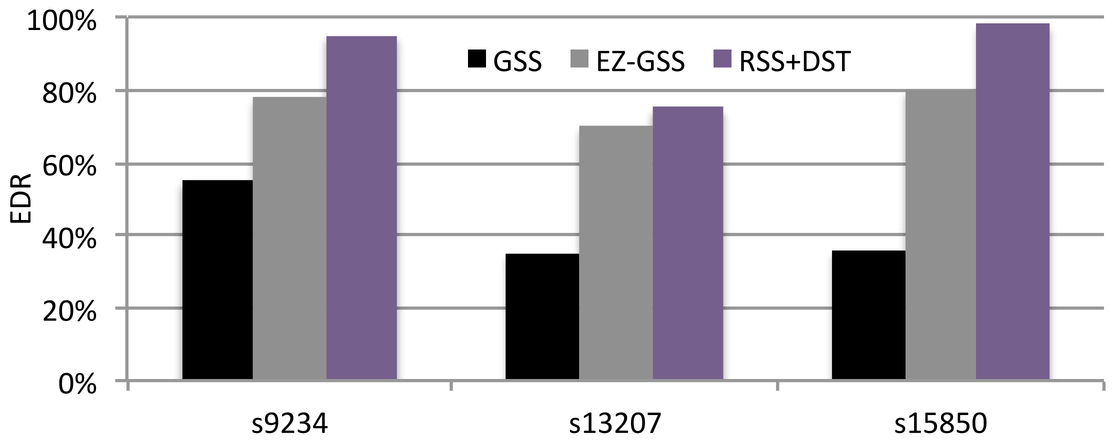


Figure 6-12. Comparison of EDR performance when only one region is active

We would now like to observe the performance of our approach on some real circuits obtained from the Opencores [47] website. We choose three circuits for our purpose, namely RS232 Uart, OPB Onewire and i2cslave. These will be referred to as uart, one and slave respectively for further discussion. We synthesized these using Synopsys Design Compiler to obtain the gate-level netlist from the RTL descriptions. For each of these circuits, we consider two error regions of which one is active at a time. The results are shown in Figure 6-13. As expected, for all three benchmarks, our proposed methods EZ-GSS and RSS+DST performs much better than GSS. RSS+DST performs best in all cases; however for one performance of EZ-GSS and RSS+DST are similar. This is because of all the signals selected using RSS+DST, the ones which can detect most of the errors are selected using EZ-GSS as well.

6.4.3 Results for Three Regions

In these experiments, we created three regions for each circuit. In the first experiment, we assume only one of the three regions are active. The EDR performance comparison is shown in Figure 6-14. The RSS+DST numbers are the average of three possible scenarios of one active region in the circuit. Up to 3 times improvement is obtained by our approach compared to GSS, while compared to EZ-GSS, RSS+DST has a maximum improvement of 1.56.

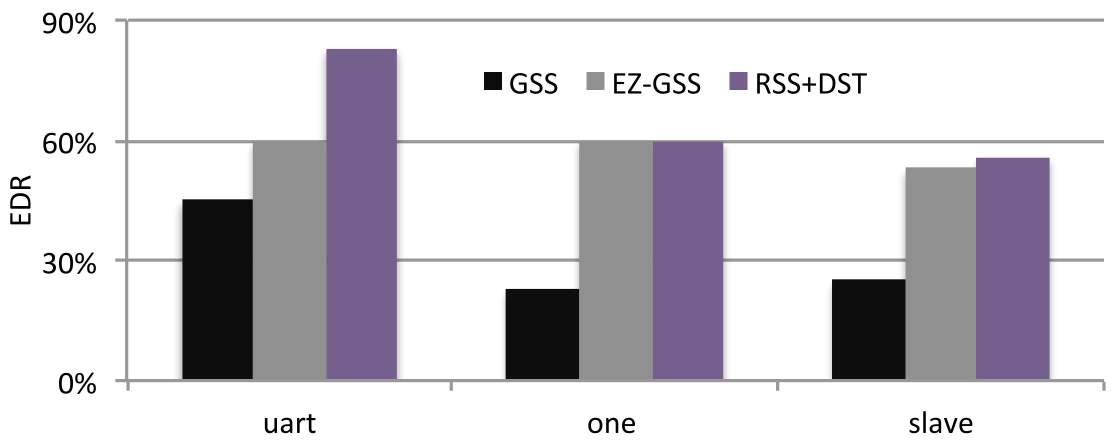


Figure 6-13. Comparison of EDR performance on the Opencores circuits

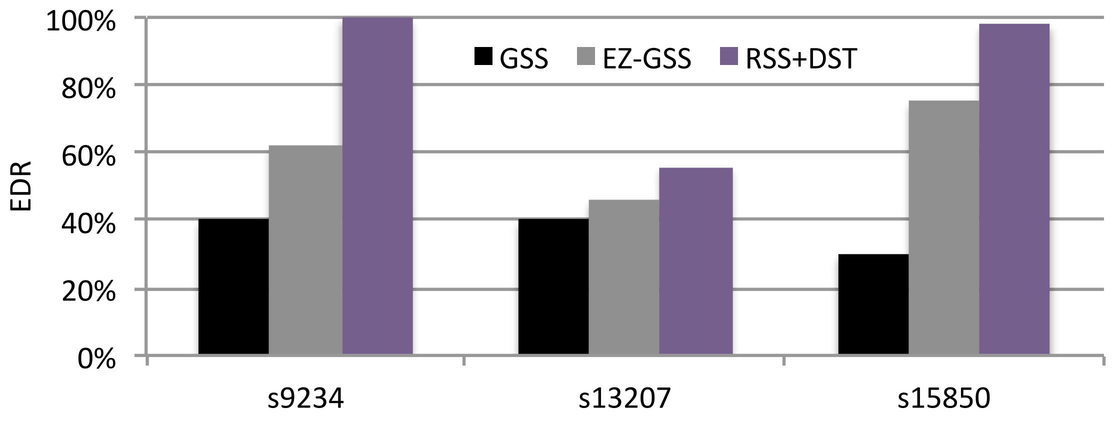


Figure 6-14. Comparison of EDR performance when one region is active

In the next set of experiments, we compare when two of the three regions are active. The results are shown in Figure 6-15. The RSS+DST numbers are the average of three possible scenarios of two active regions in the circuit. RSS+DST performs the best among the three approaches, with the maximum improvement obtained 2 times (compared to GSS) and 1.3 times (compared to EZ-GSS).

6.4.4 Results for Four Regions

In this case, we create four regions in each circuit. In the first experiment, we would like to compare when one among the four zones is active. The results are shown in

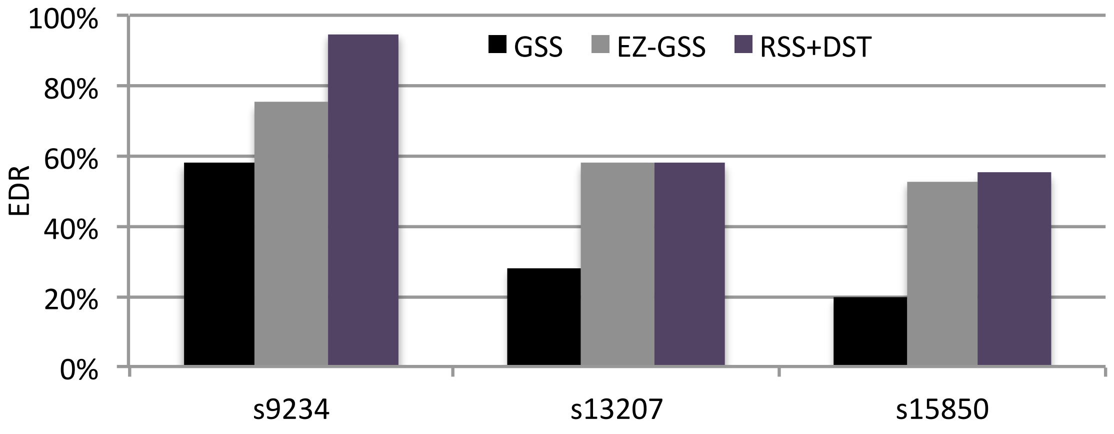


Figure 6-15. Comparison of EDR performance when two regions are active

Figure 6-16. As expected, RSS+DST performs best with the maximum improvement obtained 3.2 times compared to GSS.

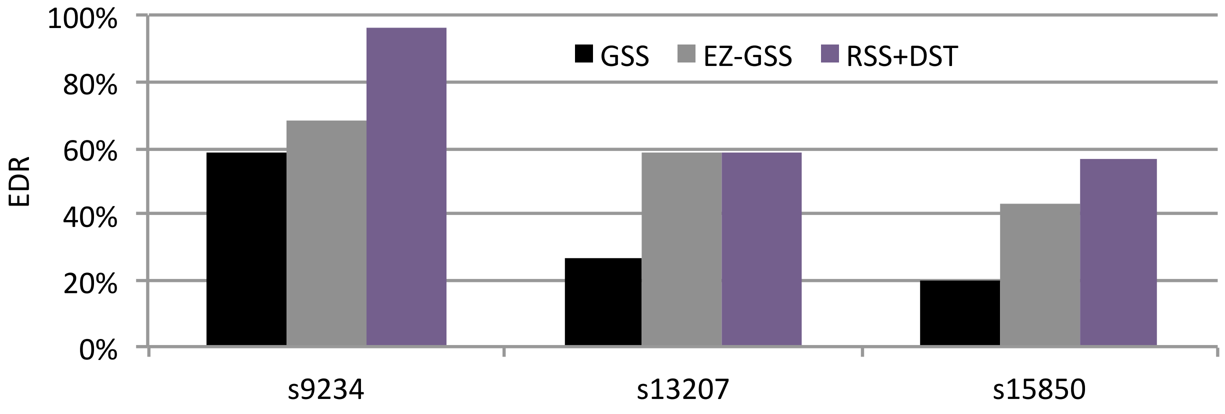


Figure 6-16. Comparison of EDR performance when one region is active

In the next experiment, we observe the EDR performance when 2 of the 4 zones are active. The results are shown in Figure 6-17. RSS+DST performs up to 2 times better in error detection compared to GSS and 1.4 times compared to EZ-GSS.

Finally, we assume that three out of the four zones are active and try to observe the error detection performance. The results, shown in Figure 6-18, reveal that RSS+DST performs better error detection than any of the other two approaches. Note that the improvement is not as significant as in other scenarios. This is because when more

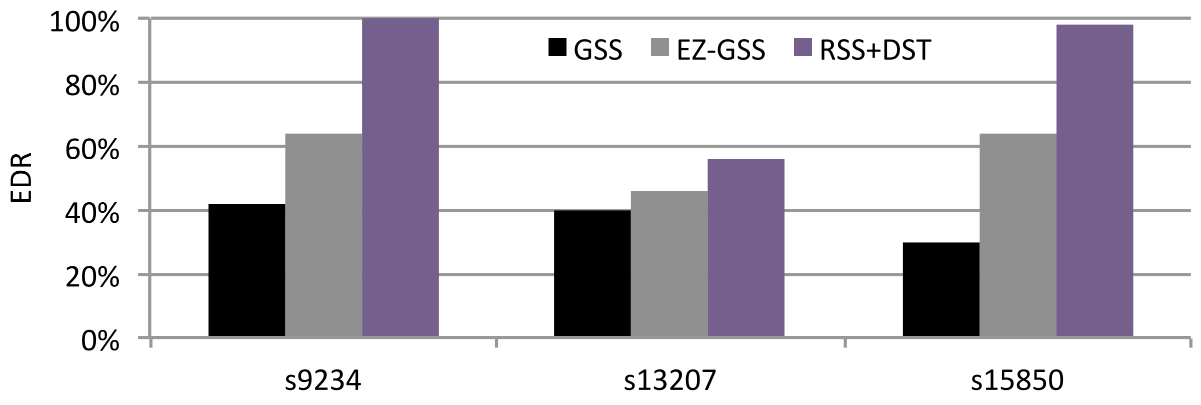


Figure 6-17. Comparison of EDR performance when two regions are active

zones are active, GSS and EZ-GSS can deliver better results relative to RSS+DST compared to when only few regions are active.

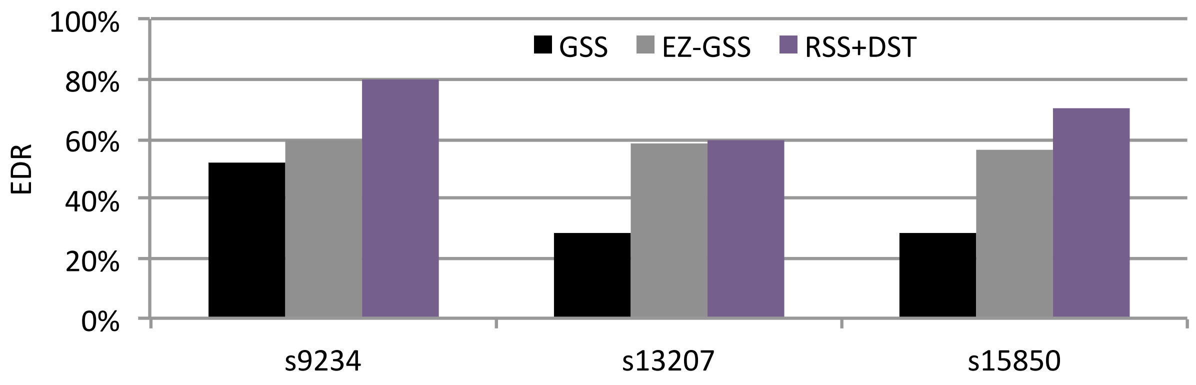


Figure 6-18. Comparison of EDR performance when 3 regions are active

6.4.5 Hardware Overhead

We have developed a Verilog module that is parameterizable for m regions per circuit and n trace signals. We have synthesized both our controller (that generates selected signals for the MUXes) and the datapath (MUX structure) described in Section 6.3 using Synopsys Design Compiler with lsi_10k technology library. The controller area corresponding to $n = 32$ and $m = 4$ (a reasonably realistic scenario) is 239μ. The corresponding datapath area consisting of 32 multiplexers is 185μ. Therefore the total area for our design is $239+185=424\mu$. The trace buffer, which is an integral part of post-silicon debug methodology would occupy much more area compared to the controller. A typical trace buffer of 32×1024 bits, when synthesized using the same library is found to occupy an area of almost 60000μ, which is about 141 times more than the controller area. We believe that the trace controller has acceptable (negligible) area overhead considering that our approach can detect up to 3 times more errors compared to state-of-the-art existing methods.

6.5 Summary

Existing trace signal selection techniques assume that errors are uniformly distributed across the circuit. This assumption may not be valid in many practical scenarios. During design time, our region-aware signal selection approach selects beneficial signals for each region based on information regarding error zones. During execution, our dynamic signal tracing controller enables designer to trace a different set of signals based on regions that are relevant (active) during a certain duration. Our experimental results demonstrated that our approach can detect significantly more (up to 3 times) errors compared to existing approaches.

CHAPTER 7
TRACE DATA COMPRESSION USING STATICALLY SELECTED DICTIONARY

During post-silicon debug, the traced signal states are stored in an on-chip trace buffer. The trace buffer size dictates the amount of data that can be stored, and hence directly affects the observability of the design. Since the trace buffer is used only for debugging, it is better to keep its size as small as possible to reduce the overall cost, area and energy requirements. An option to enhance the observability without compromising on the debug overhead is to compress the trace data before storing them in the trace buffer. We have proposed a lossless dictionary based width compression scheme that operates in real-time to compress the trace data. Unlike [16], our method chooses the dictionary offline, which provides a better compression performance as well as huge reduction in compression architecture overhead. Three different compression algorithms have been proposed to trade-off between compression performance and architecture overhead.

We have used Compression Ratio, defined in Equation 7–1, as a metric to measure the efficiency of a compression algorithm. A higher compression ratio implies a better compression.

$$Compression\ Ratio = \frac{Uncompresssed\ Data\ Size}{Compressed\ Data\ Size} \tag{7–1}$$

7.1 Trace Data Compression

The existing compression techniques compress the trace data by selecting a dictionary dynamically during execution. This not only results in inferior compression performance (due to non-optimal dictionary selection), but also increases the architecture overhead. This section describes our trace data compression techniques. The overview is shown in Figure 7-1.

Our approach is based on an important observation. In any post-silicon debug environment, after the trace data is collected from the chip, it is validated by checking

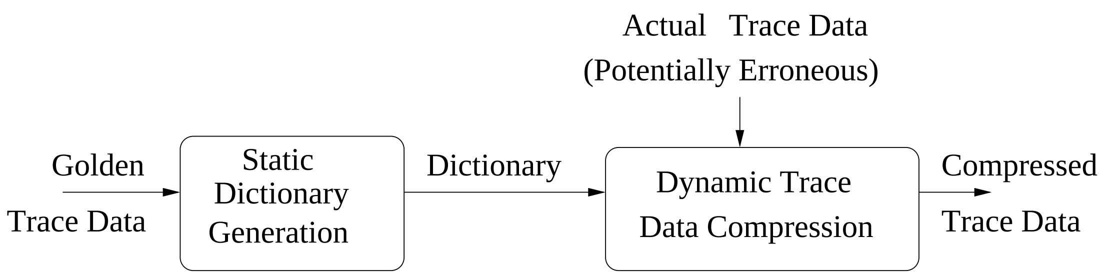


Figure 7-1. Overview of our trace compression procedure

with a set of ideal trace data, that is obtained from a golden model. Since very few (2-5%) bugs actually remain to be tracked during the post-silicon debug phase, there are a few cycles which produce erroneous values [14, 15], that are different from the ideal ones. We utilize this information to design our approach. Since the difference between the ideal and the actual trace data is very small, the same dictionary applicable for ideal trace data compression can be reused for compression of the actual trace data. This takes care of the two problems by providing a better compression performance, and reducing the architecture overhead[1] as well. These compressed data are then read out through a channel to a debugger, where they are checked against the ideal trace data. Any discrepancy in the trace data is reported as error. As can be seen from our analysis in Section 7.1.3, introduction of 2-5% error in trace data results in 2-6% penalty in compression performance, which is acceptable. It can be seen from the discussions in Section 7.2, even with the introduction of errors, our technique provides less compression penalty compared to the existing trace compression methods [16]. The remainder of this section describes our dictionary selection algorithms and also performs a theoretical analysis of the maximum penalty possible when the dictionary from the ideal trace data is used to compress the actual (potentially erroneous) trace data.

[1] no need to implement a dynamic dictionary selection algorithm.

7.1.1 Dictionary Selection Algorithms

We have explored three compression algorithms for compression of the trace data, namely Dictionary based compression (DC), Bitmask based compression (BMC) and fixed Dictionary MBSTW (fMBSTW) based compression. All these three techniques use a dictionary for compression. The dictionary selection is extremely vital since it would be reused to compress the actual trace data. We will now describe how the dictionaries are selected in order to achieve the maximum compression performance.

7.1.1.1 Dictionary-based compression (DC)

Algorithm 9 outlines the dictionary selection method. In a dictionary based compression, the main aim is to include in the dictionary all the unique entries which have maximum repetitions in the dataset. Therefore, the first step determines all the unique entries in the dataset. We then find the number of repetitions for each entry. The unique entries are sorted in a descending order of the number of repetitions. The entries with the highest number of repetitions are included in the dictionary. Details on dictionary selection for bitmask-based compression has been explained in [54, 55].

Algorithm 9: Dictionary selection algorithm for DC

```
M = Number of unique entries
N = Number of Dictionary Entries
DIC = Dictionary
for each entry in M do
   Calculate the number of repetitions in the entire dataset
end for
Sort the M entries in decreasing order of repetition count
Include the first N entries in DIC
```

7.1.1.2 Bitmask-based compression (BMC)

The dictionary selection for bitmask based compression follows the same trend as the dictionary based compression, that is, select dictionary entries giving the maximum savings. However, there is a minor difference between the two. While savings for DC corresponds to just the repetitions, for BMC it includes those due to bitmask based matchings as well. Hence, the savings for each unique entry should be calculated based on the direct as well as bitmask based matches. The entries are then sorted in order of savings and included in the dictionary. The dictionary selection algorithm is shown in Algorithm 10.

Algorithm 10: Dictionary selection algorithm for BMC

```
M = Number of unique entries
N = Number of Dictionary Entries
DIC = Dictionary
for each entry in M do
    Calculate the savings due to repetition and bitmask based matching in the entire
    dataset
end for
Sort the M entries in decreasing order of total savings
Include the first N entries in DIC
```

7.1.1.3 Fixed dictionary MBSTW compression (fMBSTW)

The compression technique for fMBSTW algorithm follows the same technique as MBSTW compression [16]. The difference from MBSTW is that the dictionary is selected statically and the number of dictionary entries is limited. We would now explain the dictionary selection steps for fMBSTW in Algorithm 11. This algorithm is shown for a 2-fMBSTW (2-strings are encoded together, similar to 2-MBSTW). This can be further extended to 3-fMBSTW, where 3 strings are encoded together.

Figure 7-2 shows an illustrative example for dictionary selection using Algorithm 11. In this example, the strings in the trace data are represented using p, q, r, s, t. The amount of savings for each 2-tuple is shown in Figure 7-2. We want to have a dictionary of size 4. As can be seen, the highest savings is obtained from the 2-tuple $< r, s >$. Both of these are now included in the dictionary. The last_entry is s here. Now, we proceed to see which 2-tuple with the first entry s has the maximum savings. $< s, p >$ is selected as the 2-tuple and included in the dictionary. When searching for the next 2-tuple, it is seen that $< p, r >$ gives the highest savings. However, r is already present in the dictionary.

Algorithm 11: Dictionary selection algorithm for fMBSTW

```
M=No. of unique entries
N=No. of Dictionary Entries
DIC=Set of Dictionaries
first_entry = last_entry =NULL
Create a 2-tuple for each pair of entries in M
for each 2-tuple do
  Calculate the savings across the entire dataset assuming only this tuple is in the
  dictionary
end for
Find the 2-tuple with the highest savings and add it to DIC
first_entry = first entry of 2-tuple
last_entry = last entry of 2-tuple
N = 2
while Size of DIC less than N do
  find the 2-tuple that starts with last_entry and produces maximum savings
  if such a 2-tuple exists then
    Include the 2-tuple in DIC, N = N+1
  else
    Find any 2-tuple (not containing an entry already in DIC) which has the highest
    savings and include it in DIC
    last_entry = last entry of the 2-tuple, N = N+2
  end if
end while
```

Hence, $< p, r >$ is avoided. The 2-tuple having the next highest savings is $< p, t >$. Therefore, t is selected for the dictionary. In this way, the dictionary is built up.

7.1.2 Dynamic Trace Data Compression

Our final goal is to debug the DUT, for which we need the trace data from it. Application of a set of tests produces the trace data from the DUT which are compressed to reduce the size of the trace buffer. The overview of the compression architecture is shown in Figure 7-3. As can be seen, the compression architecture consists of two parts, the dictionary and the actual compression engine. Depending on the design and associated constraints, a specific compression algorithm and its corresponding dictionary is used. For example, when BMC is most suitable for a design, the compression engine will have BMC in it and the dictionary will be the one selected

Tuple	<p,q>	<p,r>	<p,s>	<p,t>	<q,p>
Savings	12	25	4	17	11

Tuple	<q,r>	<q,s>	<q,t>	<r,p>	<r,q>
Savings	12	12	7	19	14

Tuple	<r,s>	<r,t>	<s,p>	<s,q>	<s,r>
Savings	28	21	15	9	12

Tuple	<s,t>	<s,p>	<s,q>	<s,r>	<s,t>
Savings	12	14	7	9	11

Savings for each tuple

r
s
p
t

Final Dictionary

Figure 7-2. Example of dictionary selection in fMBSTW

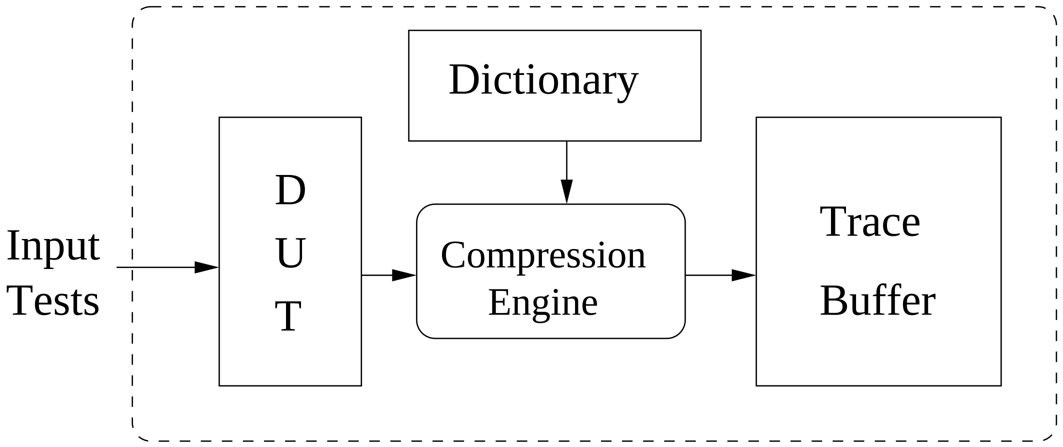


Figure 7-3. Actual Trace Data Compression

for BMC. It should be noted, that the dictionary size is fixed here and not variable as in the case of dynamic dictionary selection [16]. Actually, [16] tried to include every single unique string in the dictionary. This increases the dictionary size, thereby introducing significant architecture overhead and also degrades the compression performance (since the number of bits used to index the dictionary increases with an increase in dictionary size). Our approach eliminates these disadvantages by keeping a limited number of profitable entries in the dictionary.

7.1.3 Performance Analysis with Erroneous Trace Data

Our approach is promising due to use of statically selected dictionary. However, this dictionary will be used to compress actual (potentially erroneous) trace data. This section analyzes our procedure and determines the performance degradation that may occur when the dictionary obtained from ideal trace data is used to compress the actual trace data. We have kept the trace data length constant at 32 bits. We introduce a term compression penalty, which is the ratio of the number of extra bits needed for compression when error is introduced, compared to the original trace data length. Obviously, a lower compression penalty signifies less number of bits needed to accommodate the error, and hence, a better compression performance.

$$Compression\ Penalty\ (CP) = \frac{Number\ of\ extra\ bits\ needed}{Size\ of\ original\ trace\ data} \tag{7–2}$$

We first analyze the compression penalty for DC and BMC. Next, similar analysis is performed for fMBSTW.

7.1.3.1 Compression penalty for DC and BMC

We try to obtain the compression penalties for the two methods DC and BMC. In this section, we make two important observations.

Theorem 1. When statically selected dictionary (based on golden trace data) is used, the compression penalty is bounded by the percentage of error introduced in the actual trace data.

Proof. Let there be x strings in the original trace data. Let the percentage of error in case of actual trace data be l, expressed as a fraction ($l < 1$). The introduction of error changes $l \times x$ strings. In the worst case, all these $l \times x$ strings will be among the strings originally compressed, and these will now be uncompressed due to contamination. Let the number of bits required to compress the rest (that is $(1-l) \times x$ strings) in the dataset be M[2] . It should be noted that these strings are not affected due to error injection and hence, the value of M remains constant in both cases. Let the number of dictionary entries be 2^d, so that d bits are needed to represent the dictionary. The $l \times x$ strings were compressed in the ideal case using $(1+d)$ bits each. If y_{ideal} be the number of bits after compression for the ideal trace data, it can be rewritten as,

$$y_{ideal} = M + l \times x \times (1+d) \tag{7–3}$$

Now, let's analyze the actual trace data. In the worst case, all of the $l \times x$ strings remain uncompressed. Each of these strings will require 33 bits[3] to be represented. The M bits required to represent the $(1-l) \times x$ strings will remain the same. If y_{faulty} is the number of bits needed to represent the strings now, it can be represented as

$$y_{faulty} = M + l \times x \times (33) \tag{7–4}$$

which implies,

$$y_{faulty} = y_{ideal} + l \times x \times (32-d) \tag{7–5}$$

Therefore, number of extra bits needed, represented as y_{extra}, is

$$y_{extra} = y_{faulty} - yideal = l \times x \times (32-d) \tag{7–6}$$

[2] Some of the strings may be compressed, while the rest uncompressed

[3] 32 bits (original size), plus one bit to indicate not compressed

If CP_{DC} is the compression penalty for DC, then from the definition,

$$CP_{DC} = \frac{I \times (32-d)}{32} \tag{7–7}$$

As can be seen CP_{DC} is always less than I, and hence is bounded by it. □

For example, with 8 dictionary entries, we have d=3, and assuming the error rate is 5%, (which is the maximum error rate in these scenarios [14, 15]), we get

$$CP_{DC} = 4\%$$

Thus, we see that a very slight compression penalty is introduced in DC even in the worst case. It can be seen from Equation 7–7 that increase in dictionary size can lessen this degradation.

Theorem 2. Compared to the ideal case (if dictionary was selected using erroneous trace data), the compression penalty using statically selected dictionary (using golden trace data) will be bounded by the twice the percentage of error.

Proof. We would like to see if the actual trace data were compressed without the help of ideal dictionary, how much compression would be obtained. In this case, the dictionary entries might differ from the ideal dictionary. If n is the extra number of strings that can be compressed in the actual case and M is the number of strings that were compressed in the ideal case, then the total number of strings compressed are $m+n-I\times x$. It is obvious that the maximum value of n can be $I\times x$, otherwise, these new strings would have been compressed in case of ideal trace compression, that is, these new strings would have been represented in the ideal dictionary. Therefore, the maximum number of strings compressed is M, which is the same case as in golden trace data.

As an example, consider a hypothetical trace data set of 20 entries. Suppose we choose the best 2 entries in the dictionary, each of which can compress a total of 5 entries. Therefore the total number of compressed entries will be 10. Corresponding to the symbols described above, x=20, m=10 and d=2. Let the error rate be 10%, that is l=0.1. When error is introduced, the number of strings contaminated is $l \times x$, that is, 2. In the worst case, both these strings were part of M and are now left uncompressed due to errors. The number of compressed strings now are $m - l \times x$, which is equal to 8. Now, if we try to compress these erroneous data with a different set of dictionary, let the number of extra strings being compressed be n. It is obvious that if n is greater than 2, the new dictionary would have been selected in the first place, so that the value of M would be different. So, the maximum value of n is bounded by $l \times x$.

However, in the best case, these contaminated strings can be all compressed using some other entry, which is not part of the dictionary now. Let us reiterate our previous example to explain this. For example, all of the $l \times x$ contaminated entries can be compressed using some other entry. Now, if that entry has high enough frequency, it will be included in the dictionary. In this example, the maximum frequency (original, without contamination) that an entry can have is 5; otherwise, it would have been included in the original dictionary. Therefore, the maximum number of strings that can be compressed with the new dictionary is $m + l \times x$, that is, 12 in this case. Hence, the maximum number of strings that can be compressed extra using the dynamically selected dictionary is $(m + l \times x) - (m - l \times x)$, that is, $2 \times l \times x$, which means the difference in compression ratio should be $2 \times l$. Therefore, the difference in compression efficiency between the dictionary based on golden data and dictionary based on actual data, will be bounded by twice the error rate in the data. □

It can be noted that the analysis for BMC will be similar to DC. This is because, even for BMC, the worst case comes when some strings which were completely compressed (not using bitmasks) change to uncompressed due to error introduction.

7.1.3.2 Compression penalty for fMBSTW

To find the compression penalty, we analyze the worst case condition for 2-fMBSTW here. The worst case scenario can be divided in two parts. The first part is similar to that of DC and BMC, that is, the worst part comes when some completely compressed strings become uncompressed. The second part of the condition is explained as follows. Suppose, two consecutive strings correspond to two consecutive dictionary entries a, b. Therefore, all the two strings will be compressed using the 11 prefix, followed by the dictionary entry corresponding to a. However, if either a or b gets contaminated by error, in the worst case, one of them is uncompressed and the other one gets compressed separately, which requires more bits to compress the trace data. We now investigate the compression penalty in this approach.

Let the error rate and the number of strings be l and x as before. Let d be the number of bits to represent the dictionary index. Therefore, the number of such strings changed is $l \times x$. Each of these string corresponds to a $< a, b >$ tuple which is broken due to perturbation. Before the introduction of error, the number of bits required to compress these is given as y_{ideal} in Equation (7)

$$y_{ideal} = l \times x \times (2 + d) \tag{7–8}$$

Here, 2 bits are needed to represent the prefix 11 and d bits for the dictionary index of a. After perturbation (of b), in the worst case, a is independently compressed as single bits using the prefix 01[4] . Therefore, the number of bits needed to represent are $(36 + d)$[5] . There will be $l \times x$ such occurances. Therefore, the total number of bits needed to represent the erroneous tuples is given by y_{faulty} as

$$y_{faulty} = l \times x \times (36 + d) \tag{7–9}$$

[4] as discussed in Section 7.1.1.3

[5] $(2 + d) + (2 + 32)$, where $2 + d$ bits are needed to compress a and $2 + 32$ bits are needed to represent the uncompressed string b

As before, let M be the number of bits required to compress the other $(1-l) \times x$ strings. Since M is unchanged in either case, the number of extra bits needed, is given by

$$y_{extra} = y_{faulty} - y_{ideal} = l \times x \times 34 \tag{7–10}$$

Therefore, the compression penalty is given by,

$$CP_{fMBSTW} = \frac{l \times 34}{32} \tag{7–11}$$

With a 5% error rate we can see that,

$$CP_{fMBSTW} = 5.31\%$$

Thus, even with an introduction of 5% error, the compression penalty is small. These analysis will be later verified with experimental results in Section 7.2.3.

7.2 Experiments

We have compared the compression performance of our approach with the algorithms proposed by Anis et al. [16] (MBSTW and WDLZW). We have also investigated our compression performance when the number of dictionary entries are varied. We have shown that our methods require much less compression architecture overhead compared to those in [16]. Finally, in Section 7.2.3, we have also analyzed the effect of introduction of errors on compression ratio and validated the equations developed in Section 7.1.3. We have applied all the algorithms on the 5 largest ISCAS 89 benchmarks.

7.2.1 Compression Performance

First, we compare the compression performance of our algorithms with the algorithms in [16] using the traces obtained from ISCAS 89 benchmark circuits. The traces were obtained by following the approach outlined in [7]. The results are reported in Figure 7-4. We have fixed the dictionary entry to be 8 in each of the two compression algorithms, DC and BMC. For MBSTW, we have used the 2-MBSTW algorithm[6] . For the fMBSTW algorithm, the number of dictionary entries is floored to the nearest integer which is a power of 2. It can be seen that the fMBSTW approach works best in all cases except s38584. This is because the traces of s38584 have very less number of unique entries. As a result, even with 8 dictionary entries, a large portion of the circuit can be compressed using DC. DC works better than MBSTW in most cases and worse only in some cases (s9234 and s35932). The reason for this is the large number of unique entries in those trace data, which are effectively captured by MBSTW, but not by the 8-entry dictionary used in DC. If the number of bits needed to represent the compressed data is analyzed, it can be seen that fMBSTW provides up to 60% reduction in compressed data size compared to MBSTW and 70% compared to WDLZW. WDLZW provides worst performance for almost all the benchmarks. The high redundancy in the trace dataset is responsible for its somewhat good performance in s38584 and s38417.

Next, we vary the dictionary size of DC to see the effect on compression ratio. The results are shown in Figure 7-5. We have varied the number of dictionary entries from 8, 16, 32 and 64. As can be seen from Figure 7-5, the variation is not uniform for all the benchmarks. For s9234, s13207 and s35932, the compression ratio increases with increase in dictionary entries. On the other hand, for s38584 and s38417, increase in number of dictionary entries worsens the compression ratio and the optimal compression is achieved at 8 dictionary entries. Once we reach an optimal compression ratio, any increase in the number of dictionary entries will add to the total compressed data size both due to the increased number of entries in the dictionaries and increase in the number of bits representing the dictionary index.

[6] Provides better performance than the 3-MBSTW algorithm

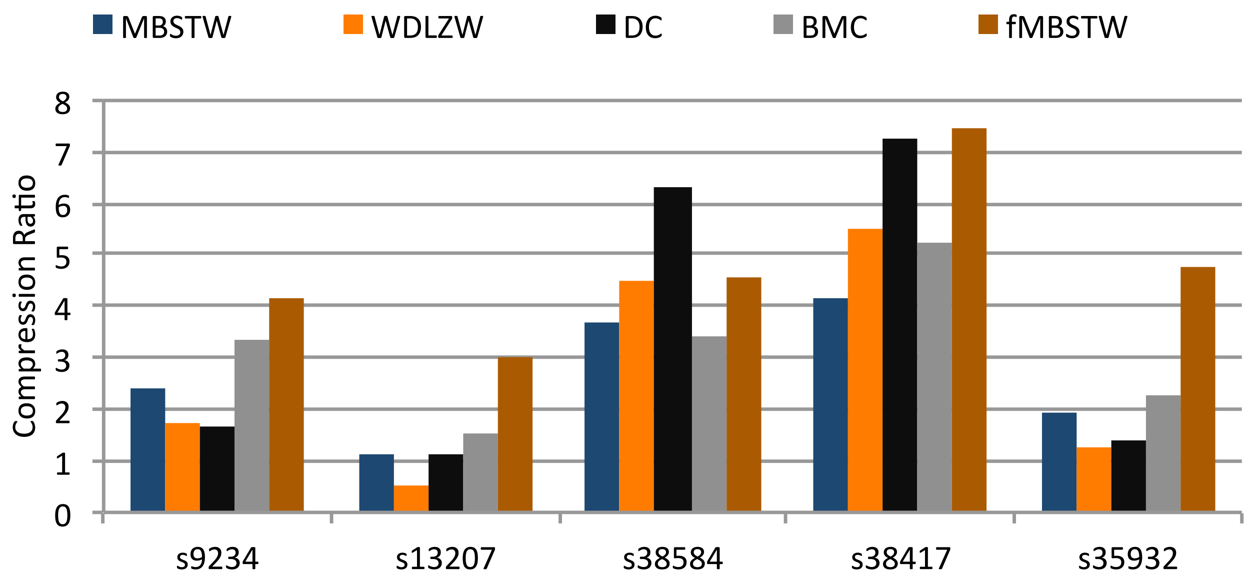


Figure 7-4. Comparison of compression performance

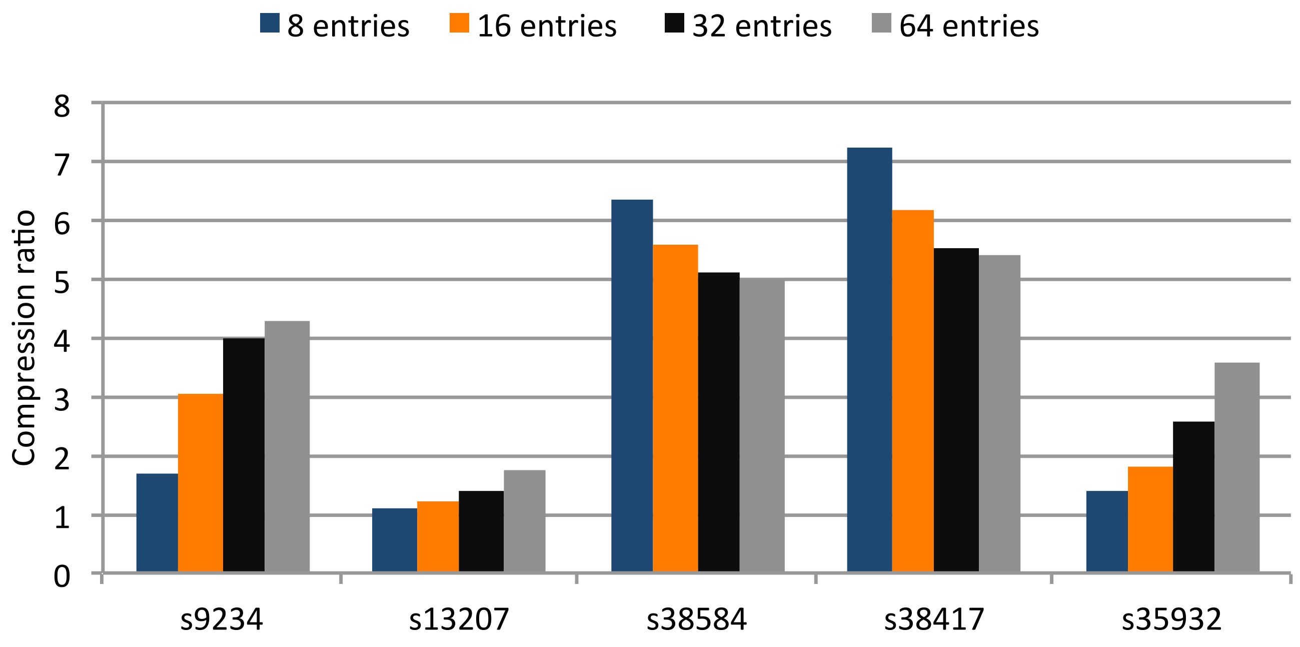


Figure 7-5. Compression performance with dictionary entries

7.2.2 BRAM Requirement (Hardware Overhead)

The dictionary for compression has to be stored in on-chip 32-bit BRAMs. We have computed the total size of BRAMs needed for compression using each of these algorithms. Figure 7-6 compares the requirements for each of these approaches. It can be seen that since the dictionary size is always fixed (8 entries) for DC and BMC, the number of BRAMs required in these two algorithms is significantly less than any other approaches. WDLZW has the highest number of BRAM requirements since it captures all the double symbol repetitions (which worsens the compression performance). For fMBSTW, the number of BRAMs is kept floored to the nearest higher power of 2 for the number of unique entries in the stream[7] . From Figure 7-6, it can be seen that our two methods (DC and BMC) provides almost 96% less compression architecture overhead compared to MBSTW and almost 99% less than WDLZW.

It can be seen that there is a tradeoff between better compression ratio and lower architecture overhead. As can be seen from Figure 7-4 and Figure 7-6, either of the two techniques BMC or fMBSTW can be applied based on priority - BMC can be used for least area overhead (up to 96% reduction) with reasonable compression improvement (10%) compared to MBSTW, whereas fMBSTW should be used for best possible compression (up to 60%) while providing reasonable (up to 84%) reduction in BRAM requirement.

7.2.3 Compression Performance with Erroneous Trace Data

We now like to validate the analysis done in Section 7.1.3. Errors have been inserted randomly at a rate of 2% to 10% in steps of 2% in the trace data, and the same is compressed using DC, BMC and fMBSTW. Figure 7-7 shows the comparison of compression penalty in DC with varying percentage of error. It can be seen that the change in compression penalty complies with Equation 7–7 in Section 7.1.3. For

[7] Results in Figure 7-4 are also reported using this configuration

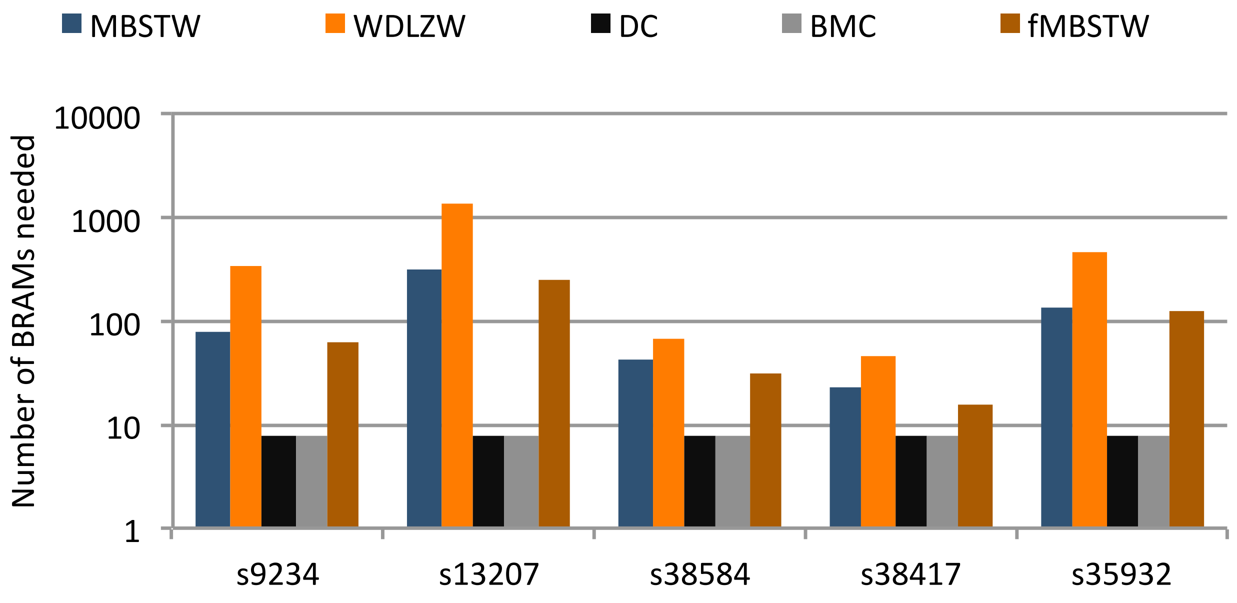


Figure 7-6. BRAM requirements

example, putting a value of $l = 2\%$ in Equation 7–7 will result in a compression penalty of less than 2%, which matches in the figure for all the benchmarks. We have conducted similar experiments for BMC based compression technique as well. The results in Figure 7-8 shows that the compression penalty also follows Equation 7–7.

Now, we would like to verify the last part of the discussion in Section 7.1.3, that is, the change in compression penalty with error rate for fMBSTW. We have conducted similar experiments and the results are shown in Figure 7-9.

An important observation here is that the change in penalty is sharper than the case of Figure 7-7 or Figure 7-8. This is quite obvious as per the discussion in Section 7.1.3, since in fMBSTW, 2 strings are affected when an error is introduced, whereas in DC or BMC, only 1 string is affected.

Finally, we compare how the introduction of errors affect the compression performance in cases of MBSTW and fMBSTW. The results are shown in Figure

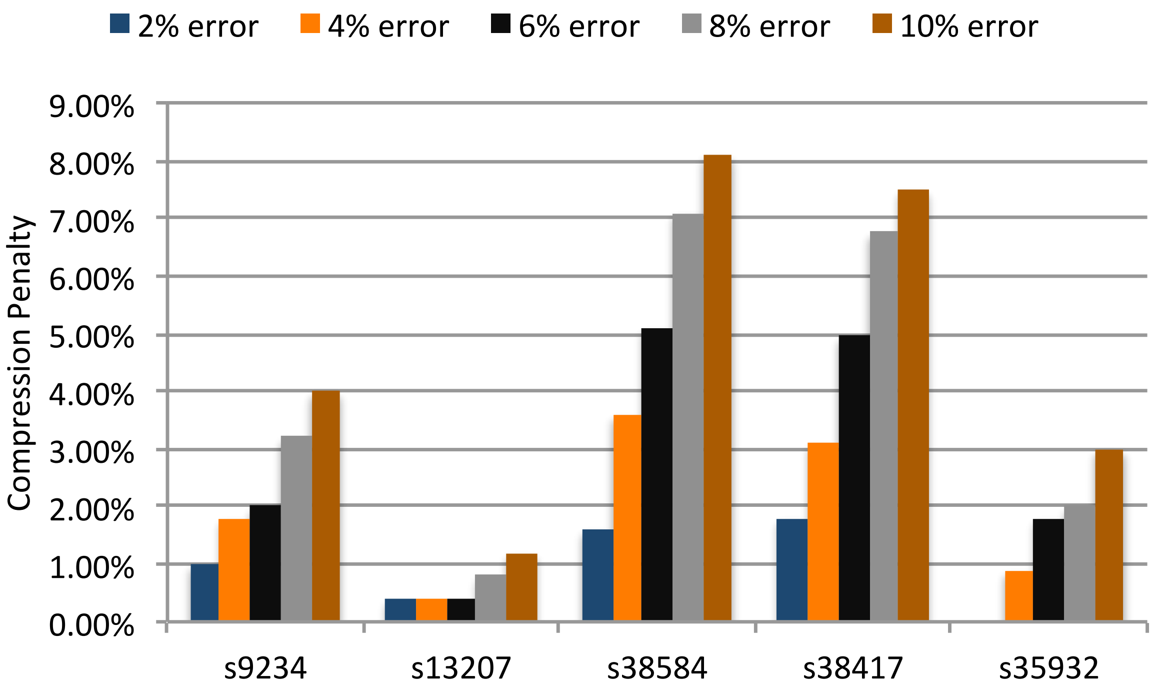


Figure 7-7. Comparison of compression penalty for DC

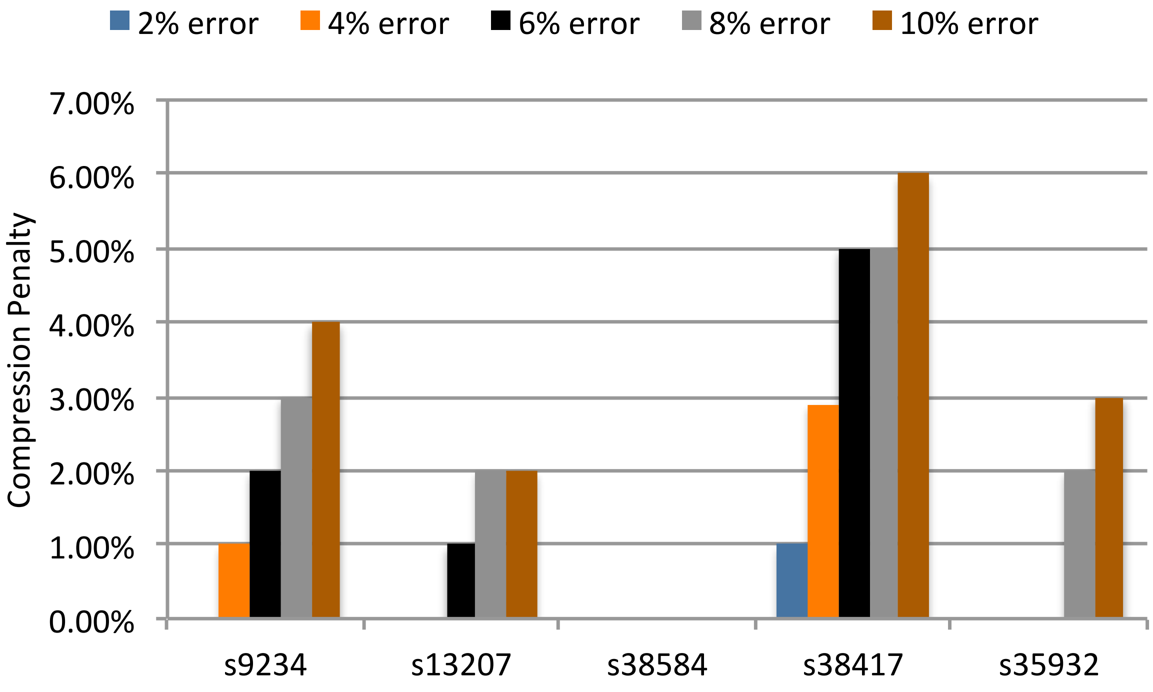


Figure 7-8. Comparison of compression penalty for BMC

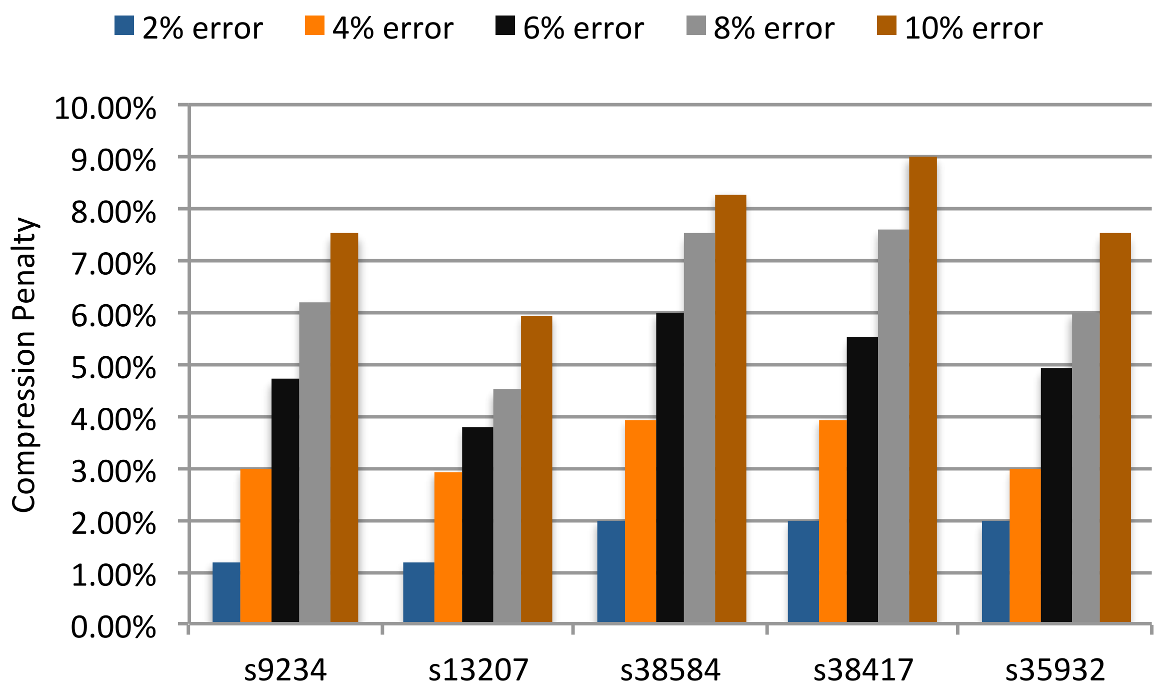


Figure 7-9. Comparison of compression penalty for fMBSTW

7-10. We have introduced 2% error for every benchmark's trace data. It can be seen that the compression penalty obtained using fMBSTW is always less than MBSTW, the maximum difference being 4% for s38417. The reason for higher penalty in MBSTW is that if error gets introduced early, MBSTW cannot benefit from a profitable sequence. In summary, our approach (fMBSTW) will perform significantly better irrespective of the percentage of errors in the dataset.

7.3 Summary

The trace buffer size is limited due to area/cost constraints. Trace data compression schemes have been popular which deals with dynamic dictionary based compression that enables to store more signal states using a fixed size trace buffer. We have proposed a trace data compression technique, which employs a statically computed dictionary. We have used three compression algorithms for compressing the trace data. Our approaches can produce up to 60% better compression performance, and

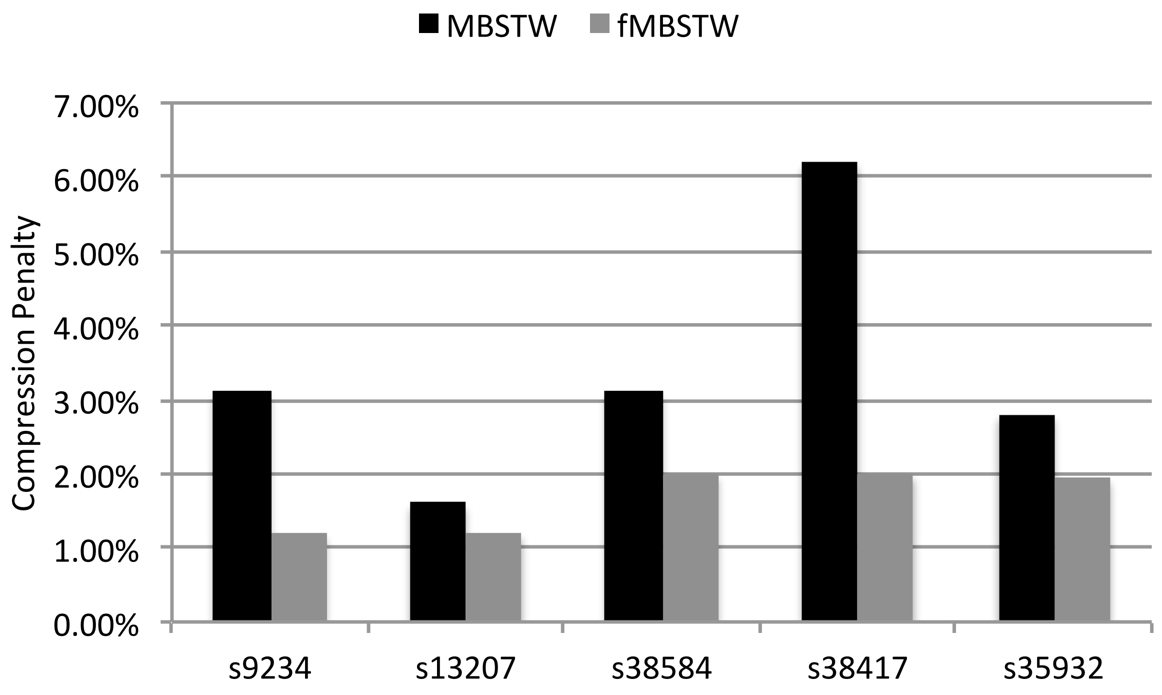


Figure 7-10. Comparison of compression penalty

reduce the compression hardware overhead up to 84% compared to best-known existing approaches.

CHAPTER 8
OBSERVABILITY-AWARE DIRECTED TEST GENERATION

Two major problems governing efficient error detection are the quality of input tests and selected trace signals. Chapters 3 - 6 primarily focused on improvement of signal observability. However, in order to detect errors, it is equally important to consider the controllability aspect of tests. During post-silicon validation, not all the primary outputs of a design block are visible (since they may be internally connected to some other components of the SoC design). Also, the number of primary outputs of a circuit is typically larger than the trace buffer width, which determines the number of signal states that can be stored per cycle. Hence, the primary outputs can not be used as observation points. Existing methods [5–7] on signal selection assume that the input tests are always random in nature. However, once the trace signals are known, Automatic Test Pattern Generation (ATPG) tools can be used to generate efficient tests for error detection if the probable error locations are available. In modern SoC design methodology, it is found that regions where errors are detected during pre-silicon verification are more likely to be erroneous during post-silicon validation. Therefore, the pre-silicon engineer can provide information about the probable erroneous locations for post-silicon validation. This information is extremely essential in determining the input tests as well as the trace signals. The inter-dependence of input tests and trace signals is shown in Figure 8-1.

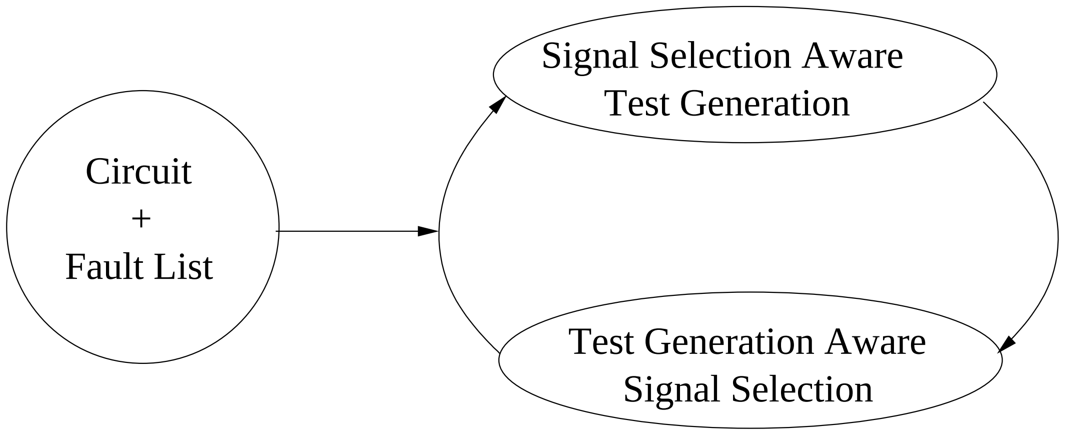


Figure 8-1. Outline of proposed technique

Our proposed approach takes as input the circuit and the trace buffer width. If the trace buffer width is *n*, our goal would be to select *n* best trace signals and determine their corresponding test cases, so that error detection is maximized. As an input, we also consider the probable erroneous locations in the circuit. This information is provided by the pre-silicon engineer. In the first step, we run the ATPG tool considering the primary outputs as observation points to obtain a set of directed tests for error detection. Next, we use these tests to determine the profitable trace signals.

The ATPG tool is used to generate tests using the new trace signals as observation points and obtain the fault coverage. This process is repeated, that is, the tests generated are re-used to determine a new set of observation points (trace signals). The ATPG tool is used to determine the fault coverage and produce new set of tests. If the fault coverage does not improve, the old set of observation points are selected for tracing and the corresponding tests as input tests. On the other hand, if the improvement is significant, the entire process is repeated. This process continues until the fault-coverage reaches 100% or does not improve in subsequent runs. The framework of our proposed approach is shown in Figure 8-2.

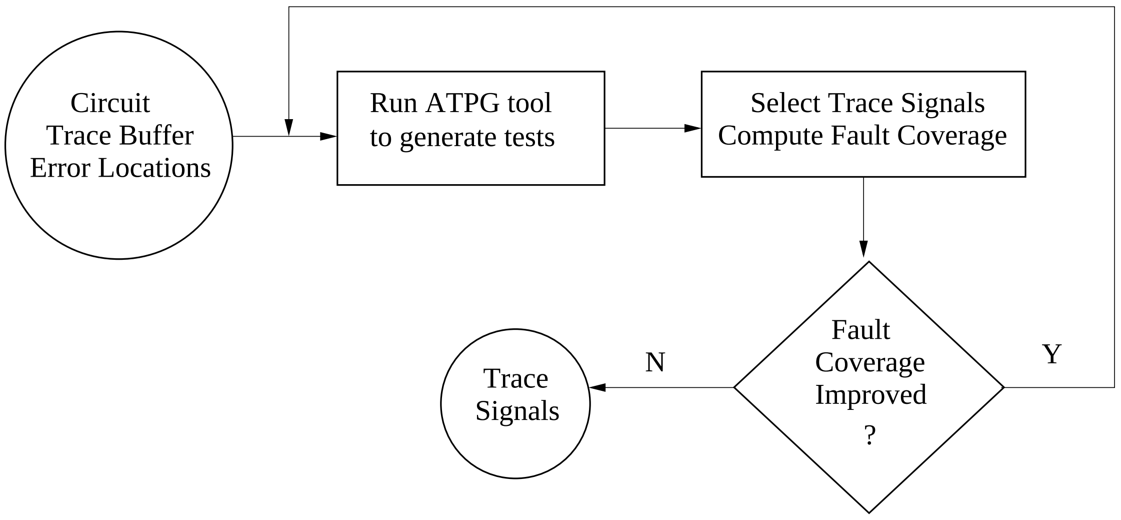


Figure 8-2. Observability-aware test generation flow

In this chapter, we consider test generation in the presence of electrical errors for both soft errors and crosstalk fault. Section 8.1 describes our test-aware signal selection technique. Section 8.2 presents our trace signal aware test generation approach.

8.1 Test-aware Signal Selection

Once the tests are determined using ATPG tool, the next set of trace signals need to be determined to improve the error detection performance. In general, during selection of trace signals, the input tests are assumed to be random. We would like to look at a special case when the input test sets are known prior to signal selection. Knowledge of input tests can be used to determine the signals very efficiently, specially, when the main focus is error detection. Our signal selection procedure is presented in Algorithm 12. The remainder of this section describes the three important steps of the signal selection algorithm in detail.

Algorithm 12: Test aware signal selection

```
Input: Circuit, Trace Buffer Width, Test Set $T$
Output: Trace Signals
for Each Test Vector do
    1: Simulate each fault in the circuit.
    2: For each signal in the circuit, determine whether it can detect the fault.
end
3: Compute the error detection ability of each signal.
while Trace buffer width is not reached do
    4: Find the signal with the highest Error Detection Ability and select it.
    5: Remove overlap.
end
Return Selected trace signals.
```

8.1.1 Fault Simulation

The best way to know whether a fault can be detected using a particular observation point and a test vector is to simulate the fault and notice the state of the observation point. Since we already have the set of test vectors, the fault simulation is straightforward. For each test vector, we first do a simulation without any error and observe the correct states of the various signals. Now, we perform simulation for every fault with the same test vector. For each fault, the signal states of the circuits are observed. If they are different from the ideal simulation, it is obvious that the fault is propagated to that signal. This process is repeated for each test case and each fault. For example, if there are m test vectors and n faults, there will be a total of $m \times n$ simulations. For each signal, we note the faults that it can detect. This is recorded as a binary variable EPP. For example, if in Figure 8-3, c can detect an error in a using any of the test vectors, $EPP_{c,a} = 1$. On the other hand, since d can never detect any error in a, $EPP_{d,a} = 0$.

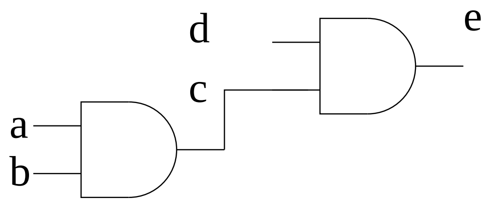


Figure 8-3. Example circuit

8.1.2 Error Detection Ability Computation

Error Detection Ability (EDA) of a node (signal) is a measure of the errors that a particular node can detect. A node can only detect errors in its fan-in cone. For example, if we consider Figure 8-3, any error in c can only propagate to e and not to a, b or d. Therefore, the only nodes whose errors c can detect are a and b. EDA of a node is the sum of all the errors that are detected using fault simulation.

$$EDA_c = EPP_{c,a} + EPP_{c,b} \tag{8–1}$$

It should be noted that a node can detect an error using multiple test cases, however, it should be counted only once. We enforce this by ensuring that EPP is a Boolean number. For example, if by simulating 2 test cases, c can detect a and b in both cases, EDA_c would be 2 and not 4. Once EDA value for each node is computed, the node with the highest EDA value is selected for tracing. The next section describes how to remove the overlap of already selected signals before determining the next profitable signal.

8.1.3 Overlap Removal

This part of the signal selection algorithm is used to remove effects of already selected signals and thus, select appropriate signals for improved error detection. In order to explain this, let us again get back to Figure 8-3. Let us consider node c, which is the first node to be selected for tracing. If $EPP_{c,a} = 1$ and $EPP_{c,b} = 1$, that is, the errors in a and b can propagate to c, contributions of $EPP_{e,a}$ and $EPP_{e,b}$ should not be included while computing EDA_e. In this case,

$$EDA_e = EPP_{e,c} + EPP_{e,d} \tag{8–2}$$

Thus, overlapping nodes, whose contributions have already been accounted for, should not be taken into account when computing the EDA value of a node. The process of overlap removal and signal selection continues until the trace buffer is full.

8.2 Trace Signal Aware Test Generation

Once the set of selected signals are known, the next step would be to generate another set of tests based on these signals that can maximize the error detection ability. The ATPG tool is used for this purpose. We used ATALANTA as an ATPG tool that generates tests depending on the fault list and considering the output nodes as observation points. In order to generate tests based on the selected trace signals, we modify the netlist to replace the trace signals as observation points. The ATPG engine will generate the tests assuming the trace signals as the observation points. The fault coverage using these new set of tests is computed. If they do not improve, the set of selected signals are reported as trace signals. The tests generated using the ATPG tool are used as directed input tests. Otherwise, the process in Section 8.1 is repeated.

8.2.1 Soft Errors and Faults

Soft errors are caused due to ionizing radiations from radioactive impurities present in a chip during manufacture. These may result in ionizing radiations like alpha-particles. When these alpha particles come in contact with a semiconductor, their kinetic energy gets converted to electrical energy [56], which results in creation of a large number of free electrons and holes. This leads to a creation of an inversion layer as well as a voltage glitch on the affected transistor. If the glitch is of sufficient magnitude, a faulty logic value is introduced temporarily on a node in the circuit. This is known as Single Event Transient (SET). If the faulty value is propagated to a primary output, the event is known as Single Event Upset (SEU). We try to generate directed tests to detect all SETs resulting in possible SEUs.

Traditionally, soft errors are assumed to affect memory elements since they contain the maximum density of bits susceptible to soft errors. Various mechanisms have been developed to protect the memory elements using Error Correcting Codes (ECC). However, with decrease in feature size and increase in design complexity, combinational circuits are equally vulnerable to soft-errors [31]. Protection of combinational elements is more expensive in terms of chip area, power and performance issues compared to memory elements. Hence, it is important that faults in combinational circuits due to soft errors are detected early. A soft-error may be masked inherently (that is, not propagated along the circuit) due to the following factors.

- **Logical Masking.** This occurs when a particle hits an input of a gate, and one of the other inputs have a controlling value. In this case, the controlling input will dominate the propagated value and hence, the erroneous value introduced by the soft-error will never be propagated. Let us consider the example circuit in Figure 8-3. Let a soft-error affect the node a of the circuit. Here, a is an input to an AND gate whose other input is b and output is c. Now, when the error is introduced at a, if the value of b is 0, that is, if b is the controlling input of the AND gate, the error value will never get propagated to the output c. Thus, the error will be masked.

- **Electrical Masking.** If the error introduced due to a particle strike is suppressed by electrical properties of subsequent logic gates, it will never reach an output and hence is masked.

- **Latching-Window Masking.** If the error reaches a latch at a cycle when the latch is not accepting its input value, it will never propagate through the latch and hence will be masked.

These masking effects result in lower soft-error rates in combinational circuits compared to memory elements. However, with decrease in feature size, transistors become faster and hence, electrical masking is reduced. Also with deeper pipeline, processor clock rates increase, with a subsequent increase in sampling rate of latches. As a result, effect of Latching-Window masking also decreases. Therefore, effect of soft-errors on combinational circuits have become more prominent these days.

The error model that is used for modeling soft-errors is a simple stuck-at fault model. The nodes affected by radiations get stuck at certain fixed values depending on the amount of free electrons or holes created. The effect of soft errors on a node value depends on the following factors:

- **Output Capacitance.** A weaker node capacitance allows it to store less charge and hence faster discharge. Therefore, the node is more susceptible to soft errors.
- **Pull-up network.** If a node has weak pull-up network, its logic 1 value can be easily changed to logic 0 by SET, thus leading to a stuck-at-0 fault.
- **Pull-down network.** If a node has a weak pull-down network, its logic 0 value can be flipped to logic 1 by SET, thus, leading to a stuck-at-1 fault.

Detection of soft errors require generation of test cases that would activate the particular errors and propagate them to the primary outputs of the circuit. As discussed before, during post-silicon validation, not all the primary outputs of a block may be visible since only some of the internal signals of the chip are traced. Therefore, in order to properly detect the faults, the erroneous values should propagate towards the traced signals and not to the primary outputs. The test generation problem should focus on generating a set of test cases that would activate and propagate a maximal number of soft errors (if possible, all of them) to the observation points, that is, the trace signals.

Let us consider the example in Figure 8-4 to explain the test generation problem for soft errors. The two error points are P and Q, where the errors can be represented as $s-a-0$ and $s-a-1$, respectively. We would like to generate tests that would activate them as well as propagate them to the primary outputs. The 5 input signals are named

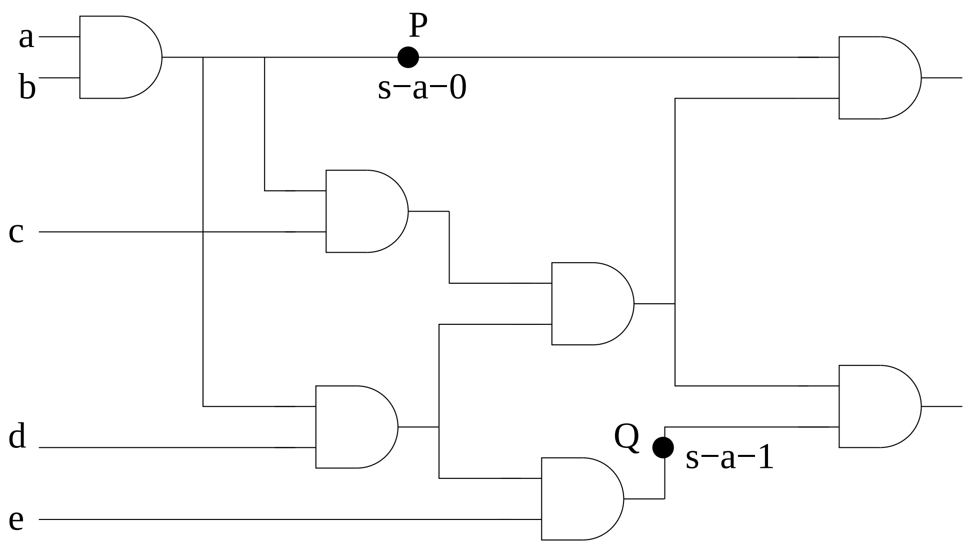


Figure 8-4. Example circuit illustrating test generation for soft errors

a, b, c, d and e. To detect the $s - a - 0$ fault at P, the input tests should be $< 1, 1, 1, 1, X >$ whereas the test required to detect the $s - a - 1$ fault at Q, the input tests should be $< 1, 1, 1, 1, 0 >$. Therefore, the test that can detect both faults is $< 1, 1, 1, 1, 0 >$. ATPG algorithms can be designed to generate tests that would detect these faults.

Algorithm 13 describes our test generation procedure for soft-errors. In the first step, we identify all the internal signals (gate signals and fan-out branches) in the zone. For each of these signals, we perform test generation for $s - a - 0$ and $s - a - 1$ faults using ATPG.

Algorithm 13: Test generation for soft error detection

```
Input: Circuit, Trace signals, Soft-error affected zone $Z$
Output: Test set to detect the faults
1: Find the signals corresponding to $Z$.
Signals corresponding to nodes as well as fan-out signals.
2: Create a fault list with stuck-at-0 and stuck-at-1 at each node.
3: Use ATPG to generate tests for these faults.
Use the trace signals as observation points.
Return the set of tests.
```

We developed a simulator to check whether the tests developed could actually excite and propagate the errors. As an ATPG tool, we use ATALANTA, which was developed at the CAD lab at Virginia Tech. ATALANTA takes as input the circuit as a netlist and the fault list, and generates test sets that would help detect the errors. It also gives an estimate of the percentage of faults that are covered. If the signals to be traced are already known and we want to detect the faults at only those points and not at the primary outputs, we would have to modify the netlist. ATALANTA will always force the errors to propagate towards the primary outputs. Hence, it is necessary to replace the primary outputs in the netlist with the trace signals. ATALANTA would now be able to generate tests that propagate the errors towards the trace signals.

8.2.2 Crosstalk Faults

Crosstalk faults are caused by parasitic coupling capacitances between adjacent lines in a chip [40]. With decrease in feature size, effect of coupling capacitances and hence, crosstalk faults become more prominent, thus leading to signal integrity problems [43]. Crosstalk faults are caused when the coupling capacitance between two lines exceed a certain threshold. In such a case, if there are transitions on either or both the lines, the transition on one will influence the other and hence, the voltage levels change causing either a delay or a glitch. The line whose voltage level changes is known as victim, while the line which changes the voltage level is called aggressor. We will explain crosstalk glitches and delays using the example circuit in Figure 8-5 which has 5 lines (signals), namely a, b, c, d and e. Let us assume the coupling capacitances between lines c and d exceed the threshold so that they can act as probable aggressor-victim pairs.

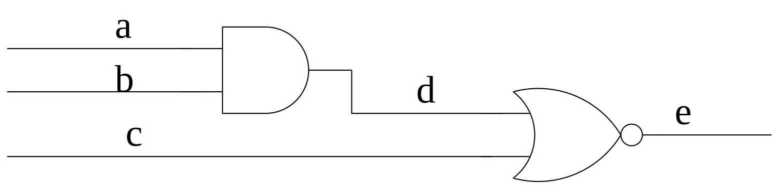
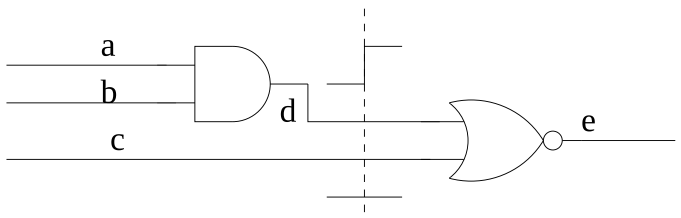
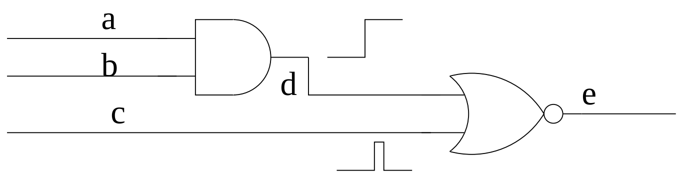


Figure 8-5. Example circuit illustrating crosstalk faults

During crosstalk glitch, the victim line stays at a static state, while the aggressor undergoes a transition. If the transition effect is opposite to the state of the victim, a glitch is created. For example, if the victim is at a state of 0, while the aggressor has a positive transition, a positive glitch is formed on the victim line. Similarly, if the victim line is in a state of 1 and a negative transition is formed on the aggressor line, a negative glitch is created. Figure 8-5 has been redrawn in Figure 8-6A to show that when line *c* is in a steady state and line *d* transits, a positive glitch on line *c* is formed as shown in Figure 8-6B.



A – Source of crosstalk glitch



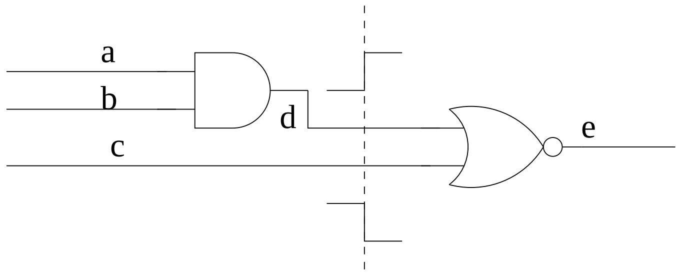
B – Crosstalk glitch

Figure 8-6. Positive glitch on *c*

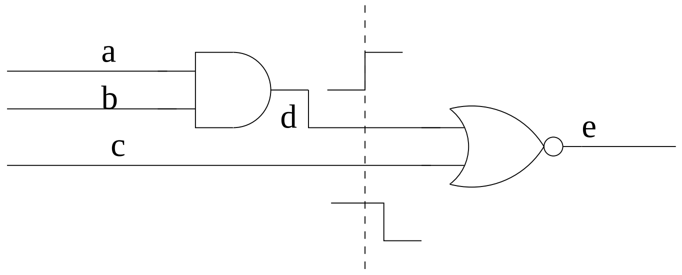
On the other hand, delays are created when both aggressor and the victim undergo transition. If the transitions are in the same direction, the overall delay is reduced. If the transitions are in opposite direction, the signal propagation delay is increased. Figure 8-7B shows a positive delay on line *c* due to transitions on both lines *c* and *d* (Figure 8-7A).

It should be noted that both the transitions need to be simultaneous in order for the delay to take effect. As can be seen in Figure 8-8, if the two transitions are not simultaneous, there will not be any delay.

The effect of crosstalk fault, that is, delay or glitch will be propagated to fan-out gates. In case of sequential circuits, if the glitch duration or delay is less than the clock



A Source of crosstalk delay



B Crosstalk delay

Figure 8-7. Positive delay on *c*

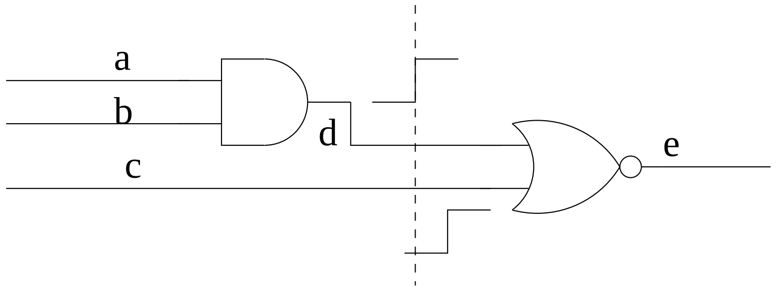


Figure 8-8. Non-simultaneous transitions

frequency, it gets suppressed. However, for combinational circuits, the effects get propagated to the outputs.

The test generation algorithm for crosstalk faults is shown in Algorithm 14. The first step of the algorithm is to find all the aggressor-victim pairs by observing their coupling capacitances. In this case, we consider single aggressor-single victim pairs only. However, the algorithm can be extended to multiple aggressors as well. Information on coupling capacitances is obtained from the layout information of the chip. Once we have identified all the pairs, the next step would be to generate tests that would provide transitions on either or both the lines depending on the desired type of crosstalk effect.

```
Algorithm 14: Test generation for crosstalk fault detection
    Input: Circuit, list of coupling capacitances, threshold
    Output: Test set to detect the faults
    1: Find all the pair of lines that contribute to crosstalk faults.
    2: Duplicate the circuit.
    3: Use ATPG to generate tests for these faults.
    Return Test set.
```

We now explain our algorithm using crosstalk delay, that is, transitions should be present on both lines. Crosstalk glitches can be explained in a similar way. Duplication of circuit is needed to create transitions on both aggressor and victim. For a combinational circuit, which does not have a clock signal, in order to emulate a transition, we need to make sure that the signals on a particular line change in two adjacent time units. Let us consider the example circuit in Figure 8-5. Suppose, lines c and d have been identified as crosstalk pairs. We would like to generate two sets of tests, such that they fire transitions on both these lines. If we want to observe the effect of crosstalk glitch, transition should be enabled on only one line. In order to generate the transitions, we have duplicated the circuit in Figure 8-9. Corresponding to each signal in Figure 8-5, there is a corresponding signal in Figure 8-9. For example, signal a in Figure 8-5 will be duplicated as $a\prime$ in Figure 8-9. Thus all the inputs are duplicated as well. The ATPG is used to generate the tests for this duplicated circuit; hence, it generates 2 tests for the original circuit, one corresponding to each set of inputs. In this example, the inputs to a, b, c will correspond to the test in the first time frame, while inputs to $a\prime, b\prime, c\prime$ will correspond to test in the second time frame. Thus, in order to generate a transition at line c in Figure 8-5, the inputs to c and $c\prime$ should be different in Figure 8-9. This can be forced by connecting an exclusive-or gate, whose two inputs are c and $c\prime$. Since an exclusive-or gate will be 1 only when the two inputs are different, this ensures a transition in line c in Figure 8-5. Similarly, d and $d\prime$ in Figure 8-9 are input to another exclusive-or gate, thus, forcing a transition in d in Figure 8-5. We want to generate test cases that would provide transitions on both lines. This is ensured by connecting an AND gate at the output of the two XOR gates.

The ATPG is then used to generate tests so that the output o of the AND gate is 1. This ensures transition on both lines. The ATPG tool can be run assuming the point o is $s-a-0$. In this case, the ATPG tool will generate test to force o to be 1, and hence ensure a transition on both lines.

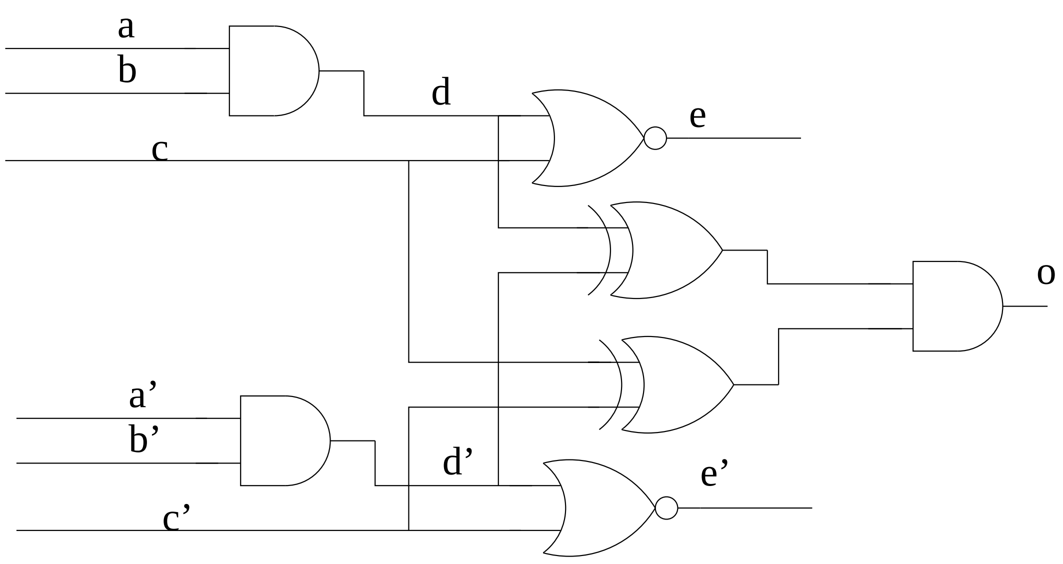


Figure 8-9. Duplicated circuit

We want the delay at the victim to be propagated to the output. In order to ensure that, o is connected to the fan-out branch of the victim and thus propagated to an observation point, or primary output. For example, if d (or $d\prime$) is the victim line in Figure 8-9, which incurs some delay, we add o to the fan-out cone of $d\prime$, in order to ensure that the delay in d in Figure 8-5 actually gets propagated to a primary output e ($e\prime$ in this case). The modified circuit is shown in Figure 8-10. If we have trace signals, the observation points (trace signals) are enabled such that the ATPG generates tests which propagate the delay to these trace signals. Similar test-generation procedure can be applied for crosstalk glitches, in which case, the transition should be only along the aggressors.

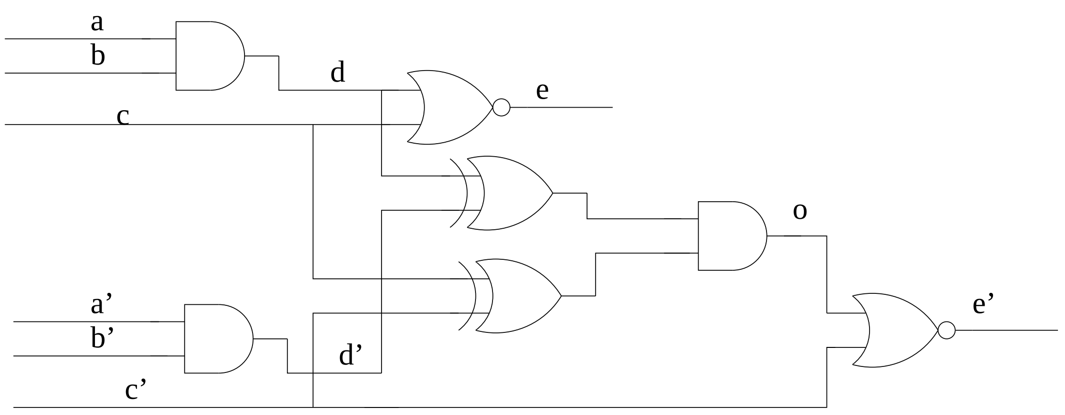


Figure 8-10. Modified circuit

8.3 Experiments

We have applied our proposed approach on the ISCAS ’85 combinational benchmarks. For the first set of experiments, soft errors are the only electrical errors considered present in the circuits. For each experiment, we applied 250 errors for each circuit. Random nodes are selected as error points. The ATPG tool, ATALANTA is used to generate directed tests in order to detect those faults.

An analysis of the memory requirement for our test generation algorithm described in Section 8.2 is shown in Table 8-1. The first column gives the name of the benchmarks while the second column provides the memory requirement to generate the directed tests in Kbyte. The third column indicates the size of each benchmark in KByte. The last column presents the number of reduced test sets.

Table 8-1. Memory requirement for test

Circuit	Memory Requirement (KB)	Size (KB)	Reduced Tests
c7552	27136	81317	29
c6288	18640	55155	10
c5315	15894	55298	26
c3540	23683	36632	45
c2670	33947	29601	33

We compare the performance of our test-aware signal selection algorithm with the standard profile based signal selection algorithm, where the inputs are all assumed to be random. A set of 250 points are selected in each circuit as potential erroneous regions. A simulation of 1000 cycles is run assuming no error is present. The input to the circuit is fed with the test sets generated by ATALANTA. Then, for each error, another set of 1000 simulations are performed assuming the error is present in the circuit. If any of the traced signal states during this simulation is different from the perfect simulation, an error is said to be detected. The Error Detection Ratio (EDR[1]), is chosen for comparing error detection performance. The comparison of EDR for 5 of the largest ISCAS '85 benchmarks is shown in Figure 8-11. The proposed method column refers to our proposed test-aware signal selection algorithm. On the other hand, the profile based column refers to standard profile-based trace signal selection algorithm. It should be noted that to make a fair comparison, we have used the same set of tests in both scenarios. In profile-based case, the primary outputs are used as observation points.

As can be seen in Figure 8-11, our proposed method performs consistently better than the profile-based signal selection algorithm, with maximum improvement being 57% for $c5315$. This is as expected, since our algorithm uses tests as inputs to select signals, which gives a better insight during error propagation probability computation, and hence, subsequent selection of trace signals to detect errors.

We would like to observe the variation of EDR with trace buffer width for soft error detection. The trace buffer width has been increased in steps of 16, 32 and 64, respectively. The results are shown in Figure 8-12. For most cases, except $c5315$, 100% EDR is reached when trace buffer width is 32.

Next, we investigate how our proposed approach described in Figure 8-2 converges for the 5 benchmarks. The results are presented in Table 8-2. For each of the benchmark, we choose a trace buffer of size 32. The error set remains the same (250 soft errors that we considered in this section). The first column gives the circuit name. The second

[1] Defined earlier in Equation 5–4.

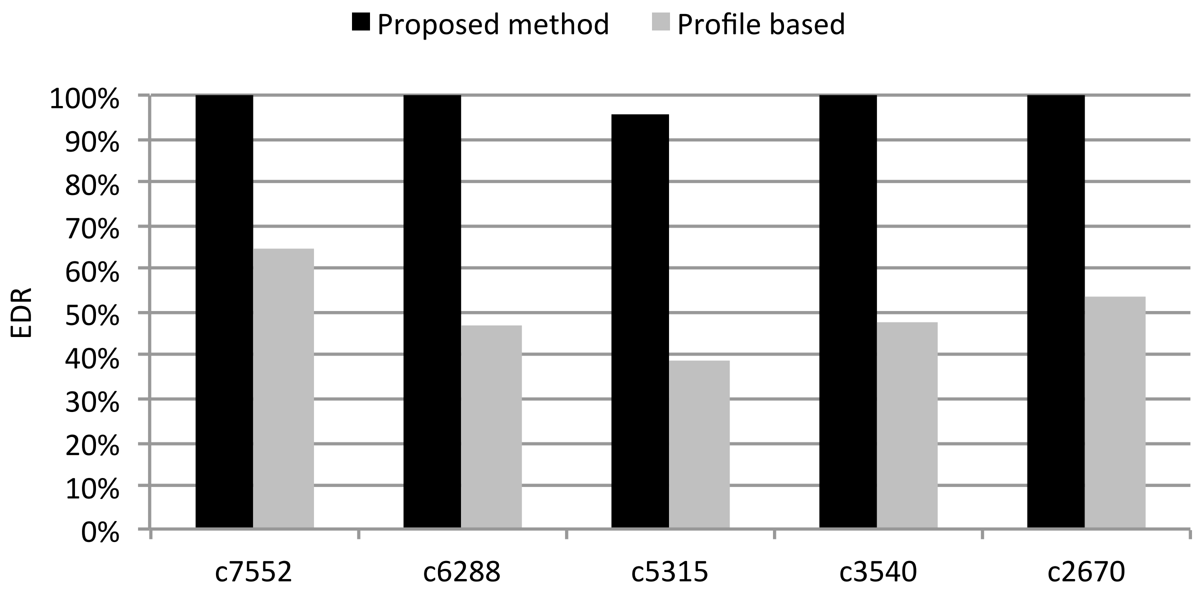


Figure 8-11. Comparison of signal selection methods for soft errors

column presents the number of runs needed to converge, that is the number of times the signal selection and test generation procedures are executed in Figure 8-1. The last column refers to the final coverage obtained using directed tests. It can be seen that for most benchmarks, two runs are sufficient to reach a steady state for both tests and selected signals. This proves the efficiency of our test aware signal selection algorithm, which selects the trace signals wisely so that the fault coverage reaches 100% within two iterations. The ATPG tool generated directed tests that enable the errors to propagate towards the trace signals. On the other hand, existing trace signal selection algorithms rely on random inputs as test. Hence, they provide significantly lower fault coverage as shown in Figure 8-11.

Now, we would like to observe the performance of our test-generation algorithm for crosstalk faults described in Section 8.2.2. Similar to Figure 8-11, we have used the 5 largest ISCAS '85 benchmarks. The test generation algorithm for crosstalk faults described in Algorithm 14 is extremely time consuming as it requires manual modification of the circuit in order to insert the additional AND and XOR gates

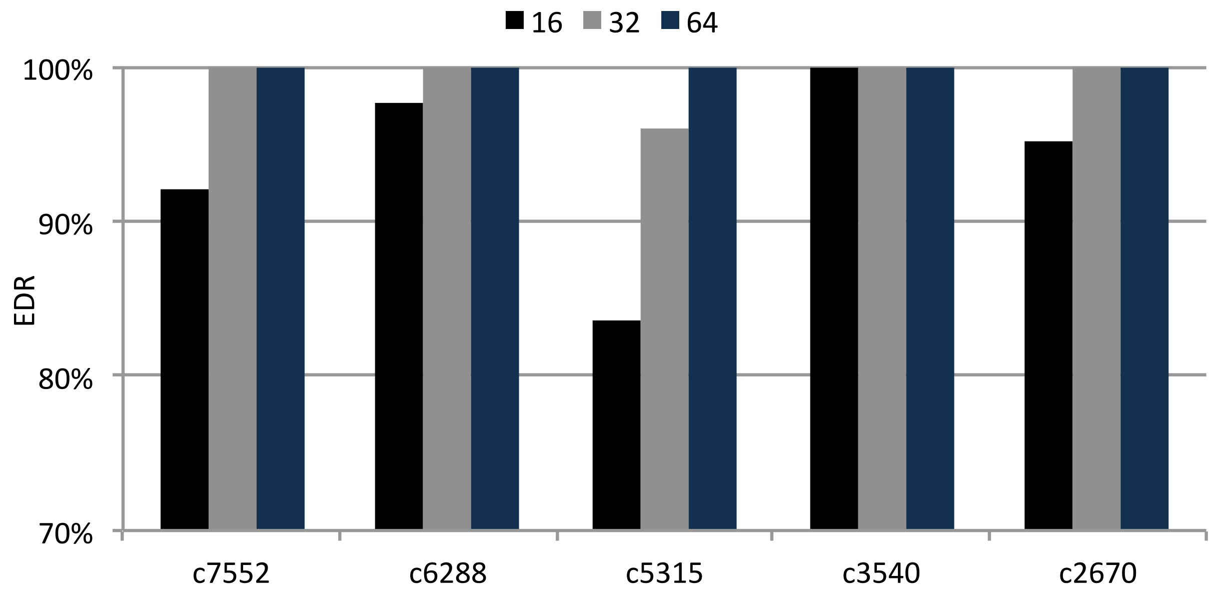


Figure 8-12. Variation of EDR with trace buffer width for soft error detection

Table 8-2. Fault coverage in case of soft errors

Circuit	Number of Runs	Final Coverage
c7552	2	100%
c6288	2	100%
c5315	3	99.2%
c3540	2	100%
c2670	2	100%

described in Figure 8-10. Hence, we reduced the number of faults from 250 to 10. Similarly, the trace buffer width is also reduced to 4 instead of 32, that is, in this case, 4 signals will be stored every cycle. In order to make fair comparison, the same experiment is repeated for profile-based signal selection technique using the same set of parameters. The results are shown in Figure 8-13. As can be seen, our proposed method provides significant improvement over the profile-based technique, with the maximum improvement being 100% for $c3540$. The performance of the profile-based method is seen to be degraded compared to soft errors as seen in Figure 8-11. The reason behind this is the limited number of errors chosen (10). These comprise of less than 1% of the total number of signals in the design. Hence, using the profile-based method with limited number of trace signals (4), it will be extremely difficult to capture the faults.

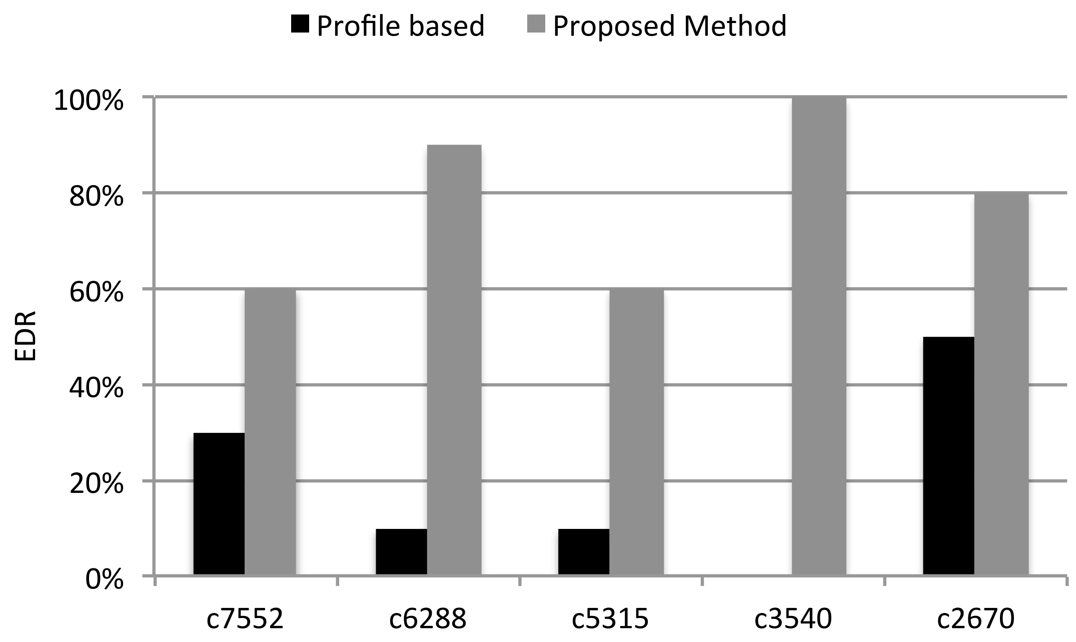


Figure 8-13. Comparison of signal selection methods for crosstalk faults

Similar to Table 8-2 for soft errors, we want to see how fast the loop in Figure 8-2 can converge in case of crosstalk faults. The parameters remain the same as in Figure 8-13, that is, 10 error points with trace buffer of width 4. The results are shown in Table 8-3. Similar to soft errors, it can be seen that 2-3 runs are sufficient for obtaining the maximum coverage possible. This proves the efficiency of our test generation techniques and test-aware signal selection algorithms for crosstalk faults.

Table 8-3. Coverage of crosstalk faults

Circuit	Number of Runs	Final Coverage
c7552	2	60%
c6288	3	100%
c5315	2	60%
c3540	2	100%
c2670	3	90%

8.4 Summary

Limited observability is a major bottleneck in detecting errors during post-silicon validation and debug. In this chapter, we have proposed an efficient observability-aware directed test generation technique that selects efficient observation points and generates corresponding test sets to improve detection of electrical errors. We have implemented our approach using ATLANTA ATPG tool and applied on ISCAS ’85 benchmarks. Up to two times improvement in error detection has been observed compared to the existing signal selection approaches.

CHAPTER 9
CONCLUSIONS AND FUTURE WORK

Post-silicon validation is an important component of modern chip design methodology. Limited observability is a major concern during post-silicon debug. This dissertation proposed efficient techniques to enhance the observability during post-silicon debug to reduce overall validation effort. This chapter concludes this dissertation and outlines future research directions.

9.1 Conclusions

Due to dramatic increase in design complexity and decrease in time-to-market window, a lot of bugs escape the pre-silicon verification phase and get manifested during the normal operation of the chip. Post-silicon validation is used to detect these bugs before a chip is delivered to the customer. A major challenge during post-silicon validation is the limited observability of on-chip signals. Some recent techniques help to trace some of the internal signal states and store them in an on-chip trace buffer for future debug. The size of the trace-buffer determines the signal states that can be stored and hence, provides a constraint on signal observability. To improve observability, this dissertation made several important contributions as summarized below.

In Chapter 3, we developed efficient techniques to select trace signals in order to improve the restoration of the untraced signals. Existing approaches use a partial restorability based signal selection techniques, that are inferior both in terms of restoration ratio and signal selection time. We have proposed a total restorability based signal selection algorithm that provides up to 3 times better restoration performance while reducing the signal selection time by an order-of-magnitude. We have further proposed an RTL-level signal selection algorithm that reduces the memory and time overhead considerably with minor impact on the restoration performance.

In Chapter 4, we proposed a combined trace and scan based approach to improve signal restoration. Scan based debug mechanisms have been used in manufacturing-testing domain for a long time. While trace signals provide a good temporal visibility, scan signals can improve the spatial observability. We have provided an efficient combination of both in order to improve the overall observability along both directions. Our proposed technique provides up to 17% better restoration compared to existing trace-scan combined approaches.

The existing signal selection techniques focus on improving the overall signal restoration in the circuit. However, restoration may not be directly related to detection of errors in the circuit; since errors move only along the fan-out cone of a signal, while restoration can proceed in both directions. In Chapter 5, we have proposed a signal selection technique that helps in detecting errors across the circuit. While previous approaches primarily rely on statically selected signals, Chapter 6 presents a dynamic signal selection technique that would be beneficial in a wide variety of scenarios including when only a set of regions in a design is important from debug perspective. Our proposed technique selects signals dynamically depending on the currently active regions in the circuit as well as the probable error locations. Experimental results demonstrate that our proposed method can detect up to 3 times more errors compared to static signal selection approaches.

The trace buffer size constraints the number of signals that can be stored during post-silicon debug. If the trace data are compressed before storing in the trace buffer, larger number of signal states can be stored using the same trace buffer size. Existing trace signal compression algorithms relied on dynamic dictionary generation which may not select the best dictionary entries, and hence, may provide a poor compression performance. In Chapter 7, we have proposed a dynamic compression algorithm based on static dictionary, which improves the restoration performance and reduces the hardware overhead associated with it. Our proposed technique can provide up to 60% improvement in restoration performance, while reducing the hardware overhead by 84% compared to existing techniques.

In Chapter 8, we have proposed an efficient observability-aware test generation framework to improve both the controllability and observability during post-silicon validation. We have developed a test generation technique based on the observation points (trace signals). Based on the generated tests, we refine our selected set of observation points in order to enhance the overall error detection capability. We made two important contributions: i) trace signal selection based on input tests, and ii) test generation based on the selected signals. Our proposed method is found to provide significant improvement in error detection performance.

9.2 Future Research Directions

Post-silicon validation has emerged as an important concern in any chip design methodology. It is expected that various aspects of post-silicon validation and debug will continue to be challenging research problems in the development of future SoC designs. The research proposed in this dissertation can be extended in the following directions:

The proposed signal selection approaches considered designs with single clock domain and no gated clocks. However, actual circuits may have mutiple clock domains, with some of the clocks being derived from the signals in the circuit. The signal selection algorithms presented in this dissertation can be extended to accomodate these factors. Specifically, in case of circuits with gated clocks, some signals should be selected in order to reconstruct the derived clocks. Another underlying assumption of the proposed approaches is that there are only flip-flops and no latches. Flip-flops change their states only when transitions occur, while latches can change whenever a particular condition is reached. In other words, flip-flops are edge-trigerred while latches are level-trigerred. Proposed approaches can be extended to include the effect of latches and hence select the signals appropriately.

We have seen that signal selection at RTL-level reduces the time and memory overhead compared to gate-level signal selection. The overhead can possibly be further reduced by performing the signal selection at higher abstraction level, for example, at TLM (Transaction Level Model) level [57]. However, signal selection at higher abstraction level comes with the additional penalty of possible degradation in restoration performance. The signal selection algorithms should be modified in order to incur minimum penalty in overall restoration.

The signal selection algorithms presented in this dissertation, whether for restoration or error detection, assumed an empty trace buffer; that is, even the first signal to be traced has to be determined. It may be possible that the design engineer provides the validation engineer a set of signals that needs to be traced every cycle. These can be some important control signals, or signals which are highly prone to errors. The signal selection algorithms need to be modified to utilize the information about the signals already provided in order to select the subsequent signals for the trace buffer.

The observation-aware test generation method described in Chapter 8 deals with electrical errors. However, logical errors are also equally important during post-silicon validation. It is therefore necessary to develop similar test generation strategies for logical and functional errors. The approach proposed in Chapter 8 can be modified to include logical errors. A generic high-level test-generation framework needs to be developed that can take into account different types of errors and faults described in Section 1.1.

The trace data in Figure 1-2 is transferred from the logic to the trace buffer via a set of interconnection fabrics. These fabrics are extremely expensive in nature. Since they do not directly contribute to the functionality of the chip except validation, their cost should be kept at minimum. Efficient signal selection algorithms can be used to reduce the cost of these fabrics. Knowledge of the trace signals as well as the layout information of the chip can help us develop efficient routing algorithms to minimize the cost of data transfer to the trace buffer.

REFERENCES

[1] http://www.itrs.net. International Technology Roadmap for Semiconductors (ITRS).

[2] A. Nahir, A. Ziv, R. Galivanche, A. Hu, M. Abramovici, A. Camilleri, B. Bentley, H. Foster, V. Bertacco and S. Kapoor, "Bridging pre-silicon verification and post-silicon validation", in *DAC*, 2010, pp. 94–95.

[3] M.J. Howes and D.V. Morgan, "Reliability and degradation: semiconductor devices and circuits", in *Wiley-Interscience Journal*, vol. 1, pp. 454, 1981.

[4] J. Bateson, "In-circuit testing", Van Nostrand Reinhold, 1985.

[5] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug", *IEEE TCAD*, vol. 28, no. 2, 2009, pp. 285–297.

[6] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation", in *DATE*, 2009, pp. 1338–1343.

[7] K. Basu and P. Mishra, "Efficient Trace Signal Selection for Post Silicon Validation and Debug", in *International Conference on VLSI Design*, 2011, pp. 352–357.

[8] H.F. Ko and N. Nicolici, "Automated trace signals selection using the RTL descriptions", in *ITC*, 2011, pp. 1–10.

[9] R. Datta, A. Sebastine, and J. Abraham, "Delay fault testing and silicon debug using scan chains", in *ETS*, 2004, pp. 46–51.

[10] X. Gu, W. Wang, K. Li, H. Kim, and S. Chung, "Re-using DFT logic for functional and silicon debugging test" in *ITC*, 2002, pp. 648–656.

[11] G. J. Van Rootselaar and B. Vermeulen, "Silicon debug: scan chains alone are not enough", in *ITC*, 1999, pp. 892–902.

[12] J. Gao, Y. Han, and X. Li, "A New Post-Silicon Debug Approach Based on Suspect Window", in *VTS*, 2009, pp. 85–90.

[13] Y. Yang, N. Nicolici, and A. Veneris, "Automated data analysis solutions to silicon debug", in *DATE*, 2009, pp. 982–987.

[14] J. Yang and N. Touba, "Expanding trace buffer observation window for in-system silicon debug through selective capture", in *VTS*, 2008, pp. 345–351.

[15] E. Anis and N. Nicolici, "Low cost debug architecture using lossy compression for silicon debug", in *DATE*, 2007, pp. 225–230.

[16] E. Anis and N. Nicolici, "On using lossless compression of debug data in embedded logic analysis", in *ITC*, 2007, pp. 1–10.

[17] O. Caty, P. Dahlgren, and I. Bayraktaroglu, “Microprocessor silicon debug based on failure propagation tracing”, in *ITC*, 2005, pp.284–293.

[18] F. Koushanfar, D. Kirovski, and M. Potkonjak, “Symbolic debugging scheme for optimized hardware and software”, in *ICCAD*, 2000, pp. 40–43.

[19] F. M. De Paula, M. Gort, A. J. Hu, S. Wilton, and J. Yang, “Backspace: Formal analysis for post-silicon debug”, in *FMCAD*, 2008, pp. 1–10.

[20] N. Nataraj, T. Lundquist, K. Shah, “Fault localization using time resolved photon emission and stil waveforms”, in *ITC*, 2003, pp. 254–263.

[21] A. DeOrio, I. Wagner and V. Bertacco, “Dacota: Post-silicon validation of the memory subsystem in multi-core designs”, in *HPCA*, 2009, pp. 405–416.

[22] D. Josephson and B. Gottlieb, “The crazy mixed up world of silicon debug [ic validation]”, in *CICC*, 2004, pp. 665–670.

[23] M. Abramovici, P. Bradley, K. Dwarkanath, P. Levin, G. Memmi and D. Miller, “A reconfigurable design-for-debug infrastructure for SoCs”, in *DAC*, 2006, pp. 7–12.

[24] S. Prabhakar and M. Hsiao, “Using Non-Trivial Logic Implications for Trace Buffer-based Silicon Debug”, in *ATS*, 2009, pp. 131–136.

[25] W. Mao and R. K. Gulati, “Improving gate level fault coverage by RTL fault grading”, in *ITC*, 1996, p. 150–159.

[26] N. Yogi and V. Agrawal, “Spectral RTL test generation for gate-level stuck-at faults”, in *ATS*, 2006, pp. 83 –88.

[27] H. Ko and N. Nicolici, “Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging”,, in *ETS*, 2010, pp 62–67.

[28] S. Prabhakar and M. Hsiao, “Multiplexed trace signal selection using non-trivial implication-based correlation”, in *ISQED*, 2010, pp. 697 – 704.

[29] X. Liu and Q. Xu, “On multiplexed signal tracing for post-silicon debug”, in *DATE*, 2011, pp. 1 – 6.

[30] T.C. May and M.H. Woods, “Alpha-particle-induced soft errors in dynamic memories”, in *IEEE Transactions on Electron Devices*, vol. 26, no. 1, 1979, pp. 2 – 9.

[31] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger and L. Alvisi, “Modeling the effect of technology trends on the soft error rate of combinational logic”, in *DSN*, 2002, pp. 389 – 398.

[32] S. Mitra, N. Seifert, M. Zhang, Q. Shi and K.S. Kim, “Robust system design with built-in soft-error resilience”, in *IEEE Computer*, vol. 38, no. 2, 2005, pp. 43 – 52.

[33] M. Nicolaidis, “Time redundancy based soft-error tolerance to rescue nanometer technologies”, in *VTS*, 1999, pp. 86 – 94.

[34] P. Hazucha and C. Svensson, “Impact of CMOS technology scaling on the atmospheric neutron soft error rate”, in *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, 2000, pp. 2586 – 2594.

[35] A. Sanyal, S.M. Alam and S. Kundu, “A built-in self-test scheme for soft error rate characterization”, in *IOLTS*, 2008, pp. 65 – 70.

[36] A. Sanyal, K. Ganeshpure and S. Kundu, “On Accelerating Soft-Error Detection by Targeted Pattern Generation”, in *ISQED*, 2007, pp. 723 – 728.

[37] R. Anglada and A. Rubio, “Brief communication. Logic fault model for crosstalk interferences in digital circuits”, in *International Journal of Electronics Theoretical and Experimental*, vol. 67, no. 3, 1989, pp. 423 – 425.

[38] A. Rubio, J. Pons and R. Anglada, “A crosstalk tolerant latch circuit design”, in *Midwest Symposium on Circuits and Systems*, 1990, pp. 653 – 656.

[39] S. Kundu, S.T. Zachariah, Y.S. Chang and C. Tirumurti, “On modeling crosstalk faults”, in *IEEE TCAD*, vol. 24, no. 12, 2005, pp. 1909 – 1915.

[40] H. Takahashi, K.J. Keller, K.T. Le, K.K. Saluja and Y. Takamatsu, “A Method for Reducing the Target Fault List of Crosstalk Faults in Synchronous Sequential Circuits”, in *IEEE TCAD*, vol. 24, no. 2, 2005, pp. 252–263.

[41] W.Y. Chen, S.K. Gupta and M.A. Breuer, “Test generation for crosstalk-induced faults: framework and computational results”, in *Journal of Electronic Testing*, vol. 18, no. 1, 2002, 17 – 28.

[42] A. Sanyal, K. Ganeshpure and S. Kundu, “Test Pattern Generation for Multiple Aggressor Crosstalk Effects Considering Gate Leakage Loading in Presence of Gate Delays”, in *IEEE TVLSI*, vol. 20, no. 3, 2012, pp. 424 – 436.

[43] S. Chun, T. Kim and S. Kang, “ATPG-XP: Test Generation for Maximal Crosstalk-Induced Faults”, in *IEEE TCAD*, vol. 28, no. 9, 2009, pp. 1401 – 1413.

[44] E. Taylor, H. Jan and J. Fortes, “Towards Accurate and Efficient Reliability Modeling of Nanoelectronic Circuits”, in *IEEE-NANO*, 2006, pp. 395–398.

[45] *http://www.icarus.com/eda/verilog/*.

[46] S. P. Mohanty, N. Ranganathan, E. Kougianos and P. Patra, “Low-Power High-Level Synthesis for Nanoscale CMOS Circuits”, Springer Verlag, 2008.

[47] www.opencores.org

[48] N. Alawadhi and O. Sinanoglu, “Revival of Partial Scan: Test Cube Analysis Driven Conversion of Flip-Flops”, in *VTS*, 2011, pp. 260–265.

[49] J. Saxena, K. Butler, and L. Whetsel, “An analysis of power reduction techniques in scan testing”, in *ITC*, 2002, pp. 670–677.

[50] J. S. Yang and N.A. Touba, “Automated selection of signals to observe for efficient silicon debug”, in *IEEE VTS*, 2009, pp. 79 – 84.

[51] H. Shojaei and A. Davoodi, “Trace signal selection to enhance timing and logic visibility in post-silicon validation”, in *ICCAD*, 2010, pp. 68 – 72.

[52] D. Chatterjee, C. McCarter and V. Bertacco, “Simulation-based signal selection for state restoration in silicon debug”, in *ICCAD*, 2011, pp. 595–601.

[53] N. Tamarapalli and J. Rajski, “Constructive multi-phase test point insertion for scan-based BIST”, in *ITC*, 1996, pp. 649–658.

[54] K. Basu and P. Mishra, “Test Data Compression Using Efficient Bitmask and Dictionary Selection Methods”, in *IEEE TVLSI*, vol. 18, no. 9, 2010, pp. 1277–1286.

[55] K. Basu and P. Mishra, “A novel test-data compression technique using application-aware bitmask and dictionary selection methods”, in *ACM GLSVLSI*, 2008,pp.83–88.

[56] P. Dodd and L.W. Massengill, “Basic mechanisms and modeling of single-event upset in digital microelectronics”, in *IEEE Trans. on Nuclear Science*, vol. 49, no. 6, 2002, pp. 3100-3106.

[57] *http://www.accellera.org/home/*

BIOGRAPHICAL SKETCH

Kanad Basu has received his Ph.D. from the Department of Computer and Information Science and Engineering, University of Florida in 2012. He received his Bachelor of Engineering degree from Jadavpur University, Kolkata, India in 2003. His research interests include system level design, testing and verification. He has published several articles in peer reviewed journals and conferences. He has received the Best Paper Award at the International Conference on VLSI Design 2011. He is a recipient of the CISE departmental travel grant as well as the University of Florida International Center Outstanding Achievement award.