

Copyright
by
Joon-Sung Yang
2009

**The Dissertation Committee for Joon Sung Yang Certifies that this is the approved
version of the following dissertation:**

Enhancing Silicon Debug Techniques via DFD Hardware Insertion

Committee:

Nur A. Touba

Tony Ambler

Lizy K. John

David Z. Pan

Lili Qiu

Enhancing Silicon Debug Techniques via DFD Hardware Insertion

by

Joon Sung Yang, M.S.; B.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August, 2009

Dedicated to my family

Acknowledgements

I wish to express my heartfelt gratitude to Prof. Nur A. Touba. I believe I have been very fortunate to have him as my adviser during my graduate study. I would like to thank him for his tremendous guidance, support and encouragement. He was my mentor who helped me work on challenging problems, all the while showing me the correct path to follow as a researcher.

I express my gratefulness to my Ph. D. committee members - Prof. Tony Ambler, Prof. Lizy John, Prof. David Z. Pan and Prof. Lili Qiu for their input and support, and Prof. Sibum Sung who served as my committee member for the qualifying exam.

My deepest appreciation goes to Benoit Nadeau-Dostie at LogicVision for his technical discussions and Abhijit Jas at Intel, a former CAT (Computer Aided Testing), for his helpful advice. I especially would like to thank Jinkyu Lee who helped me become a CAT lab member. My time at University of Texas at Austin has been wonderful in the company of all my friends – Rudrajit Datta, Avijit Dutta and Richard Putman from CAT lab, Jae-Yong Chung, Kihyuk Han, Hur Jin, Kihwan Jun, Hongsuk Kim, Seyoung Kim, Wonsoo Kim, Inwook Kong, Taesoo Jun, Gahngsoo Moon, Jaewook Lee, Joohun Lee, Joonsung Park, Donghyuk Shin and Ick-Jae Yoon for their help and unstinted support. Some of them were really good mates for Winning 11. Especially, I thank Jungho Jo who has been my roommate. We were really good roommates. Also, I am grateful to Lunch Bank for lunch deliveries and Melissa Campos and Melanie Gulick for their kind and helpful administrative support.

My special thanks go, as always, to my girl friend, Soo Jin Park who has always stood beside me and cheered me up. Finally, I would like to thank my parents, brother

and sister in Korea for their endless love and support. They have been unstinting in my support all these years all the while constantly encouraging me in my efforts. I would not have been what I am, without their love and pray.

Enhancing Silicon Debug Techniques via DFD Hardware Insertion

Publication No. _____

Joon Sung Yang, Ph. D.

The University of Texas at Austin, 2009

Supervisor: Nur A. Touba

As technology is advancing, larger and denser devices are being manufactured with shorter time to market requirements. Identifying and resolving problems in integrated circuits (ICs) are the main focus of the pre-silicon and post-silicon debug process. As indicated in the International Technology Roadmap for Semiconductors (ITRS), post-silicon debug is a major time consuming challenge that has significant impact on the development cycle of a new chip. Since it is difficult to acquire the internal signal values, conventional debug techniques typically involve performing a binary search for failing vectors and performing mechanical measurement with a probing needle. Silicon debug is a labor intensive task and requires much experience in validating the first silicon.

Finding information about when (temporal) and where (spatial) failures occur is the key issue in post-silicon debug. Test vectors and test applications are run on first silicon to verify the functionality when it arrives. Scan chains and on-chip memories have been used to provide the valuable internal signal observation information for the silicon debug process. In this dissertation, a scan-based technique is presented to detect the circuit misbehavior without halting the system. A debugging technique that uses a

trace buffer is introduced to efficiently store a series of data obtained by a two dimensional compaction technique. Debugging capability can be maximized by observing the right set of signals to observe. A method for an automated selection of signals to observe is proposed for efficient selection. Investigation in signal observability is further extended to signal controllability in test point insertion. Noble test point insertion techniques are presented to reduce the area overhead for test point insertion.

Table of Contents

List of Tables	xii
List of Figures	xiii
Chapter 1 Introduction	1
1.1 Complete, But Non-Real Time Observation Technique.....	2
1.2 Selective, But Real Time Observation Technique.....	3
1.3 Contributions and Organization of the Dissertation	4
Chapter 2 Enhancing Silicon Debug via Periodic Monitoring	8
2.1 Introduction.....	8
2.2 Procedure for Configuring Multiple MISRs	10
2.2.1 Initial Clustering Procedure	12
2.2.2 Merge & Update Procedure	13
2.3 Three Step Debug Process	15
2.3.1 Step One : Checking Intermediate Parity.....	15
2.3.2 Step Two : Storing Parity of MISRs	18
2.3.3 Step Three : Storing MISR Signatures	18
2.4 Error Bypassing	19
2.5 Experimental Results	20
2.6 Conclusions.....	22
Chapter 3 Expanding Trace Buffer Observation Window for In-System Silicon Debug through Selective Capture	23
3.1 Introduction.....	23
3.2 Overview of Proposed Scheme	26
3.3 Details of the Three Debug Sessions	27
3.3.1 Sessions 1 – Estimating Expanded Observation Window Size ..	27
3.3.2 Session 2 – Determining Suspects	28
3.3.2.1 2-D Compaction.....	29
3.3.2.2 Tag Data Generation	31
3.3.3 Session 3 – Capturing Suspect Clock Cycles	35

3.4 Hardware Architecture of Debug Module	37
3.5 Experimental Results	38
3.6 Conclusions.....	41
Chapter 4 Automated Selection of Signals to Observe for Efficient Silicon Debug.....	42
4.1 Introduction.....	42
4.2 Overview of Proposed Scheme	45
4.3 Details of Signals to Observe Selection.....	46
4.3.1 Generating Error Transmission Matrix.....	46
4.3.2 Merging Relatively Independent Flip-Flops.....	49
4.3.3 Determining the Set of Signals to Observe.....	51
4.4 Experimental Results	54
4.5 Conclusions.....	56
Chapter 5 Test Point Insertion Using Functional Flip-Flops to Drive Control Points	57
5.1 Introduction.....	57
5.2 Overview of Proposed Scheme	61
5.3 Details of Control Point Replacement Flow	63
5.3.1 Finding Candidate Functional Flip-flops.....	63
5.3.2 Selecting Candidate Flip-Flop	66
5.3.3 Testability Consideration.....	67
5.3.3.1 Path Inversion and Control Point Structures.....	68
5.3.3.2 Illegal Reconvergence.....	71
5.4 <i>TP_Enable</i> Signal Probability	72
5.5 Experimental Results	73
5.6 Conclusions.....	79
Chapter 6 Reducing Test Point Area for BIST through Greater Use of Functional Flip-Flops to Drive Control Points	81
6.1 Introduction.....	81
6.2 Overview of Test Point Insertion Using Functional Flip-Flops.....	83
6.3 Alternative Selection Algorithm.....	87

6.4 Experimental Results	92
6.5 Conclusions.....	94
Chapter 7 Conclusion and Future Work	95
7.1 Conclusion	95
7.2 Future work.....	96
Bibliography	98
Vita	103

List of Tables

Table 2.1:	Erroneous Response Detection Clock Cycles.....	21
Table 2.2:	Number of Debug Sessions.....	21
Table 3.1:	Results for Proposed Method for Different Size Trace Buffers and Error Rates	40
Table 4.1:	Number of Flip-Flops Observed by Proposed Method.....	54
Table 4.2:	Average Erroneous Response Detection Latency Results for 300 Different Random Error Injections	56
Table 5.1:	Area Overhead Reduction Results	73
Table 5.2:	Testability Comparison of Proposed Method with Standard Implementation	76
Table 6.1:	Dedicated Flip-Flop Replacement Comparisons	93
Table 6.2:	Improvement Comparisons	94
Table 6.3:	Testability Comparisons	94

List of Figures

Figure 2.1: Scan Chain and Logic Cone.....	11
Figure 2.2: Graph Representation of Logic Cone Relation.....	12
Figure 2.3: First Cluster Generation (a), (b).....	13
Figure 2.4: Merge & Update Process (a), (b).....	14
Figure 2.5: Merge & Update Example ($n = 2$ and $m = 3$) (a), (b)	14
Figure 2.6: Configured MISRs, DFD and Logic Cones.....	16
Figure 3.1: Session 1 : Parity Generation.....	28
Figure 3.2: Sessions 2 : 2-D Compaction.....	29
Figure 3.3: Example of 2-D Compaction using MISR with $k=5$ and Cycling Register with $m=5$ for 15 clock cycles.....	31
Figure 3.4: Tag Data Generation Algorithm	33
Figure 3.5: Example of 2-D Compaction using MISR with $k=5$ and Cycling Register with $m=5$ for 30 clock cycles.....	34
Figure 3.6: Session 3 : Selective Capture with Tag Bit.....	35
Figure 3.7: Data in Trace Buffer for 15 Tag Bits from Example in Fig 3.3.....	36
Figure 3.8: Data in Trace Buffer for 15 Compressed Tag Bits from Example in Fig. 3.5	36
Figure 3.9: Hardware Architecture of Proposed Debug Module	38
Figure 4.1: Example of a Simple Logic.....	46
Figure 4.2: Error (in A and E) Propagation.....	47
Figure 4.3: Error Transmission Matrix.....	48
Figure 4.4: Updated Error Transmission Matrix	50
Figure 4.5: ILP Formulation for Updated Error Transmission Matrix.....	52

Figure 4.6: General ILP Formulation for Updated Error Transmission Matrix	53
Figure 5.1: Example of Control Points	58
Figure 5.2: Proposed Design Synthesis Flow with Testability and Area Overhead Minimized Test Point Insertion	61
Figure 5.3: Example of a Circuit with Control Point Insertion (<i>Ctrl</i>)	64
Figure 5.4: Conventional and Proposed Control Point	66
(a) Conventional Control Point with Dedicated Flip-Flop	66
(b) Example of Proposed Control Point with Functional Flip-Flop	66
Figure 5.5: New Types of Control Point Structure for Different Path Inversion Parity	68
(a) <i>Type 1</i> : Non-Inverting Functional Path with AND Ctrl	68
(b) <i>Type 2</i> : Non-Inverting Functional Path with OR Ctrl	68
(c) <i>Type 3</i> : Inverting Functional Path with OR Ctrl	68
(d) <i>Type 4</i> : Inverting Functional Path with AND Ctrl	68
Figure 5.6: AND Tree Example with Proposed Control Point Structure	70
Figure 5.7: Example of a Circuit with Illegal Reconvergence	71
Figure 5.8: Testability vs. <i>TP_Enable</i> Signal Probability	78
Figure 6.1: New Types of Control Point Structure for Different Path Inversion Parity	86
(a) <i>Type 1</i> : Non-Inverting Functional Path and AND Ctrl	85
(b) <i>Type 2</i> : Non-Inverting Functional Path and OR Ctrl	85
(c) <i>Type 3</i> : Inverting Functional Path and AND Ctrl	86
(d) <i>Type 4</i> : Inverting Functional Path and OR Ctrl	86
Figure 6.2: Example of a Circuit with One Control Point (<i>Ctrl</i>)	88
Figure 6.3: Alternative Selection Algorithm	89

Figure 6.4: Dedicated Flip-Flop in Fig. 6.2 Replacement
by a Functional Flip-Flop.....92

Chapter 1: Introduction

The advance of technology allows sophisticated designs with millions of transistors. Because of the increasing design complexity, pre-silicon verification is still insufficient to eliminate (electrical and functional) bugs and thus nonconforming chip behavior can still be found after the design is manufactured. Therefore, identifying and resolving problems in integrated circuits (ICs) are the main focus of the pre-silicon and post-silicon debug process.

In pre-silicon verification stage, design verification for checking the correct circuit behavior can be performed mainly via simulation techniques using testbenches and formal verification using different levels of design abstraction. Due to simulation time and limited resources, exhaustive simulation to achieve 100% coverage with larger and complicated designs becomes impractical. Along with the bug escapes in the pre-silicon stage, the inaccuracies in modeling integrated circuits (ICs) with process variation during the manufacturing process are the main reason why manufactured chips show operation misbehaviors or fail to meet specifications. Identifying and resolving problems in ICs after first silicon arrives is called the post-silicon debug. Unlike during pre-silicon verification, the accessibility and visibility of internal signals are very limited in post-silicon debug, post-silicon debug is a very time consuming task. As shown in the International Technology Roadmap for Semiconductors (ITRS) [ITRS 05], post-silicon debug is a major time consuming challenge that takes more than 30% of the development cycle which is significant impact on the development cycle of a new chip. Because the internal system states can be read out when the system operation is halted or can be captured in on-chip memories, there are limitations in acquiring the internal signal information. The narrow observability of internal signals makes silicon debug costly

and time consuming. Therefore, it is very important to enhance the signal observability in the post-silicon debug. This dissertation addresses issues in the post-silicon debug process for enhancing the internal signal observability and proposes several debugging techniques via inserting Design for Debug (DFD) hardware insertion.

This chapter provides background on the issues related to post-silicon debug for improving the debug process by increasing the internal signal observability. Section 1.1 describes a complete, but non-real time observation technique and section 1.2 describes a selective, but real time observation technique. The contributions and organization of this dissertation are provided in section 1.3.

1.1 COMPLETE, BUT NON-REAL TIME OBSERVATION TECHNIQUE

Scan chains are widely used to support manufacturing test by allowing reading the system states. Acquisition of internal signal information is the key issue of the post-silicon debug process. Therefore, scan chains are reused for post-silicon debug. Scan-based debug technique [Hopkins 06], [Vermeulen 02] can achieve high observability of internal signals. However, it requires halting the system to scan out responses from the circuit-under-debug (CUD). Circuit misbehavior can be identified via internal system states which are read out through the scan chains. Because some bugs are visible after thousands of clock cycles later after they are activated, it is desirable to store the histories of system states. However, scan-based debug techniques do not allow real time debug data acquisition. This is because observing the system state requires halting the system to perform the scan dump and hence is not suitable for at-speed debug [Hopkins 06].

Scan dumps play a key role in binary search based debug [Yen 06] for observing the state of the circuit-under-debug. Binary search based debug involves iteratively dividing the search space in half until the first cycle that the error is exercised and observed is found. The drawback of binary search based debug is that it can require a

large number of debug sessions to zero in on the first failing cycle where each session requires halting the system to perform a scan dump. Therefore, scan-based debug can achieve complete observation through the scan dumps, but non-real time observation capability cannot be obtained.

1.2 SELECTIVE, BUT REAL TIME OBSERVATION TECHNIQUE

As stated in the previous section, it may take many cycles for an error in the system to propagate to a primary output where it can be observed. Hence, there is a long time gap from when a bug is invoked to when it is visible. Scan-based debug provides high signal observability, however, due to this latency, finding the root cause of errors using this technique becomes a time consuming task. Hence, recording or monitoring the real time signal information is helpful to understand the continuous operation of a chip for post-silicon debug. On chip memories or pins are widely used for acquiring real time signal observation. The debugging capability is restricted by the available resources such as a memory space, the number of available pins, etc. To efficiently utilize the hardware resources, the number of signals which can be observed is limited and signals are selectively determined for observation. Therefore, the signal observability in this debugging technique is not as high as the one achieved in Sec. 1.1.

Trace buffers are commonly used to capture data from some selected signals to aid the debug process [Abramovici 06], [Anis 07a, 07b], [Hopkins 06], [Yang 08a]. They provide at-speed signal capture capability over a number of clock cycles which enhances the observability of the internal signals. Compression techniques can be applied to extract more information for silicon debug and this further improves the visibility for the debug process. However, in trace buffer based debug, the on-chip storage space limits the number of signals and the number of clock cycles over which the information is available. Due to this limitation, the information provided by the trace

buffers may not be enough to find both temporal (when) and spatial (where) bug information for failures in a silicon.

Signals of interest can also be directly connected to the chip pins and they can be analyzed by the external logic analysis equipments [Hammond 89, 90]. The number of available pins and the operating frequency of external logic analyzer bottleneck the debugging capability.

The debug technique which is introduced in this section removes the necessity of system halt for debug data acquisition. However, the number of signals that can be observed is limited by bandwidth and storage requirements. Therefore, a limited number of signals can be selected and they can be stored for the debug purpose. In this post-silicon debugging technique, the real time observation capability is achieved, however, a limited number of signals can be monitored and hence less observability is obtained than scan-based debug.

1.3 CONTRIBUTIONS AND ORGANIZATION OF THE DISSERTATION

In this dissertation, we propose five techniques to tackle the main issues in the post-silicon debug : non-real time observation problem in scan-based debug in section 1.1 and the limited resource problem in trace buffer or pin access based debug in section 1.2.

In Chapter 2, we propose a new debug technique for scan-based debug based on reusing non-destructive scan chains. Shadow flip-flops or latches duplicate the contents of the scan elements. They are often used to provide a non-destructive scan out capability that preserves the existing system state after the scan dump. Many systems are fully scannable with non-destructive capability which is helpful for both test and debug [Carbine 97], [Vermeulen 02], [Kuppuswamy 04]. Shadow flip-flops or latches are used for manufacturing test and idle in normal system operation. By exploiting this fact, the shadow flip-flops are configured to operate as multiple-input signature registers

(MISRs) during system operation. While the shadow flip-flops normally do not perform any function when the system operates, in the proposed method they are formed into multiple MISRs to enhance silicon debug capability based on structural information. Compressed information from the multiple MISRs is periodically monitored with externally provided data to detect erroneous behavior. A three step debug process is used to zero in on the first failing clock cycle trying to minimize the number of scan dumps. By providing observability of the system state without the need for scan dumps, the proposed method can detect erroneous behavior far earlier than conventional debug methods can. In addition, we propose a debug method to bypass errors which can facilitate downstream debug. Because the presence of a bug may prohibit accurate downstream debug, a faulty response replacement or data masking method may be needed to assist in validating a system.

In Chapter 3, we propose a new debug technique for the efficient use of on-chip memory. A new method for expanding the depth of the observation window for a trace buffer is proposed which requires only 3 debug sessions. It can expand the depth (the number of cycles stored in a trace buffer) of the trace buffer by orders of magnitude which can greatly speed up the debug process. The proposed method exploits the fact that it is not necessary to capture error-free data in the trace buffer since that information is obtainable from simulation. Clock cycles in which errors are possibly present are captured in the trace buffer. During the first debug session, the rough error rate is measured, in the second debug session, a set of suspect clock cycles where errors may be present is determined, and then in the third debug session, the trace buffer captures only during the suspect clock cycles. The suspect clock cycles are determined through a two dimensional (2-D) compaction technique using a combination of multiple-input signature register (MISR) signatures and cycling register signatures. By intersecting the

signatures, the proposed 2-D compaction technique leaves only a small set of remaining suspect clock cycles for which the trace buffer needs to capture data.

In Chapter 4, a method to maximize the effectiveness of limited internal signal observability is proposed based on careful selection which signals to observe. Since there are limited hardware resources available for silicon debug, it is very important to select the right signals to observe for maximizing the debugging capability. An automated procedure is described for selecting the signals to observe to maximize early error detection during silicon debug. By detecting circuit misbehavior soon after it occurs, the search space for zeroing in on the root cause of the misbehavior is greatly reduced thereby speeding up the debug process. The proposed method exploits the nature of error propagation in sequential circuits by observing signals which are most often sensitized to possible error sites. The set of signals to observe is determined by using an error transmission matrix that is generated by analyzing which flip-flops are sensitized to other flip-flops. Signal observability is enhanced by merging data from relatively independent flip-flops. The final set of signals to observe is determined through integer linear programming (ILP) which provides a set of locations that maximally cover the possible errors with a given condition. Experimental results demonstrate the debug effectiveness of this approach compared with conventional methods.

In this dissertation, the investigation on the signal observability is further extended to the signal controllability. Observability is enhanced by adding observation points and controllability on a particular node is enhanced by adding a control point. Since dedicated flip-flops are placed for the observability and controllability, area overhead is an issue in inserting test points. Chapter 5 introduces a novel method for test point insertion for reducing test area overhead. Instead of dedicated flip-flops for

driving the test point, the proposed method uses functional flip-flops to drive control test points for the area overhead reduction. Experimental results indicate that the proposed method significantly reduces test point area overhead and achieves essentially the same fault coverage as the implementations using dedicated flip-flops driving the control points.

Chapter 6 investigates methods to further reduce the area overhead by replacing dedicated flip-flops which could not be replaced in chapter 5. A new algorithm (alternative selection algorithm) is proposed to find candidate flip-flops out of the fan-in cone of a test point. Experimental results indicate that most of the not-replaced flip-flops in chapter 5 can be replaced and hence even more significant area reduction can be achieved with minimizing the loss of testability

Finally, Chapter 7 concludes the contributions of this dissertation and provides the directions for future work.

Chapter 2: Enhancing Silicon Debug via Periodic Monitoring

This chapter presents a new debug technique for scan-based silicon debug. Scan-based debug methods give high observability of internal signals, however, they require halting the system to scan out responses from the circuit-under-debug (CUD). This is time consuming as many scan dumps may be required. In this chapter, conventional scan chains that have non-destructive scan out capability are configured to operate as multiple MISRs during system operation. Information from the multiple MISRs is monitored periodically to identify erroneous behavior. A procedure for constructing the MISRs to maximize debug capability is described. A three step process is used to zero in on the first clock cycle in which an error is present with a small number of scan dumps. Moreover, a method for bypassing errors is described to permit debug in the presence of multiple bugs.

2.1 INTRODUCTION

As stated in Chapter 1, scan chains are used to support manufacturing testing and can be reused for post-silicon debug to increase debug capability [Carbine 97], [Hopkins 06], [Gu 06]. Scan dumps give high observability of internal signals and states after the occurrence of a triggering event. Scan dumps play a key role in binary search based debug [Yen 06] for observing the state of the circuit-under-debug (CUD). Binary search based debug involves iteratively dividing the search space in half until the first cycle that the error is activated and observed is found. In [Vermeulen 02], methods for using scan chains to further increase observability were introduced. Hardware debug modules integrated with scan chains are added to a chip and provide the capability to start, stop, reactivate, or single step execute the debug process with the scan chain values being delivered through the IEEE 1149.1 standard test access port (TAP). The drawback of

binary search based debug is that it can require a large number of debug sessions to find the first failing cycle where each session requires halting the system to perform a scan dump.

Shadow flip-flops or latches are often used to provide a non-destructive scan out capability that preserves the existing system state. Many systems are fully scannable with non-destructive capability which is helpful for both test and debug [Carbine 97], [Vermeulen 02], [Kuppuswamy 04]. Note that while the system is running, shadow flip-flops or latches are not used for system operation. This fact is exploited in the work.

In this chapter, we propose a new debug technique based on reusing non-destructive scan chains. The shadow flip-flops are configured to operate as multiple-input signature registers (MISRs) during system operation. The shadow flip-flops normally do not perform any function when the system operates, however, in the proposed method they are formed into multiple MISRs to enhance silicon debug capability. Compressed information from the multiple MISRs is monitored periodically with externally provided data to identify erroneous behavior. The MISRs are constructed based on structural information of the circuit to maximize their debug efficiency. A three step debug process is used to zero in on the first failing clock cycle trying to minimize the number of scan dumps. By providing high observability of the system state without the need for scan dumps, the proposed method can detect erroneous behavior far earlier than conventional debug methods can. In addition, we propose a debug method to bypass errors which can facilitate downstream debug. Because the presence of a bug may prohibit accurate downstream debug, a faulty response replacement or data masking method may be needed to assist in validating a system.

This chapter is organized as follows: Sec. 2.2 describes how to configure multiple MISRs using non-destructive scan chains. Sec. 2.3 describes the features of the three step debug process, and Sec. 2.4 describes an error bypassing method, Sec. 2.5 shows the experimental results, and Sec. 2.6 concludes the chapter.

2.2 PROCEDURE FOR CONFIGURING MULTIPLE MISRS

Scan based debug gives greater observability than trace buffer based debug [Anis 07a, 07b], however, observing the system state requires halting the system to perform the scan dump and hence is not suitable for at-speed debug [Hopkins 06]. It may take many cycles for an error in the system to propagate to a primary output where it can be observed, so there can be a long time gap between from when a bug is invoked and to when it is visible. Due to this latency, it can be time consuming to find the root cause of errors using only a scan dump based debug methodology.

As described in [Gu 02] and [Vermeulen 02], there have been several techniques proposed to reuse DFT logic for silicon debug. In this chapter, conventional scan chains with non-destructive scan out capability are reused and configured to operate as a set of MISRs. A periodic checking scheme is proposed to monitor the states of a system without scan dumps. The MISRs configured from the shadow flip-flops in scan chains keep compacting the system state and linear compactors further compress the MISR signatures to greatly reduce the volume of debug data. By having only a very small amount of highly compressed data which represents the system state, it is possible to monitor this data and detect any misbehavior in the circuit much earlier than when it would normally become functionally observable at the chip pins. Structural information is used to configure the multiple MISRs in a way that helps to more rapidly diagnose the root cause of the erroneous behavior. By carefully configuring the MISR signatures, it is possible to extract spatial information about where the error is originating from.

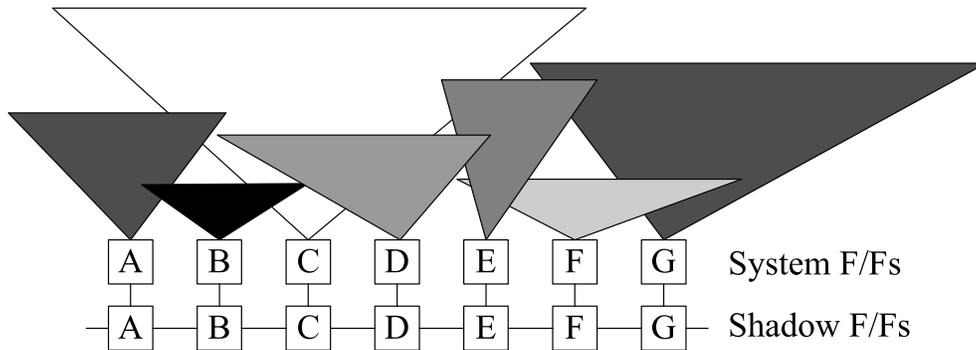


Figure 2.1. Scan Chain and Logic Cone

Fig. 2.1 shows an example of the logic cones driving the system flip-flops along with the shadow flip-flops present for non-destructive scan. The flip-flops are labeled as *A-G* for the illustrative purposes. By traversing the netlist, the degree of overlap between the logic cones that drive each flip-flop can be determined. The partitioning of flip-flops into multiple MISRs is based on this logic cone analysis. If only a single large MISR was constructed, it would require long wires to generate the feedbacks and may cause other issues related to the physical design. Partitioning the flip-flops into multiple MISRs addresses this problem and can also be used to spatially isolate the candidate error sites to speed up the debug process. The proposed approach is based on partitioning scan cells that are the most structurally related in the design together in the same MISR. This helps to minimize routing of the MISRs as well as maximize spatial diagnosis capability by reducing the probability that an error propagates to multiple MISRs.

In the proposed approach, the MISRs are configured using a graph that represents the degree of logic cone overlap between different flip-flops. In Fig. 2.2, each node corresponds to a flip-flop in Fig. 2.1. The weight on the edges corresponds to the amount of logic cone overlap between the logic cones of the corresponding nodes measured in terms of the number of gates that are shared between the two cones. For

example, the logic cone driving flip-flop A overlaps with cones of B and C , and the degree of overlap is 20 and 10, respectively. MISRs are generated using an initial clustering procedure followed by an iterative merge & update procedure. The details of algorithm are described in the following subsections.

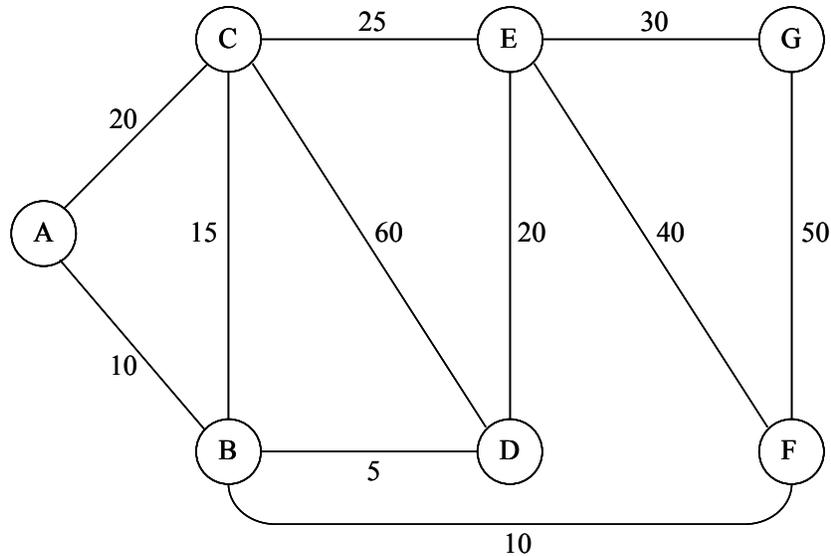


Figure 2.2. Graph Representation of Logic Cone Relation

2.2.1 Initial Clustering Procedure

There are two inputs to the initial clustering procedure, one is the number of MISRs, n , and the other is the minimum size of a MISR, m , which determines the minimum aliasing probability. The initial clustering procedure select n clusters of the m most overlapped logic cones in each. The edge with the largest weight is selected as a starting point for the first cluster. Because the weight represents the logic cone overlap size, the largest weight has a higher probability of error propagation to both cones assuming that the functional and electrical bugs occur with Gaussian distribution. For the same reason, if there are equal size overlaps, the flip-flop driven by the largest logic

cone and its neighbor flip-flop are selected. Therefore, the edge between node C and D is selected from Fig. 2.3(a), and node C and D are merged to begin constructing the first MISR (dashed circle in Fig. 2.3(a)).

The graph is updated after the nodes are merged. Nodes which are merged generate a composite node in a graph. From Fig. 2.3(b), since the node C and D are merged, they form a new node and the edge weights are updated. Additional nodes are added to the cluster in a greedy fashion by selecting the edge with the largest weight attached to the current cluster until the size of the cluster reaches m which ensures a certain minimum aliasing probability for the MISR. This process is repeated to create the n initial clusters.

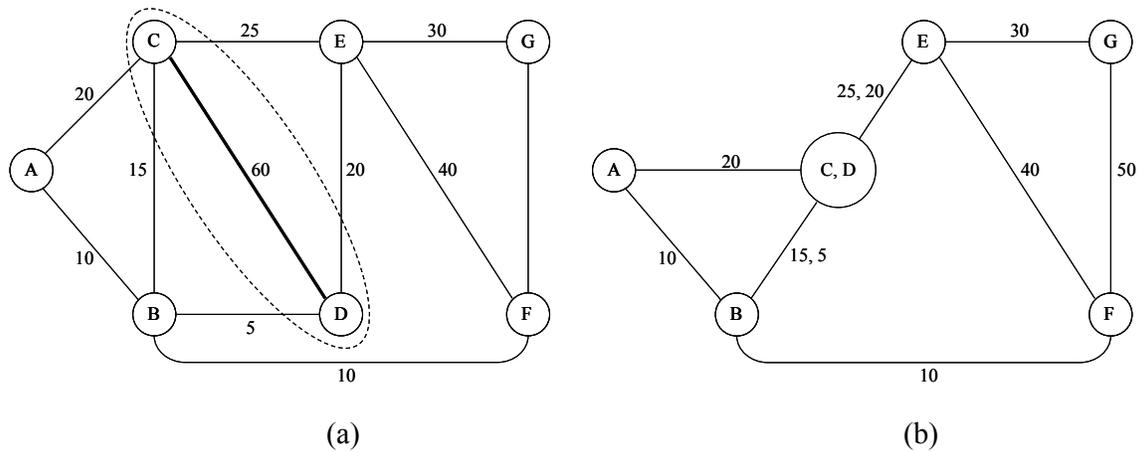


Figure 2.3. First Cluster Generation

2.2.2 Merge & Update Procedure

Once the initial clusters have been constructed, a merge & update procedure is iteratively performed to merge the remaining nodes together using the largest weight on

an edge at each step. At the conclusion of the procedure, all the nodes will have been added to the initial clusters to form the n MISRs.

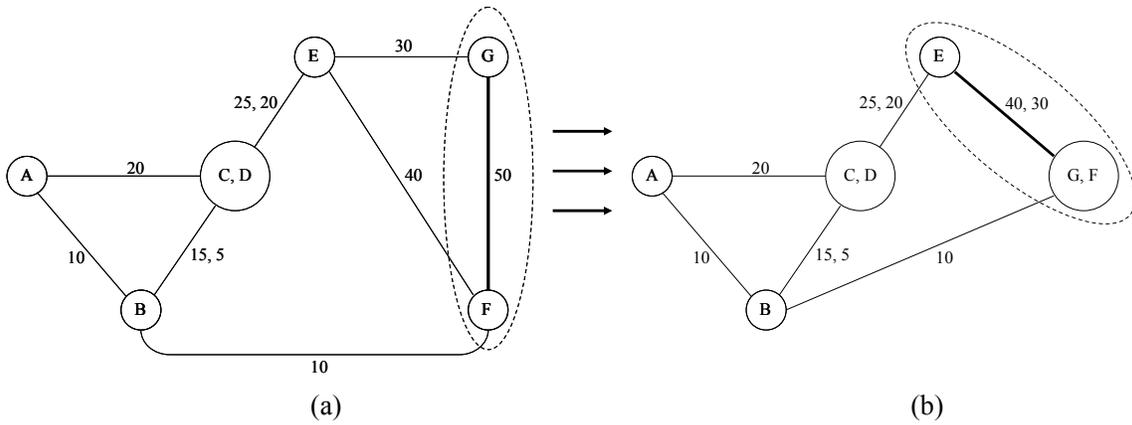


Figure 2.4. Merge & Update Process

In the merge & update process, since G and F have the largest weight edge, the second cluster is generated in Fig. 2.4(a). The iterative process merges node B into the second cluster.

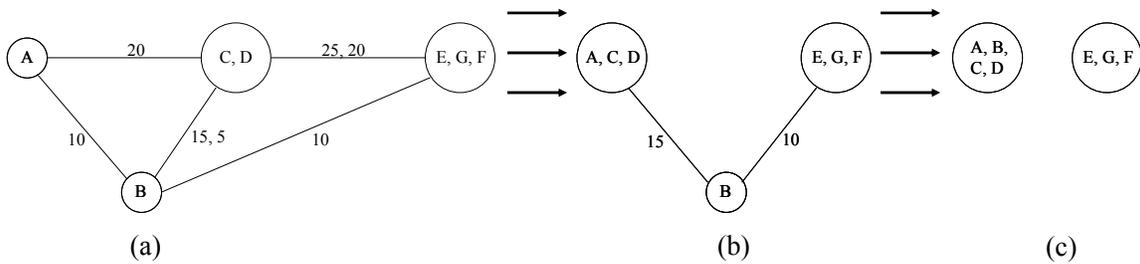


Figure 2.5. Merge & Update Example ($n = 2$ and $m = 3$)

In this small example, assume that the number of MISRs is 2, $n = 2$, and the minimum size of a MISR is 3, $m = 3$. The initial clustering procedure generates two clusters each with three nodes, A, C, D and E, F, G, as shown in Fig. 2.5(b). In Fig.

2.5(c), the final node *B* is merged with node *A*, *C*, *D* since it has more overlap with that cluster, and the two MISRs are finally generated.

2.3 THREE STEP DEBUG PROCESS

Scan-based debug methods give high observability of internal signals by shifting out the internal state values. Conventional scan-based debug needs to stop the system to get the information from the CUD. Checking the internal states by scan dumps is a very time consuming task. If a large number of scan dumps is required, this may be an unattractive way of validating a chip. In Sec. 2.2, a procedure for constructing multiple MISRs is proposed. The MISRs keep compacting the internal state such that erroneous circuit responses will easily corrupt the MISR signatures. Therefore, if the erroneous input comes into the MISRs, although the internal states cannot be read out, the MISR signatures still provide a way to identify the error. In this section, we propose a technique to utilize the internal state information without scanning out the data via scan chains when they have non-destructive capability. A three step debug process is used to zero in on the first erroneous clock cycle. In the first step, single parity information is generated at every clock cycle for periodic monitoring. In the second step, more parity information is stored in a trace buffer to zoom in closer to the failing clock cycle. In the third step, MISR signatures are stored and checked so that the first erroneous cycle can be identified.

2.3.1 Step One : Checking Intermediate Parity

After configuring the MISR as described in Sec. 2.2, the MISRs are used to generate the signatures by compacting the outputs of the logic cones driving it. Linear compaction hardware is used to generate a single parity bit for each MISR signature. This can be done by XORing some subset of the flip-flops in the MISR. XORing the

parities of the MISRs generates a single parity bit which represents the entire system state. Fig. 2.6 shows the logic cones with the MISRs and linear compaction circuits (which are simply XOR networks).

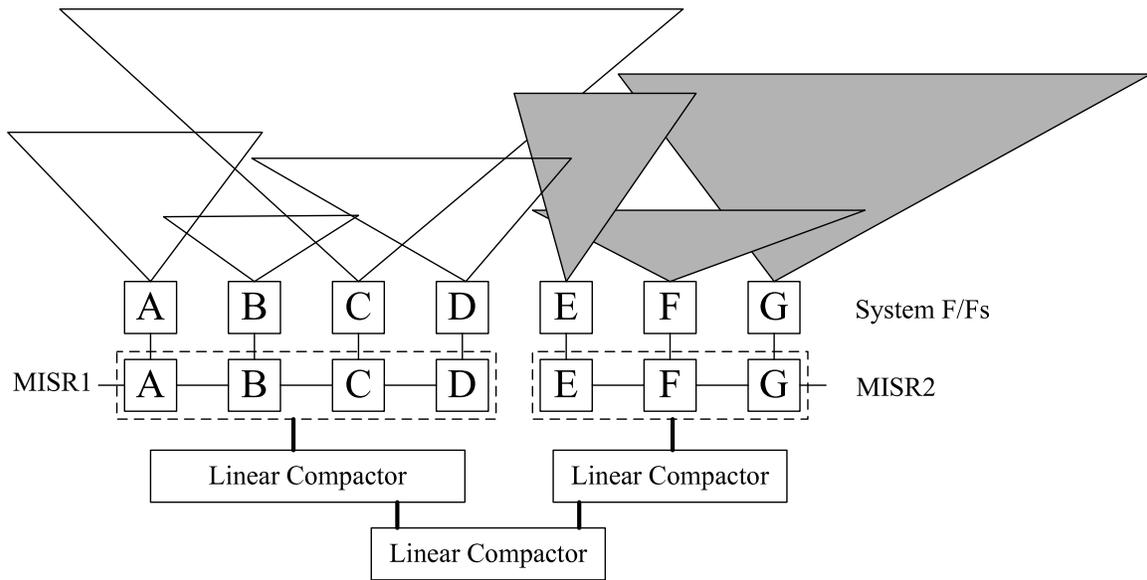


Figure 2.6 Configured MISRs, DFD and Logic Cones

The composite parity generated by combining the MISR parities is monitored periodically to identify erroneous behavior. It is compared with an externally provided golden parity from off-chip (either checked with an ATE or checked with data stored in some external memory). The periodic parity checking allows at-speed debug with far less volume than the conventional scan-based debug. Conventional scan-based debug requires to stopping the system to perform a scan dump and thus periodic monitoring is not feasible. The monitoring period is determined by the relationship between the system and the automatic test equipment (ATE) operating speed (or the speed of the external memory). Since the system typically runs internally at a higher frequency than the ATE does, the ATE can ideally check a golden parity bit at a period equal to the ratio

of the internal chip clock rate to the ATE clock rate, i.e., *frequency of chip / frequency of ATE*. For example, if a chip operates at 3GHz and the ATE runs at 100MHz, the ideal monitoring period is 30 cycles.

Periodic real-time parity monitoring checks the highly compacted circuit response which is represented as a single parity bit and can detect the first erroneous clock cycle within some limited latency which depends on the frequency of the monitoring and the degree of aliasing. The debug session can be stopped when the first mismatching parity is found instead of waiting until the system fails in a functionally observable manner. Since only a single highly compacted bit is being monitored, aliasing is an issue. An even number of error bits will cause parity to alias and hence periodic monitoring would fail to detect a faulty state. However, because the MISRs continue to compact a corrupted signature after the first error, the cumulative aliasing probability exponentially decreases. The probability of 10 consecutive aliases when monitoring the parity is $1/2^{10}$ which is 1 in a thousand.

In addition to monitoring the parity, the proposed approach also involves storing the parity information in a trace buffer at every clock cycle. When the trace buffer gets full, the older data is overwritten so that when the period monitoring halts the debug session, the trace buffer contains the running history of the parity information for the most recent clock cycles. Even though there may be aliasing, the parity history is stored and can be used to find earlier erroneous cycles than the periodic monitoring did. If a 512 Byte trace buffer is used with a 500 cycle monitoring period (one check every 500 clock cycles), then the trace buffer has the history of last 8 monitoring periods. This can be used to more closely zero in on the first erroneous cycle and significantly reduces the debug search space.

2.3.2 Step Two : Storing Parity of MISRs

Periodic parity monitoring reduces the range of cycles over which more careful debug is needed. Moreover, since the trace buffer contents provide cycle by cycle information, the first erroneous clock cycle can be approximated with even better clock resolution. This helps to find the neighborhood of the first erroneous clock cycle, however, it may not be able to zero in on the exact clock cycle since the parities stored in the trace buffer also have aliasing issues for even bit errors.

In step two, the parities of the MISRs are used to provide more specific information for identifying the first erroneous clock cycle as well as spatial information on where the error originates. By looking at the parity of each MISR signature and identifying which ones contain errors, information about which logic cones the errors are getting generated in can be deduced.

The periodic monitoring isolates the error to a small range and allows the debug process to be halted at that point. Periodic monitoring is not required in step two. The parities of the MISRs are stored in the trace buffer from a trigger point based on the information obtain in step one about the rough location of the first error. The triggering can be implemented similar to the internal breakpoint mechanism used in [Cabrine 97], [Anis 07a, 07b]. When the debug session stops, the trace buffer will contain each MISR's parity information for the last set of clock cycles. The number of cycles worth of data will be equal to *size of trace buffer/ number of MISRs*.

2.3.3 Step Three : Storing MISR Signatures

After steps one and two, the search space for the first erroneous clock cycle is significantly reduced and some spatial information is available. However, the first erroneous clock cycle is not precisely known due to aliasing uncertainty. It is necessary to shift out MISR signatures and compare them with golden signatures to find the first

corrupted signature. Scan dumps are only required in step three and only a small number of MISR signature scan outs are required. The volume of debug data can be reduced in comparison to full scan dumps. For example, if the first MISR signature is corrupted while the second MISR signature is clean in Fig. 2.6, the second MISR signature does not need to be investigated at the next scan out. A programmable counter can be used and only the MISR signatures of interest need to be compared to reduce the debug effort. In comparison, binary search based debug requires performing a full scan dump at every iteration.

When the first mismatching signature is found, it provides spatial diagnostic information as well because the MISRs are configured using structural information as described in Sec. 2.2. Multiple MISRs divide the logic cones into multiple regions and aids the debug process. Logic cones that are driving fault-free MISRs can be pruned out to reduce the space of possible root causes.

2.4 ERROR BYPASSING

One difficult aspect of silicon debug is how to bypass errors in order to continue searching for additional bugs after the first bug is found. In a finite state machine (FSM), the state transition depends on the previous state and the circuit inputs. If a faulty response is found, the downstream state transitions are not guaranteed to follow the expected transitions. This makes downstream debug inefficient. A benefit of using scan chains is the accessibility to internal flip-flops. If the first bug in a design is detected and diagnosed, the downstream debug process needs to be able to continue to find additional bugs in a system. Using the proposed approach to precisely identify the first erroneous clock cycle, it is possible to determine what the correct state values should be from either simulation or emulation. By utilizing this information, golden values can be shifted in through the scan chains to return the system to the correct state and facilitate

downstream debug. Because the system can be rerun with bypassing of the erroneous state, periodic monitoring can now be used to catch other bugs.

2.5 EXPERIMENTAL RESULTS

The debug method proposed here tries to detect errors early on and thereby reduce the search space and number of debug sessions. Experiments were performed on the larger ISCAS-89 benchmark circuits [Brglez 89] and OpenRisc processor [OR1200]. Random faults were injected in the benchmark circuits and random input patterns were applied. MISRs were configured using the algorithm in Sec. 2.2 and the parity information was periodically monitored. The results obtained are shown in Table 2.1. The first and second columns show the circuit name and the number of scan elements. The third column indicates the actual error cycle in which the error is first invoked and the fourth column indicates the first clock cycle in which an error becomes functionally observable at a primary output. The next three columns show the first clock cycle that the error is identified using the methods described in Secs. 2.3.1, 2.3.2, and 2.3.3, respectively. The last column shows the number of scan outs required to precisely identify the first erroneous clock cycle. The simulation results show that periodic monitoring detects the erroneous behavior quite close to the first erroneous clock cycle and that using the parities of the MISRs help give more precise information. Hence, very few scan outs are required.

Table 2.2 shows a comparison in terms of the number of debug sessions required using the proposed method and conventional binary search for the same designs and faults as in Table 2.1. The conventional binary search based debug is initiated when the errors are functionally observable at the primary outputs of the circuit, and it uses scan dumps to check the state of a system at the end of each debug session. However, with the proposed method, the three step process requires only 3 debug sessions and only a

few number of scan dumps. MISR signatures can be stored in a trace buffer without additional debug sessions.

Table 2.1 Erroneous Response Detection Clock Cycles

Circuit	Scan Size	Actual Error Cycle	Primary Output Detect Cycle	Step 1		Step 2	Step 3	Num. of Scan Out
				Monitoring Period	Detect Cycle			
s9323	211	43	493	50	100	45	43	3
				100	100			
				200	200			
s15850	534	7	868	50	100	7	7	1
				100	100			
				200	200			
s13207	638	597	828	50	600	597	597	1
				100	600			
				200	600			
s38584	1426	1338	2362	50	1400	1338	1338	1
				100	1400			
				200	1800			
s38417	1636	1221	1680	50	1250	1221	1221	1
				100	1300			
				200	1800			
s35932	1728	5	210	50	100	6	5	2
				100	100			
				200	200			
OR1200	1989	834	1226	50	900	834	834	1
				100	900			
				200	1000			
		3975	5712	50	4050	3975	3975	1
				100	4200			
				200	4200			

Table 2.2 Number of Debug Sessions

Circuit Method	s9323	s15850	s13207	s38584	s38417	s35932	OR1200	OR1200
Binary Search Debug	16	18	20	22	12	17	18	26
Proposed Debug	3	3	3	3	3	3	3	3

2.6 CONCLUSIONS

In this chapter, a new debug technique using a three step process is proposed to zero in the first erroneous clock cycle using a small number of scan dumps. By using conventional scan chains with non-destructive scan out capability, multiple MISRs can be configured to provide high observability with periodic monitoring. Note that the proposed scheme can also be selectively applied to only part of a design, e.g., for newly implemented and unverified design blocks or parts of a design where bugs are more likely to originate from. The scan chains can also be used to restore the system state with golden values to facilitate downstream debug.

Chapter 3: Expanding Trace Buffer Observation Window for In-System Silicon Debug through Selective Capture

Trace buffers are commonly used to capture data during in-system silicon debug. The debugging method proposed in this chapter exploits the fact that it is not necessary to capture error-free data in the trace buffer since that information is obtainable from simulation. The trace buffer need only capture data during clock cycles in which errors are present. A three pass methodology is proposed. During the first pass, the rough error rate is measured, in the second pass, a set of suspect clock cycles where errors may be present is determined, and then in the third pass, the trace buffer captures only during the suspect clock cycles. In this manner, the effective observation window of the trace buffer can be expanded significantly, by up to orders of magnitude. This greatly increases the effectiveness of a given size trace buffer and can rapidly speed up the debug process. The suspect clock cycles are determined through a two dimensional (2-D) compaction technique using a combination of multiple-input signature register (MISR) signatures and cycling register signatures. By intersecting the signatures, the proposed 2-D compaction technique generates a small set of remaining suspect clock cycles for which the trace buffer needs to capture data. Experimental results indicate very significant increases in the effective observation window for a trace buffer can be obtained.

3.1 INTRODUCTION

Post-silicon debug is a major time consuming challenge that has significant impact on the development cycle of a new chip. The most difficult aspect is in-system at-speed debug where there is a need to extract data while the system is running. Trace buffers are commonly used to capture data from a limited number of signals during in-system debug [Hopkins 06], [Abramovici 06]. They are very helpful as they provide

real-time at-speed observation of signals across many clock cycles. Unfortunately, they are a limited resource and can only store a limited amount of data in one session.

Some techniques have been proposed to compress the data stored in the trace buffer to increase its effectiveness. As suggested in [Anis 07a], one can view the width of the observation window provided by a trace buffer as the number of signals observed each clock cycle and the depth of the observation window as the number of clock cycles over which the signals are observed. In [Abramovici 05] and [Hsu 06], techniques are proposed for reconstructing the values of more internal signals than are captured each clock cycle in the trace buffer and hence these techniques expand the effective width of the observation window, but not its depth.

In [Anis 07a], lossless compression methods based on dictionary coding implemented with content-addressable memory were investigated for compressing the data stored in a trace buffer. This approach can expand the depth of the observation window as well. Results in [Anis 07a] for MP3 data show that the observation window can be increased up to 3.45 times larger. However, the amount of compression provided by dictionary coding varies greatly depending on how correlated the data is. While the amount of compression is modest, a nice feature of the method in [Anis 07a] is that it is a one pass scheme which does not require re-running the debug session and hence is useful for debugging non-deterministic behavior that is not repeatable.

If the behavior is deterministic and repeatable, then the method in [Anis 07b] which requires re-running the debug session many times can be used. This approach compacts the observed signals in a MISR and stores MISR signatures in the trace buffer over progressively finer resolutions of time in each debug session. This approach implements an accelerated binary search that progressively zooms in on clock cycles in which errors occur. When the size of the current search range becomes small enough to

fit in the trace buffer, then the trace buffer is used to capture all the data in the remaining portion of the current search. This is a nice and effective idea for accelerating debug methods based on binary search, but it may not be a suitable replacement for more conventional applications of trace buffers because it can require a large number of debug sessions.

In this chapter, a new method for expanding the depth of the observation window for a trace buffer is proposed which requires only 3 debug sessions. It can expand the depth of the trace buffer by orders of magnitude which can greatly speed up the debug process. It is also compatible with other methods for expanding the width of the observation window. The proposed method exploits the fact that it is not necessary to capture error-free data in the trace buffer since that information is obtainable from simulation. The trace buffer need only capture data during clock cycles in which errors are present. During the first debug session, the rough error rate is measured, in the second debug session, a set of suspect clock cycles where errors may be present is determined, and then in the third debug session, the trace buffer captures only during the suspect clock cycles. The suspect clock cycles are determined through a two dimensional (2-D) compaction technique using a combination of multiple-input signature register (MISR) signatures and cycling register signatures. By intersecting the signatures, the proposed 2-D compaction technique leaves only a small set of remaining suspect clock cycles for which the trace buffer needs to capture data.

This chapter is organized as follows. Sec. 3.2 gives an overview of the proposed scheme. Sec. 3.3 discusses the three pass debug procedure in detail. Sec. 3.4 describes the hardware architecture of the proposed scheme. Experimental results are shown in Sec. 3.5 and conclusions are given in Sec. 3.6.

3.2 OVERVIEW OF PROPOSED SCHEME

The proposed scheme involves adding a debug module to a trace buffer which is able to support three operations which are executed in separate debug sessions. The signals being sampled in each clock cycle will be collectively referred to here as the “data word”. In the first debug session, the error rate (i.e., data word errors per clock cycle) is estimated using lossy compression. Based on the estimated error rate, the maximum expanded observation window size is computed as follows:

$$window_size \leq buffer_size / error_rate$$

where $window_size$ is the expanded observation window size, $buffer_size$ is the number of data words that can be stored in the trace buffer, and $error_rate$ is the number of data word errors per clock cycle. Since all the erroneous data words must be stored in the trace buffer, the observation window cannot contain more errors than can fit in the trace buffer.

In the second debug session, the 2-D compaction is activated during the clock cycles in the maximum expanded observation window range to determine the suspect set of clock cycles in which errors may occur. The 2-D compaction consists of using both a MISR and a cycling register and intersecting the information obtained from them to identify the suspects. The MISR is used to generate k signatures where each signature compacts $window_size/k$ consecutive data words. A cycling register of length m compacts the data words such that every m -th data word is XORed together in each signature. The cycling register indicates whether erroneous data exists in each modulo m set of data words. An erroneous data word produces a corresponding erroneous MISR signature and erroneous cycling register signature. Faulty signatures from both compactors (MISR and cycling register) are used to identify the suspect clock cycles by finding the intersections of the signatures.

In the third debug session, the trace buffer captures data during the suspect clock cycles. If there is no aliasing in the compactors, then capturing all suspect clock cycles guarantees that all errors in the expanded observation window will be captured in the trace buffer. As will be shown, the probability of aliasing is extremely small for low error rates (i.e., error rates below 1%). For in-system debug, where the part has already passed a manufacturing test, error rates are typically low as errors occur only at certain corner cases under at-speed operation of the system. The proposed scheme exploits this low error rate property allowing selective capture to achieve significant observation window size expansion which greatly enhances visibility.

3.3 DETAILS OF THE THREE DEBUG SESSIONS

The following subsections describe each of the debug sessions in detail..

3.3.1 Sessions 1 – Estimating Expanded Observation Window Size

In the first debug session, the debug module computes the parity of the data word each clock cycle and stores it in the trace buffer. When the trace buffer gets full, the older data is overwritten, so at the end of the debug session, the trace buffer contains the parity information for the last set of data words. This information is downloaded to a workstation and compared with the fault-free parity values computed through fault-free simulation. By comparing the fault-free parity with the observed parity, the number of erroneous data words can be roughly estimated. Because single-bit parity detects only the odd errors in the data word, only roughly half of errors in the data words are probabilistically detected during the first debug session. A rough estimate of the error rate can be obtained by multiplying the number of parity errors by 2 and dividing by the total number of parity bits stored in the trace buffer. For example, if two parity bits in a 512 byte trace buffer are erroneous, then the error rate is $(2bit * 2) / (512 * 8) = 0.097\%$.

The trace buffer size divided by the error rate is used to estimate the maximum trace buffer observation window size as explained in Sec. 3.2. Note that the achieved observation window size may be considerably smaller than the maximum. The reasons for this will become clear later and will be discussed in Sec. 3.5. The maximum window size as used as the starting point for 2-D compaction in the second session.

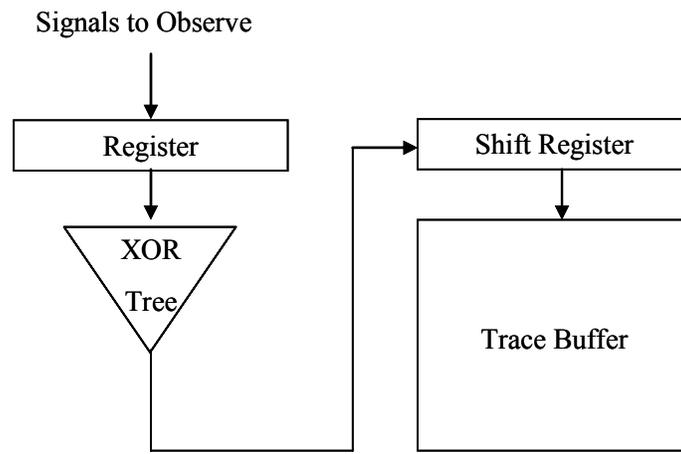


Figure 3.1 Session 1 : Parity Generation

Fig. 3.1 illustrates the operation of the debug module in the first session. Note that the XOR tree can be pipelined as necessary to meet timing requirements.

3.3.2 Session 2 – Determining Suspects

In the second debug session, signatures are generated using the MISR and cycling register beginning from the starting point of the maximum observation window estimated in session 1. The trace buffer is used to store both the MISR signatures and the cycling register signatures. Assume k locations are allocated to store the MISR signatures and m locations are allocated to store cycling register signatures. The MISR signatures are stored every $window_size/k$ clock cycles and the MISR is reset at that time so that the

signatures are independent. The cycling register signatures are generated by XORing together the data word coming in with one of the m locations in the trace buffer pointed to by a mod- m address counter. In this manner, the cycling register will generate m signatures which consist of the XOR of every m -th data word.

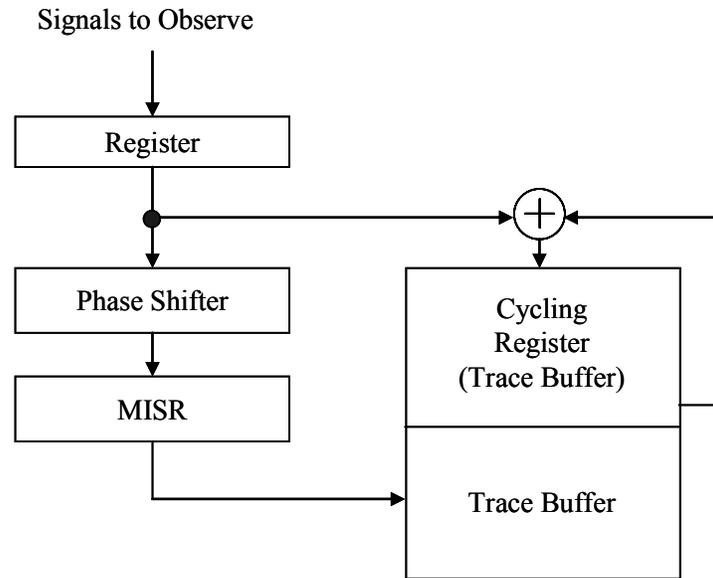


Figure 3.2 Sessions 2 : 2-D Compaction

3.3.2.1 2-D Compaction

Fig. 3.2 illustrates the operation of the 2-D compactor. Each MISR signature compacts a consecutive sequence of $window_size/k$ data words. A symbolic expression of the data words compacted in the signatures is shown in Fig. 3.3 for a small example with a total of 15 clock cycles of data words with $k=5$ and $m=5$. A MISR signature is generated every $(window_size/k)=3$ clock cycles. MS_1 represents the first MISR signature and C_1 denotes the data word in clock cycle 1. MISR signature 1 compacts the data words in cycles 1 though 3 which is expressed as $MS_1 = \{C_1, C_2, C_3\}$. The cycling

register compacts every m -th data word. The first signature in the cycling register in Fig. 3.3 is denoted as CR_1 and is expressed as $\{C_1, C_6, C_{11}\}$ since $m=5$.

If C_{13} is faulty, then MS_5 and CR_3 will mismatch with the fault-free signatures assuming there is no aliasing. The mismatching signatures, MS_5 and CR_3 are highlighted in gray in Fig 3.3. The probability of aliasing in the MISR depends on the size of MISR. For a 32 bit MISR, it is 2^{-32} , and for a 16 bit MISR, it is 2^{-16} . Hence, for a sufficiently large MISR, this aliasing probability is negligible. Aliasing in a cycling register signature occurs when an even number of bit errors occur in the same bit position. The probability of aliasing in a cycling register signature when the error rate is low is approximately equal to the probability of a two-bit error occurring in the same bit position in a cycling register signature (the probability of 4-bit and higher even bit errors are negligibly small compared with 2-bit errors) which is equal to

$$P(\text{Aliasing}) \approx 1 - \left\{ 1 - \binom{NECR}{2} (\text{Bit Error Rate})^2 (1 - \text{Bit Error Rate})^{NECR-2} \right\}^{WORDSIZE}$$

where $NECR$ denotes the number of *data words* compacted in the cycling register signature. For low bit error rates, the aliasing probability is negligible for the cycling register as well.

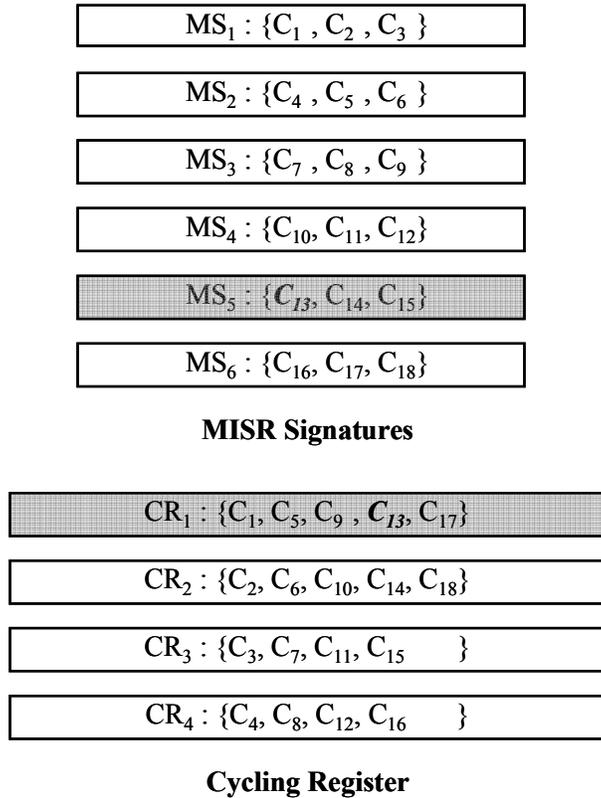


Figure 3.3 Example of 2-D Compaction using MISR with $k=5$ and Cycling Register with $m=5$ for 15 clock cycles

3.3.2.2 Tag Data Generation

As shown in Fig. 3.3, erroneous data in C_{13} corrupts signatures in the MISR and cycling register. By finding the intersection of the mismatching signatures, the suspect clock cycles can be identified. In Fig. 3.3, intersecting MS_5 with CR_3 gives C_{13} .

At the end of second session, all the MISR signatures and cycling register signatures in the trace buffer are downloaded to a workstation where they are compared with the fault-free signatures obtained from simulation. The set of suspects are formed by intersecting all mismatching MISR signatures with all mismatching cycling register signatures and including any clock cycle that is in the intersection.

In the third session, the trace buffer must capture during the suspect clock cycles. The information about when to capture is downloaded into the trace buffer before the start of the third session. The information is represented as a set of “tag bits”, one for each clock cycle in the observation window. Each suspect clock cycle is indicated by setting its corresponding tag bit to 1 and each vindicated clock cycle is denoted by setting its corresponding tag bit to 0. For the example in Fig. 3.3, the tag bit for C_{13} is set to 1, and 0 is assigned to the rest of the clock cycles. In this case, the 15 bit tag information is generated as $0000000000001_{(C_{13})}00$. In the third session, the tag bits are cycled through and used to trigger the trace buffer to capture during the suspect clock cycles.

Fig. 3.4 shows the algorithm for computing the tag bits. Each tag bit has a value of 1 only when the corresponding clock cycle belongs to both a mismatching MISR signature and mismatching cycling register signature.

One complication that arises is that since the tag bits are stored in the trace buffer, the size of a trace buffer could become a limiting factor on the size of expanded observation window. If a tag bit corresponds to one clock cycle, then the maximum number of tag bits that can be stored in the trace buffer sets an upper bound on the observation window size. For example, if a *1K byte* trace buffer is used, it can only store tag information for up to *8192 bits* and hence the observation window would be limited to *8192 cycles*. This may be lower than necessary.

To avoid this limitation, it may be necessary to compress the tag bits. A simple way to do this is to have each tag bit correspond to a consecutive sequence of clock cycles rather than a single clock cycle. The tag bits can be initially computed one per clock cycle, and then successive tag bits can be grouped and compressed into a single bit. One compressed bit is used to represent the whole group. A compressed tag bit has value 0 when there are no 1s in a group, and it has 1 if there is at least one 1. If the

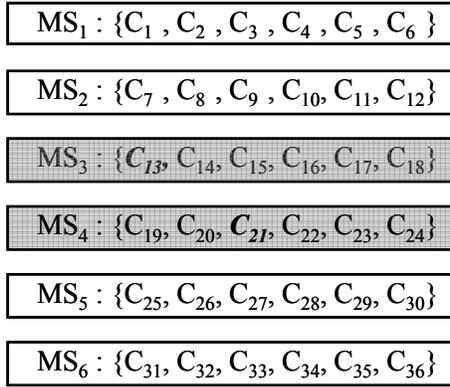
compressed tag bit is 1, the trace buffer must capture during all the corresponding clock cycles.

```
Input: MISR signatures(MS), Cycling Register
       signatures(CR), Golden MISR signature(GMS),
       Golden Cycling Register signatures(GCR)

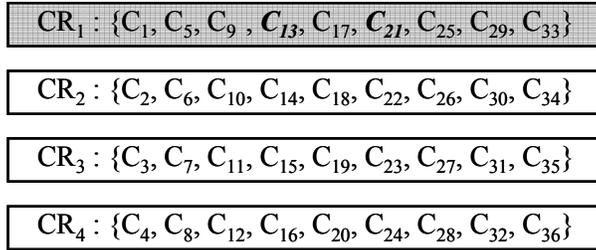
Output: Tag Bits

currentMR = 0; currentGMS = 0;
tagbit[numData] = 0;
while( currentMR < lastMR ){
    List all the element MR[currentMR];
    if( equality(MR[currentMR], GS[currentGMS]) ){
        while( !visited all the element ){
            tagbit[element] = 0;
            next_element;
        }
    }
    else{
        while( !visited all the element ) {
            if( equality(correspondingCR, GCR) )
                tagbit[element] = 0;
            else tagbit[element] = 1;
            next_element;
        }
    }
    currentMR++; currentGMS++;
}
```

Figure 3.4 Tag Data Generation Algorithm



MISR Signatures



Cycling Register

Figure 3.5 Example of 2-D Compaction using MISR with $k=5$ and Cycling Register with $m=5$ for 30 clock cycles

Fig. 3.5 shows a small example of 2-D compaction with a total of 30 clock cycles with $k=5$ and $m=5$. C_{13} and C_{23} are erroneous and corrupt MS_3 , MS_4 , and CR_3 . Intersecting the signatures identifies C_{13} , C_{18} and C_{23} as suspects. The following 30 bit tag data is generated:

$$0000000000001_{(C_{13})}00001_{(C_{18})}00001_{(C_{23})}00000000$$

If tag compression is used to group 2 tag bits into one compressed tag bit, then the 30 bit tag data is compressed into the following 15-bits 000000101001000 which is also illustrated in Fig. 3.8.

3.3.3 Session 3 – Capturing Suspect Clock Cycles

The tag data generated in session 2 is stored in the trace buffer at the start of session 3. During session 3, when in the expanded observation window, the trace buffer captures data whenever the tag bit (or compressed tag bit) for the corresponding clock cycle has a value of 1 indicating it is a suspect. As illustrated in Fig. 3.6, both the tag bits and the captured data are stored in the trace buffer. As the tag data is read out of the trace buffer, it can be overwritten in the trace buffer by the captured data. Enough slack has to be incorporated so that the captured data never overwrites any unread tag data.

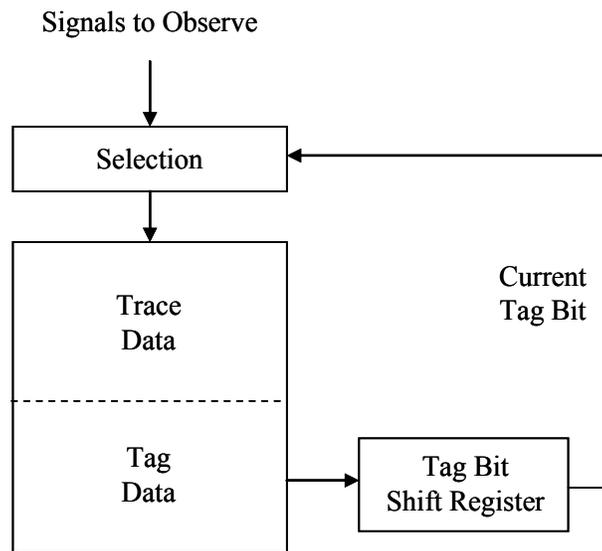


Figure 3.6 Session 3 : Selective Capture with Tag Bit

In the example in Fig. 3.3, C_{13} is identified as a suspect and the tag bits were generated as $0000000000001_{(C_{13})}00$. Fig. 3.7 shows the trace buffer after the third session. For the example in Fig. 3.5, the 30 tag bits are generated and compressed down to 15 tag bits as illustrated in Fig. 3.8. As a result of this compression, in addition to the

suspects (C_{13} , C_{18} , and C_{23}) from the original 30 tag bits, three additional clock cycles are also captured in the trace buffer, namely (C_{14} , C_{17} , and C_{24}).

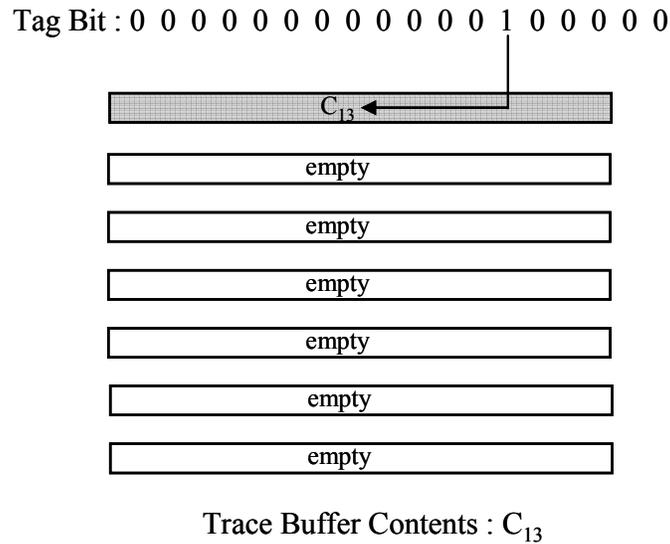


Figure 3.7 Data in Trace Buffer for 15 Tag Bits from Example in Fig 3.3

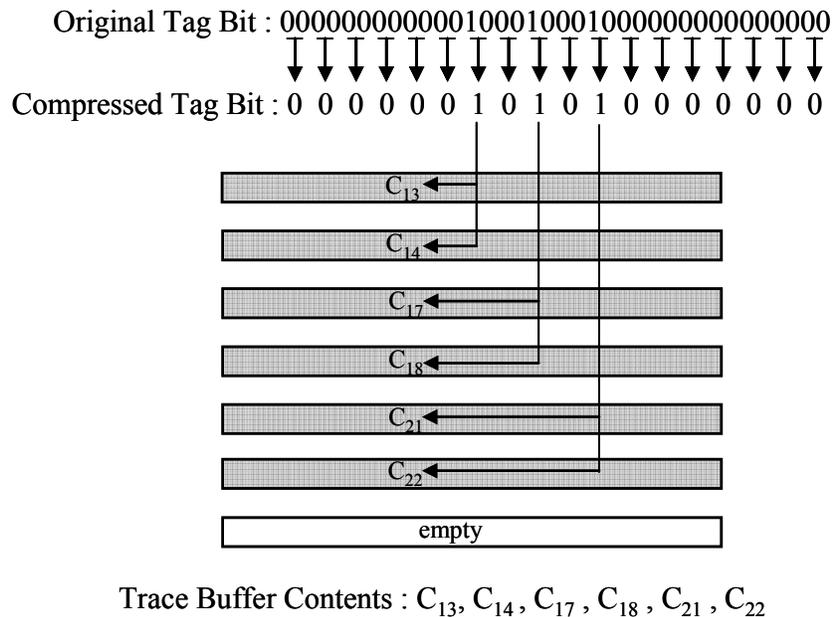


Figure 3.8 Data in Trace Buffer for 15 Compressed Tag Bits from Example in Fig. 3.5

The proposed scheme uses the information from 2-D compaction to significantly increase the size of observation window. Expanding the trace buffer observation window gives visibility over wider range of data. Hence the proposed approach reduces the overall silicon debug time.

3.4 HARDWARE ARCHITECTURE OF DEBUG MODULE

The hardware architecture for a proposed debug module is shown in Fig. 3.9. To perform the operations discussed in Sec. 3.3, the debug module activates different functional blocks using the *Mode_Ctrl* signals.

In session 1, the *Mode_Ctrl* signals select the *phase 1 block* in Fig. 3.9 which is the parity generation mode. In this mode, the debug data is compressed via an XOR tree to generate a single parity bit each clock cycle. The single parity bits are stored in the trace buffer and used for estimating the error rate in the data words.

In session 2, the 2-D compactors in the *phase 2 block*, are activated by the *Mode_Ctrl* signals. The MISR and cycling register signatures are generated and stored in the trace buffer. Since the number of intersections is generally minimized when using an equal number of MISR signatures and cycling register signatures, half of the trace buffer is used to store cycling register signatures and the other half is used to store MISR signatures.

In session 3, the *Mode_Ctrl* signals activate the selection logic in the *phase 3 block* which selectively captures the debug data based on the tag information. A tag bit shift register is used to provide serial access to the tag bits so they can be checked one bit at a time each clock cycle. The suspect clock cycles are selectively captured whenever the tag bit is 1.

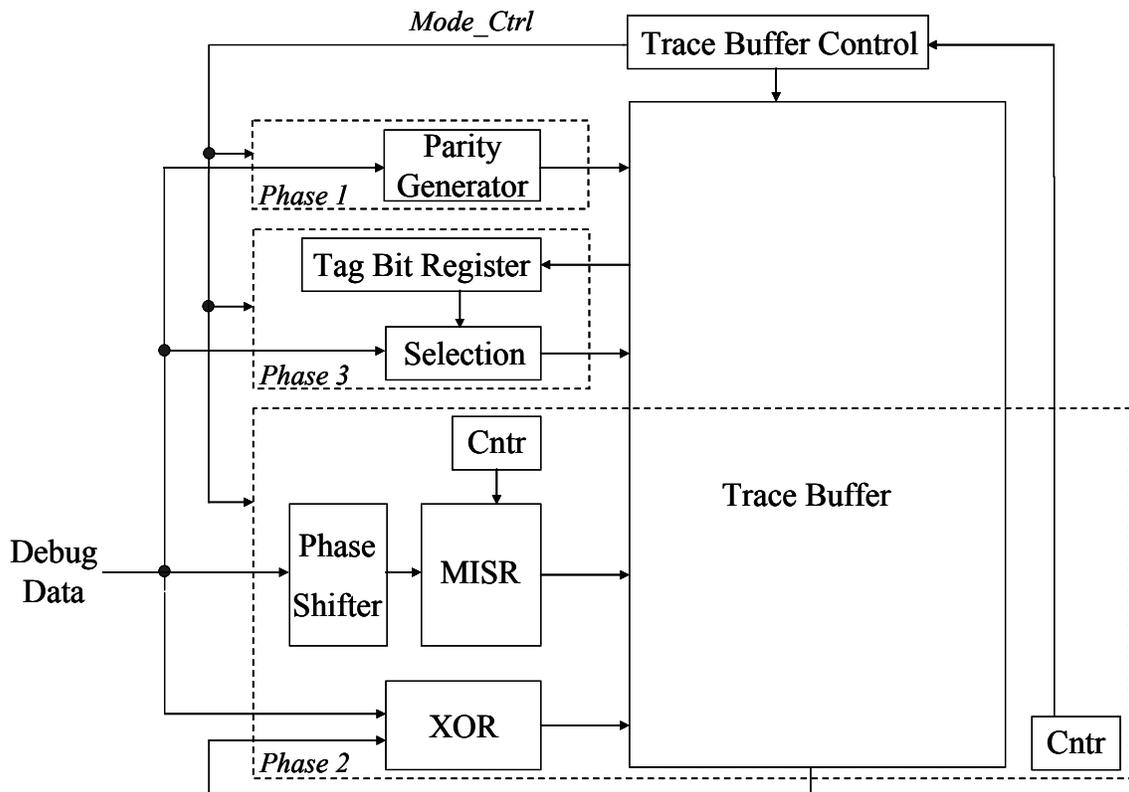


Figure 3.9 Hardware Architecture of Proposed Debug Module

3.5 EXPERIMENTAL RESULTS

In this section, experimental results are presented for an ARM based processor design [Shen 99]. Faults were injected to generate erroneous data with a low error rate. The 32-bit data bus was assumed to be observed by the trace buffer. By changing the injected faults, a set of experiments for different error rates were generated. Table 3.1 shows the results for different error rates using different size trace buffers. The first column shows the size of the trace buffer. The second column shows the error rate computed as the number of 32-bit data bus words with errors divided by the total number of 32-bit data bus words and expressed as a percentage. The third column shows the conventional observation window size in terms of the number of clock cycles worth of

32-bit data words that could be stored in the trace buffer. For example, a 512 byte trace buffer can only capture 128 clock cycles worth of 32-bit words from the data bus. The expanded observation window size that can be obtained using the proposed method is shown in the fourth column. The fifth column shows the expansion ratio which is computed as the expanded observation window size divided by the conventional observation window size. The last column shows the error aliasing percentage.

As can be seen from the results, the lower the error rate, the fewer the number of mismatching signatures from the MISR and cycling register, and hence the 2-D compaction yields fewer suspects resulting in greater observation window expansion. The experimental results had only one case where aliasing occurred and this resulted in a loss of 2.43% of the erroneous data words. Note the aliasing probability can always be reduced by using a less aggressive expanded observation window size.

As discussed in Sec. 3.2, the maximum possible expanded observation window size is equal to the trace buffer size divided by the error rate since the trace buffer must store all the erroneous data words. The expanded observation window size actually achieved with the proposed method is considerably less than that. There are two reasons for this. One is that the 2-D compaction generally yields more suspects than the actual erroneous clock cycles, and the other is that the tag bits may need to be compressed which reduces the suspect resolution thereby increasing the number of clock cycles that need to be captured. Because the maximum expanded observation window size is not achievable, one way to reduce the search space for the 2-D compaction would be to compute a tighter upper bound on the expanded observation window size. This can be done by estimating the number of 2-D signature intersections and the amount of tag bit compression based on the trace buffer size and the estimated error rate. Using this

information, a tighter upper bound on the expanded window size can be computed as follows:

$$window_size \leq \frac{buffer_size}{error_rate * ANI * tag\ bit\ group\ size}$$

where *ANI* denotes the average number of intersections and *tag bit group size* represents the number of original tag bits that need to be compressed together (as discussed in Sec. 3.2.2). This tighter upper bound on the expanded window size can be used to determine when to begin the 2-D compaction.

Table 3.1 Results for Proposed Method for Different Size Trace Buffers and Error Rates

Size of Trace Buffer	Error Rate Percentage	Conventional Observation Window	Expanded Observation Window	Expansion Ratio	Error Aliasing Percentage
512 Byte	0.016	128	19456	152	0
	0.051	128	12032	94	0
	0.097	128	8576	67	0
	0.513	128	3072	24	0
	1.387	128	1792	14	0
1K Byte	0.016	256	28672	112	0
	0.051	256	19968	78	0
	0.097	256	17152	67	0
	0.513	256	5376	21	0
	1.387	256	3328	12	0
2K Byte	0.016	512	61440	120	0
	0.051	512	33792	66	0
	0.097	512	26112	51	0
	0.513	512	9216	18	0
	1.387	512	5632	11	0
4K Byte	0.016	1024	132096	129	0
	0.051	1024	61440	60	0
	0.097	1024	39936	39	0
	0.513	1024	17408	17	2.43
	1.387	1024	10240	10	0

3.6 CONCLUSIONS

The experimental results indicate that the methodology proposed in this chapter can use 3 debug sessions to expand the observation window for a trace buffer by one to two orders of magnitude. This provides much greater visibility of the real-time at-speed operation during in-system silicon debug. The proposed methodology is compatible with other trace buffer compression techniques. Moreover, it can also be applied even when a trace buffer is only triggered during certain events which may not necessarily be in consecutive clock cycles. From the debug modules viewpoint, the stream of data that is being observed can be relative to only the clock cycles when the trace buffer would normally be triggered. The expanded observation window in this case would be expanded only across the clock cycles when the trace buffer would normally be triggered.

It should also be noted that if a design contains multiple trace buffers, the proposed methodology could be concurrently applied to all the trace buffers. So the total number of debug sessions would still be 3 regardless of how many trace buffer observations windows are being expanded.

CHAPTER 4: Automated Selection of Signals to Observe for Efficient Silicon Debug

Internal signals of a circuit are observed to analyze, understand, and debug nonconforming chip behavior. The number of signals that can be observed is limited by bandwidth and storage requirements. This chapter presents an automated procedure to select which signals to observe to facilitate early detection of circuit malfunction to help find the root cause of a bug. The proposed method exploits the nature of error propagation in sequential circuits by observing signals which are most often sensitized to possible errors. Given a functional input vector set, an error transmission matrix is generated by analyzing which flip-flops are sensitized to other flip-flops. Signal observability is enhanced by merging data from relatively independent flip-flops. The final set of signals to observe is determined through integer linear programming (ILP) which provides a set of locations that maximally cover the possible error sites within given constraints. Experimental results indicate that the cycle in which a bug first appears can be more rapidly and precisely found with the proposed approach thereby speeding up the post-silicon debug process.

4.1 INTRODUCTION

The advance of technology allows sophisticated designs with millions of transistors. Due to inaccuracies in modeling integrated circuits (ICs) along with process variations during the manufacturing process, identifying and resolving problems in ICs after first silicon is a very time consuming task [Josephson 04], [Ko 08], [Yang 08a, 08b]. Unlike during pre-silicon verification, the accessibility and visibility of internal signals are very limited in post-silicon debug and hence this is the major challenge in the validation and debug of first silicon. The narrow observability of internal signals makes silicon debug costly and time consuming.

Techniques have been proposed to enhance the observability of internal signals via complete, but non-real time observation, using scan chains and selective, but real time observation, such as using trace buffers or direct access via dedicated pins. Scan-based debug [Hopkins 06], [Vermeulen 02] gives high observability of internal signals by re-using scan chains, however, it requires halting the system to scan out responses from the circuit-under-debug (CUD). Trace buffer based debug [Abramovici 06], [Anis 07a, 07b], [Yang 08a] provides at-speed signal capture capability over a limited number of clock cycles which enhances the observability of the internal signals. The amount of data that can be observed with a trace buffer is limited by its on-chip storage space. Compression techniques can be applied to further improve the observability provided by a trace buffer [Anis 07a, 07b], [Yang 08a]. In [Vermeulen 01], a set of signals required for debugging was connected to a multiplexer module, called *SPY*, for real-time observation, and then captured in a register or monitored via chip pins.

Increased internal signal observability helps to discover erroneous behavior closer to the source of the problem, both in space and time. Some previous research has been done on ways to enhance internal observability. In [Abramovici 05] and [Hsu 06], techniques are proposed for constructing the values of more signals than are captured each clock cycle in a trace buffer. The captured silicon data is mapped to Boolean equations and non-visible values in combinational logic are expanded by a dependency and approximation method. This method provides some improvement in observing localized signals. [Park 08] shows an architectural level approach for post-silicon bug localization. It records the history of the program executed and identifies the bug location-time information at the system level. Experimental results show that its method can effectively locate bugs with high accuracy.

In [Ko 08], an automated data reconstruction method for sequential circuits is investigated. The restorability of signals is calculated to determine the signals to be traced. Results in [Ko 08] for ISCAS benchmark circuits show that this approach can restore signals up to 130 times better. However, if the logic depth between internal state elements is deep, the amount of restorability may be very limited. If the combinational logic depth is shallow, this approach can greatly help post-silicon debug with a number of internal signals implied by captured data.

In [Yang 08b], a signal monitoring technique based on non-destructive scan chains is investigated. In non-destructive scan, shadow scan latches are used to retain the internal state during scan out. Conventional scan chains that have non-destructive scan capability are configured to operate as multiple MISRs during normal system operation. Internal signal observability is increased by observing the compressed internal system states without halting the system. Information from the MISRs is periodically monitored to identify erroneous behavior. Results show that only a small number of scan dumps are needed to zero in the first erroneous clock cycle. However, this technique can only be applied to designs which have non-destructive scan chains.

In this chapter, a method to maximize the effectiveness of limited internal signal observability is proposed based on carefully selecting which signals to observe. An automated procedure is described for selecting the signals to observe to maximize early error detection during silicon debug. By detecting circuit misbehavior soon after it occurs, the search space for zeroing in on the root cause of the misbehavior is greatly reduced thereby speeding up the debug process. The proposed method exploits the nature of error propagation in sequential circuits by observing signals which are most often sensitized to possible error sites. The set of signals to observe is determined by using an error transmission matrix that is generated by analyzing which flip-flops are

sensitized to other flip-flops. Signal observability is enhanced by merging data from relatively independent flip-flops. The final set of signals to observe is determined through integer linear programming (ILP) which provides a set of locations that maximally cover the possible errors with a given condition.

The chapter is organized as follows. Sec. 4.2 gives an overview of the proposed scheme. Sec. 4.3 discusses the three procedures to determine the signals to observe in detail. Experimental results are shown in Sec. 4.4 and conclusions are given in Sec. 4.5.

4.2 OVERVIEW OF PROPOSED SCHEME

The proposed method provides information on which limited number of signals to observe in a circuit to maximize the efficiency of the post-silicon debug process. In a sequential circuit, it may take many cycles for an error to propagate to a primary output where it can be observed [Yang 08b]. In the proposed method, signals are observed along the paths where error propagation is most likely.

In debug mode, functional vectors are applied and the responses are analyzed to validate a chip. Using the functional vectors and treating the flip-flops as sources of errors, fault simulation can be performed to study the error transmission between flip-flops. The error transmission information can be represented as a matrix which will be referred to here as the “error transmission matrix”. Based on this information, the flip-flops that are most often sensitized to other flip-flops can be identified and used as candidates for the set of signals to observe.

Flip-flops are relatively independent if a single error in a circuit will not influence them simultaneously. Relatively independent flip-flops can be XORed together to increase the overall observability of the internal signals. The error transmission matrix can be updated by forming signal groups by combining (XORing) the relatively independent flip-flops in the matrix.

Because there is limited storage space provided by DFD (design for debug) hardware, e.g., a trace buffer, it is important to efficiently choose the set of signals to observe which will detect as many errors as possible. For this purpose, integer linear programming (ILP) is used to select the signal groups from the error transmission matrix.

4.3 DETAILS OF SIGNALS TO OBSERVE SELECTION

The following subsections describe each of the steps in the proposed procedure for selecting the signals to observe.

4.3.1 Generating Error Transmission Matrix

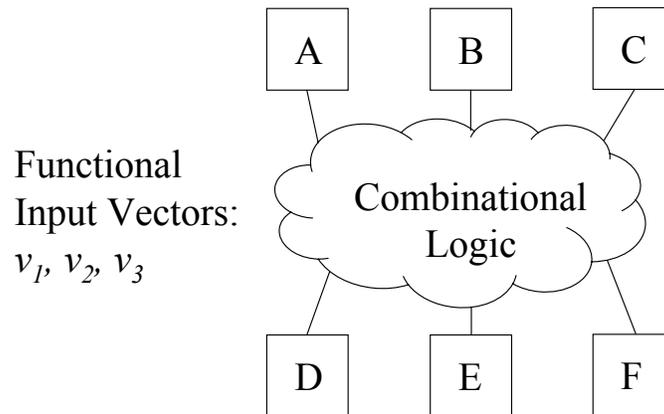


Figure 4.1 Example of a Simple Logic

Fig. 4.1 shows an example of some simple logic that has 6 sequential elements represented by rectangles named A to F and combinational logic illustrated as a cloud. For simplicity, assume three functional vectors (v_1 , v_2 and v_3) are applied to this logic. When the vectors are applied, if there is a bug, the erroneous response could be captured in some flip-flops at some time. That faulty response would likely keep propagating in a sequential circuit over multiple cycles.

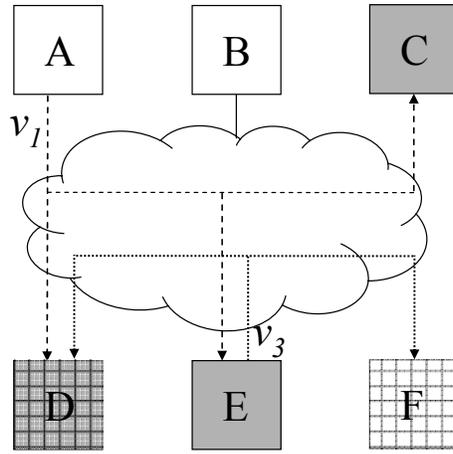


Figure 4.2 Error (in A and E) Propagation

The error transmission matrix is generated by injecting errors at each flip-flop for each vector in the vector set and performing fault simulation for one cycle to see where the error propagates. For example, simulation can be done to see which flip-flops are corrupted by an error in A when input vector v_1 is applied. Next, we make B faulty and see which flops are sensitized to the error with v_1 . To illustrate this, Fig. 4.2 shows where an error at A and E propagates. The error at A is transmitted to flip-flops C , D and E for input vector v_1 (highlighted in gray color), and an error at E is transmitted to D and F for v_3 (highlighted in dashed line), respectively.

	A	B	C	D	E	F
(A, v_1)	0	0	1	1	1	0
(B, v_1)	1	0	0	0	1	0
(C, v_1)	1	1	0	0	0	0
(D, v_1)	0	0	0	0	0	0
(E, v_1)	0	0	0	0	1	0
(F, v_1)	0	1	1	0	1	0
(A, v_2)	0	0	0	0	0	1
(B, v_2)	1	0	0	1	1	0
(C, v_2)	0	1	0	0	0	0
(D, v_2)	1	0	0	0	0	0
(E, v_2)	0	1	1	0	0	0
(F, v_2)	0	0	0	0	1	0
(A, v_3)	0	1	0	0	1	0
(B, v_3)	0	0	0	0	0	1
(C, v_3)	1	1	0	0	1	0
(D, v_3)	0	0	0	0	0	0
(E, v_3)	0	0	0	1	0	1
(F, v_3)	0	0	0	0	1	0

Figure 4.3 Error Transmission Matrix

Error transmission information corresponding to input vectors and error locations is represented in the error transmission matrix. This is illustrated in Fig. 4.3. Each column represents a flip-flop in the circuit, and each row shows the error information. (A, v_1) in the first row indicates that an error is located in A and for vector v_1 it propagates to C , D and E which each have an '1' in the first row of Fig. 4.3. Once the error transmission matrix is generated, the flip-flops that are most often sensitized to possible errors can be identified assuming bugs in silicon are modeled as occurring evenly distributed in time and space. Note that an error will likely propagate for multiple clock cycles and need not necessarily be detected in the first cycle in which it occurs. The

columns in the error transmission matrix with the most 1's are probabilistically more likely to capture errors over time since errors are transmitted to them most frequently. Hence, they are candidates for signals to observe for the better observability. Moreover, if a limited set of signals to observe is to be selected, then columns that are most non-overlapping and cover as many rows as possible are also more likely to cover more errors. This will be discussed in more detail later.

4.3.2 Merging Relatively Independent Flip-Flops

Relatively independent flip-flops in a circuit are identified and merged to achieve better observation capability. The overall goal is to find misbehavior as early as possible, so observing more signals helps silicon debug by providing more internal signal information.

If two flip-flops are relatively independent, the erroneous response for one error will not be simultaneously transmitted to both flip-flops. For the example in Fig. 4.2, since an error in A is transmitted to C , D , and E for vector v_1 , flip-flops A , B , and F are relatively independent to the possible error (A, v_1) . Therefore, A , B , and F can be merged together in this case and E can be combined with A , B , C in the (E, v_3) case.

In the error transmission matrix, if there are multiple 1's in a row, the corresponding flip-flops are relatively dependent for a possible error. By finding the columns in the matrix which are not sensitized to the same errors simultaneously, relatively independent flip-flops can be identified.

Relatively independent flip-flops can be XORed together without losing error observation for single flip-flop errors. Note, however, that flip-flops are relatively independent only with respect to single errors, so it is still possible for multiple errors to cancel. However, this serves as a good heuristic for increasing overall error coverage. In Fig. 4.3, relative independence is checked among flip-flops $(A \sim F)$ and three relatively

independent signal groups can be found. Flip-flop A , C and F are not sensitized to the same errors, and any error set does not influence flip-flop B and D simultaneously. Therefore, the first signal group (S_0), the second group (S_1) and the last group (S_2) can be expressed respectively as follows:

Signal Groups (S_0, S_1, S_2)

$$S_0 = A \oplus C \oplus F$$

$$S_1 = B \oplus D$$

$$S_2 = E$$

	A	B	C	D	E	F		S_0	S_1	S_2
$(A, v_1) : R_0$	0	0	1	1	1	0		1	1	1
$(B, v_1) : R_1$	1	0	0	0	1	0		1	0	1
$(C, v_1) : R_2$	1	1	0	0	0	0		1	1	0
$(D, v_1) : R_3$	0	0	0	0	0	0		0	0	0
$(E, v_1) : R_4$	0	0	0	0	1	0		0	0	1
$(F, v_1) : R_5$	0	1	1	0	1	0	→	1	1	1
$(A, v_2) : R_6$	0	0	0	0	0	1		1	0	0
$(B, v_2) : R_7$	1	0	0	1	1	0		1	1	1
$(C, v_2) : R_8$	0	1	0	0	0	0		0	1	0
$(D, v_2) : R_9$	1	0	0	0	0	0		1	0	0
$(E, v_2) : R_{10}$	0	1	1	0	0	0		1	1	0
$(F, v_2) : R_{11}$	0	0	0	0	1	0		0	0	1
$(A, v_3) : R_{12}$	0	1	0	0	1	0	→	0	1	1
$(B, v_3) : R_{13}$	0	0	0	0	0	1		1	0	0
$(C, v_3) : R_{14}$	1	1	0	0	1	0		1	1	1
$(D, v_3) : R_{15}$	0	0	0	0	0	0		0	0	0
$(E, v_3) : R_{16}$	0	0	0	1	0	1		1	1	0
$(F, v_3) : R_{17}$	0	0	0	0	1	0		0	0	1

Figure 4.4 Updated Error Transmission Matrix

Then, the error transmission matrix is updated based on the signal groups as shown in Fig. 4.4. Error index ($R_0 \sim R_{17}$) is used to identify the errors from (A, v_1) to (F, v_3).

A limit can be placed on the number of signals which are XORed together when the error transmission matrix is updated to minimize delay and/or routing. Results are shown in Sec. 4.4 with different limits on the maximum number of signals XORed together.

4.3.3 Determining the Set of Signals to Observe

Because there are limitations on storage, bandwidth, and overhead for observing signals, it is very important to choose the best set of signal groups to observe for the most efficient debugging. With given conditions in the error transmission matrix, debugging capability can be maximized by finding a set of signal groups that are sensitive to the broadest set of errors.

Integer linear programming (ILP) is employed to select an optimal set of signals based on the error transmission matrix. The updated error transmission matrix in Fig. 4.4 is formulated as the set of equations in Fig. 4.5.

In Fig. 4.5, R_0 denotes 0^{th} row in the updated error transmission matrix and S_0 represents 0^{th} column. Because the objective in solving ILP is to maximize the number of errors covered by a set of signal groups, the objective equation (“*maximize the covered errors*”) is expressed as the summation of the entire error sets (I). If an error is covered by any signal group, then the value ‘1’ is assigned to a corresponding error variable (R_k). And if an error is not covered, ‘0’ will be assigned (4). When a signal group is selected for observation, S_i has ‘1’ (5). For example, if S_0 is selected and S_1 is not selected, S_0 is 1 and S_1 is 0. Because the number of signal groups to observe is always larger than I (i.e. $S_0 + S_1 + S_2 \geq I$) (2), the row constraints can be represented as equations (3). In

the first row, R_0 is always detected by S_{0-0} , S_{0-1} or S_{0-2} where S_{k-i} denotes a signal group in k^{th} row and i^{th} column. Therefore, the ILP formulation from the first row can be expressed as $S_{0-0} + S_{0-1} + S_{0-2} \geq R_0$. From the second row, we can derive a constraint as $S_{1-0} + S_{1-2} \geq R_1$. This implies that if a signal selection of either S_0 or S_2 ($S_{1-0} + S_{1-2} = 1$), or selection of both S_0 and S_2 ($S_{1-0} + S_{1-2} = 2$) will cover R_1 ($R_1 = 1$), and if none of S_0 and S_1 ($S_{1-0} + S_{1-2} = 0$) is selected, R_1 would not be covered ($R_1 = 0$). Therefore, a summation of S_{1-0} and S_{1-2} is always greater than or equal to R_1 . In the same manner, a total of 18 ILP formulations are generated in (3).

$$\max : R_0 + R_1 + R_2 + \dots + R_{15} + R_{16} + R_{17} \quad (1)$$

$$s.t : S_0 + S_1 + S_2 = \text{num. of signal groups to observe} \quad (2)$$

$$\left. \begin{array}{l} S_{0_0} + S_{0_1} + S_{0_2} \geq R_0 \\ S_{1_0} + S_{1_2} \geq R_1 \\ \vdots \\ S_{16_0} + S_{16_1} \geq R_{16} \\ S_{17_2} \geq R_{17} \end{array} \right\} \quad (3)$$

$$R_0, R_1, R_2, \dots, R_{15}, R_{16}, R_{17} \in \{0, 1\} \quad (4)$$

$$S_0, S_1, S_2 \in \{0, 1\} \quad (5)$$

Figure 4.5 ILP Formulation for Updated Error Transmission Matrix

If two signal groups are to be chosen in Fig 4.4, the ILP solver selects S_0 and S_2 . 12 errors are covered by the S_0 and S_1 combination, S_0 and S_2 gives 15 covered errors, and S_1 and S_2 can cover 13 errors. Therefore, S_0 and S_2 cover the maximum number of possible errors with the given constraint in Fig. 4.5. The signals corresponding to S_0 and S_2 are selected for observation.

A general ILP formulation for locating the set of signals to observe is shown in Fig. 4.6. When there are n rows and m columns in the updated error transmission matrix, the constraints are expressed using R and S where R and S denote the error list and the signal group respectively. The formulation of the objective is shown in (1) in Fig. 4.6 to maximize the number of covered errors that satisfies (2). S_{k-i} in (3) is a signal list in i^{th} column with k^{th} row error list in the updated error transmission matrix. And x_{k-i} is an element in the intersection of k^{th} row and i^{th} column of the matrix. The solution space for R and S is $\{0, 1\}$ in (4) and (5).

$$\max: \sum_{k=0}^{n-1} R_k \quad (1)$$

$$s.t: \sum_{i=0}^{m-1} S_i = \text{num. of signal groups to observe} \quad (2)$$

$$\sum_{i=0}^{m-1} x_{k-i} S_{k-i} \geq R_k \quad \text{for } k \in \{0, 1, \dots, n-1\} \\ \text{for } x_{k-i} \in \{0, 1\} \quad (3)$$

$$R_k \in \{0, 1\} \quad \text{for } k \in \{0, 1, \dots, n-1\} \quad (4)$$

$$S_i \in \{0, 1\} \quad \text{for } i \in \{0, 1, \dots, n-1\} \quad (5)$$

Figure 4.6 General ILP Formulation for Updated Error Transmission Matrix

The final set of signals to observe is determined through ILP which provides a set of signal groups that maximally cover the possible errors with the constraints in Fig. 4.6.

The size of the error transmission matrix increases with the number of functional vectors. The matrix can be partitioned for scalability. For example, if there are n patterns, we can generate two error transmission matrices using $n/2$ patterns each. The final signal groups can be determined from the two matrices by counting the number of possible errors detected.

4.4 EXPERIMENTAL RESULTS

Experimental results are presented for ISCAS-89 benchmark circuits [Brglez 89] and an NOC (network-on-chip) design [Jang 08]. Random faults were injected in circuits to generate erroneous data. Random input patterns were applied to the ISCAS-89 benchmark circuits and deterministic functional verification vectors were applied to the NOC design. Fault simulation was conducted to generate the error transmission matrix. As discussed in Sec. 4.3.2, the number of signals which can be merged for the signal groups can be limited to avoid issues related to the physical design such as timing and wiring. We use three threshold values: 8, 12 and ∞ , to limit the number of signals included in a single group. The error transmission matrix was updated using threshold values. To find the final set of signals to observe, GNU Linear Programming Kit (GLPK) 4.32 [GLPK] was used as the ILP solver.

Table 4.1 Number of Flip-Flops Observed by Proposed Method

Circuit	Max Val	Num. of Signal Groups			
		8	12	16	32
s9234	8	23	25	35	53
	12	23	25	35	53
	∞	23	25	35	53
s38584	8	61	87	112	230
	12	86	127	173	380
	∞	893	1037	1205	1402
NOC	8	64	93	126	256
	12	92	134	184	379
	∞	367	921	1169	1543

Table 4.1 shows how many flip flops are observed with each of the three threshold values. The first column shows the circuit name. The number of flip-flops in *s9234*, *s38584*, and *NOC* are 211, 1426 and 1991, respectively. The second column shows the maximum number of signals that can be merged into one signal group when

updating the error transmission matrix. In the third column, the number of flip-flops observed is shown when 8, 12, 16 and 32 signal groups are chosen by ILP in the updated error transmission matrix. As can be seen from the results except for *s9234*, more flip-flops can be observed as the maximum value in the second column is increased. Since *s9234* is a relatively small benchmark circuit, the number of relatively independent flip-flops that are found is limited by its circuit size and not by the three threshold values.

Table 4.2 shows comparisons in terms of the average latency to detect bugs using the proposed method compared with two other techniques. The latency is measured by the number of clock cycles after the error is injected until it is observed. The measured latency is averaged over 300 different random error injections. The first column in Table 4.2 shows the name of benchmark circuit and the second column shows the type of technique used. For comparison purposes, two different ways of selecting the signals to observe were used in addition to the proposed method. In one way, signals are randomly chosen and observed to detect circuit misbehavior. In the other way, signals are chosen using structural information in the following way. The size of each logic cone is sorted and the flip-flops that are fed by the largest logic cones are selected. Debug was performed with the three methods: random, structure-based, and the proposed method to compare the efficiency of the silicon debug. For the proposed method, three different threshold values for the maximum number of signals to merge is used which are shown in the third column. As can be seen, the proposed method detects the erroneous data more rapidly in all cases. It can also be seen that the more signals that are merged, the faster debug process is achieved. These results show that careful signal selection can be used to increase the efficiency and speed of silicon debug.

Table 4.2 Average Erroneous Response Detection Latency Results for 300 Different Random Error Injections

Circuit	Type of Technique	Max Val	Average Detection Latency with Different Number of Signal Groups			
			8	12	16	32
s9234	Random	N/A	320.81	212.18	186.82	173.73
	Structure	N/A	244.28	197.71	176.82	173.01
	Proposed	8	49.36	41.84	41.06	20.52
		12	49.36	41.84	41.06	20.52
		∞	49.36	41.84	41.06	20.52
s38584	Random	N/A	197.56	184.46	131.85	109.87
	Structure	N/A	178.08	146.73	127.32	115.86
	Proposed	8	67.35	61.44	51.87	31.41
		12	59.42	54.13	49.04	19.67
		∞	10.24	6.62	0.58	0.34
NOC	Random	N/A	594.42	571.87	574.52	504.65
	Structure	N/A	643.26	551.66	541.23	492.24
	Proposed	8	201.56	153.68	124.08	68.37
		12	148.97	117.79	109.80	45.21
		∞	47.63	21.79	16.37	9.12

4.5 CONCLUSIONS

In this chapter, an automated procedure for selecting which signals to observe is proposed for more efficient silicon debug. The set of signals selected by the proposed method are most often sensitized to possible errors and they maximally cover the errors within given constraints. The result shows that the proposed method can detect the faulty response rapidly and can increase the effectiveness of DFD hardware.

It should also be noted that the proposed technique could be universally applied to any designs including those which do not have scan chains with non-destructive scan out capability.

Chapter 5: Test Point Insertion Using Functional Flip-Flops to Drive Control Points

This chapter presents a novel method for reducing the area overhead introduced by test point insertion. Test point locations are calculated as usual using a commercial tool. However, the proposed method uses functional flip-flops to drive control test points instead of test-dedicated flip-flops. Logic cone analysis that considers the distance and path inversion parity from candidate functional flip-flops to each control point is used to select an appropriate functional flip-flop to drive the control point which avoids adding additional timing constraints. Reconvergence is also checked to avoid degrading the testability. Experimental results indicate that the proposed method significantly reduces test point area overhead and achieves essentially the same fault coverage as the implementations using dedicated flip-flops driving the control points.

5.1 INTRODUCTION

Built-in self-test (BIST) involves the use of on-chip test pattern generation and output response analysis. BIST provides a number of important advantages including the ability to apply a large number of test patterns in a short period of time, high coverage of non-modeled faults, minimal tester storage requirements, can apply tests out in the field over the lifetime of the part, and a reusable test solution for embedded cores. The most efficient logic BIST techniques are based on pseudo-random pattern testing. A major challenge is the presence of *random-pattern-resistant (r.p.r.) faults* which have low detection probabilities and hence may limit the fault coverage that can be achieved with pseudo-random patterns. There are two approaches for detecting r.p.r. faults: either modify the pattern generator so that it generates patterns that detect them, or modify the circuit-under-test to eliminate the r.p.r. faults.

A number of techniques have been developed for modifying the pattern generator. These include weighted pattern testing [Schnurmann 75], [Wunderlich 87], [Pomeranz 92], [Bershteyn 93], [Kapur 94], [Jas 01], [Lai 05], pattern mapping [Chatterjee 95], [Touba 95a, 95b], bit-fixing [Touba 96], bit-flipping [Wunderlich 96], and LFSR reseeding [Konemann 91, 01], [Hellebrand 92, 95], [Krishna 01], [Rajski 02].

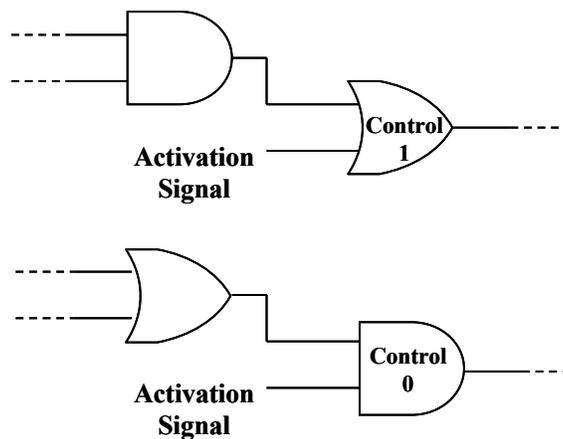


Figure 5.1 Example of Control Points

The other approach is to modify the circuit-under-test (CUT) by inserting test points [Eichelberger 83]. Test points are very efficient for eliminating r.p.r. faults and improving the fault coverage. Test point insertion (TPI) involves adding control and observation points to the CUT. Observation points involve making a node observable by making it a primary output or sampling it in a scan cell. Control points involve ANDing or ORing a node with an activation signal as illustrated in Fig. 5.1. When the activation signal is a '1', it controls the node to a 0 (1) for a control-0 (control-1) point. Typically the activation signal is driven by a dedicated flip-flop which receives pseudo-random values during BIST and is set to a non-controlling value during normal operation. Test points are added to the circuit before layout so that the performance impact can be

minimized. Circuit re-structuring is routinely used during layout to take into account additional delay due to metal wires.

Since test points add area and performance overhead, an important issue for test point insertion is where to place the test points in the circuit to maximize the coverage and minimize the number of test points. Optimal placement of test points in circuits with reconvergent fanout has been shown to be NP-complete [Krishnamurthy 87]. A number of approximate techniques for placement of test points have been developed using fault simulation [Briers 86], [Iyengar 89], path tracing [Touba 96], or testability measures [Seiss 91] to guide them. Timing driven test point insertion [Cheng 95], [Tsai 98] avoids inserting control points on critical timing paths.

Some research has investigated more efficient ways to drive the activation signals for the control points. In [Tamarapalli 96], the entire test is partitioned into multiple phases by a divide and conquer method, and control points are activated only during certain phases and deactivated during other phases. This provides greater control over the interaction of the control points with each other which can help reduce the total number of test points required. [Youseff 93] and [Nakao 99] propose methods for having one dedicated flip-flop drive the activation signal for multiple control points, i.e., sharing the dedicated flip-flops among the control points to reduce the total number of dedicated flip-flops that are required.

In spite of the efforts to reduce the overhead for TPI, the International Technology Roadmap for Semiconductors (ITRS) [ITRS 07] predicts that logic BIST for random patterns will take about 3.1% of chip area whereas the area for test compression will vary from 1.1% to 1.7%. One unpublished industrial design evaluation shows that logic BIST adds 1.34% to the chip area of which about 30% is related to the test points (0.4% to the chip area). And the other data from the unpublished industrial design evaluation

indicates that 2.68% chip area is increased by logic BIST and test points take 1.16% chip area. This suggests that test points correspond to 43% of the area increase in logic BIST. Test point area may vary depending on the circuit characteristics, the number of pseudo-random patterns used, and the fault coverage required. However, a considerable portion of the BIST area is usually related to test points, so it is important to find new techniques that can reduce the area overhead.

In this chapter, a new method for reducing the area impact of test point insertion is proposed by removing the dedicated flip-flops used for driving the control points. As will be shown in Sec. 5.5, and shown earlier in [Seiss 91], more than half of the test points, inserted are control points. Hence, replacing the dedicated flip-flops used to drive the control points with functional flip-flops in the design can significantly reduce area overhead. In the proposed test point implementation method, the location of the test points can be determined using existing test point insertion techniques such as the one described in [Seiss 91]. The new software identifies functional flip-flops which are suitable to drive the control point. Only functional flip-flops in the fan-in of the control point are considered as candidates to ensure that no new timing constraints are introduced between any two flip-flops. The method inherently introduces reconvergent paths sourced by the candidates flip-flops and has the potential to introduce redundant faults. Redundancies are avoided by taking into account the path inversion parity of the reconvergent paths. The proposed method essentially achieves the same fault coverage as an implementation based on dedicated flip-flops, but with lower area cost. The method is neutral with respect to the handling of unknowns in the circuit and test power as it does not deal with the selection of the test points, only their implementation.

This chapter is organized as follows. Sec. 5.2 gives an overview of the proposed scheme. Sec. 5.3 describes the control point replacement flow in detail. Sec. 5.4

discusses a technique to increase the fault coverage. Experimental results are shown in Sec. 5.5 and conclusions are given in Sec. 5.6.

5.2 OVERVIEW OF PROPOSED SCHEME

As mentioned earlier, area is the main issue for TPI. This section gives an overview of the main idea for significantly reducing the area without losing testability and introducing additional timing constraints.

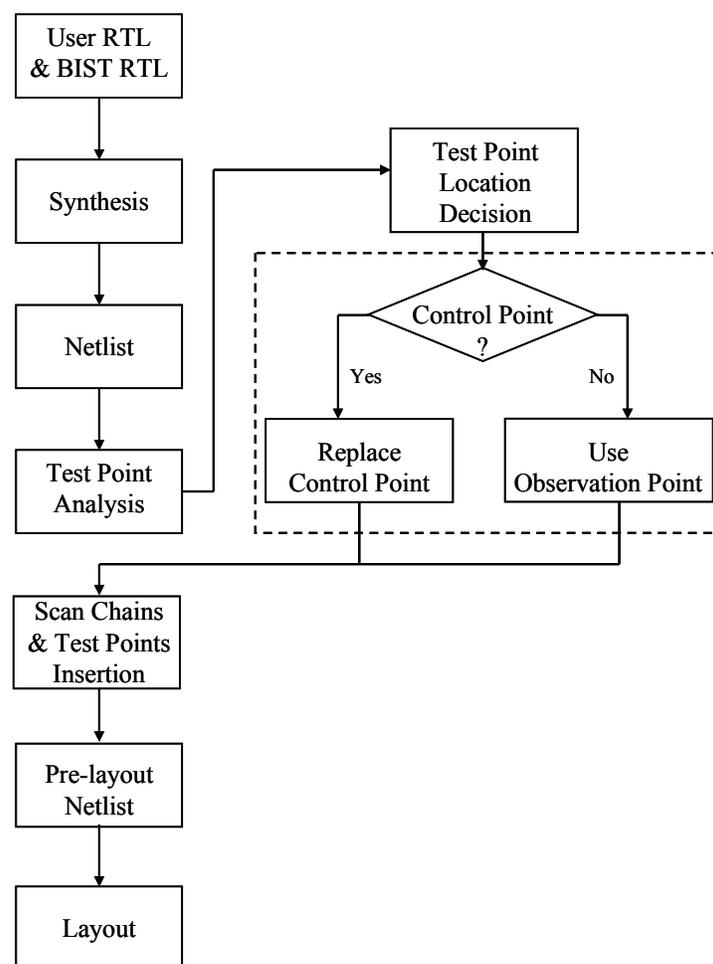


Figure 5.2 Proposed Design Synthesis Flow with Testability and Area Overhead Minimized Test Point Insertion

Fig. 5.2 shows a design synthesis flow that incorporates scan, BIST, and test point insertion. The conventional flow ends after test point insertion and generates the final design netlist. When test points are inserted, dedicated flip-flops are assigned to drive the control points and capture the observation points to achieve higher fault coverage. The idea proposed here is to minimize the area overhead by replacing the dedicated flip-flops for driving control points with existing functional flip-flops in the design. The test points are first inserted with any TPI algorithm [Briers 86], [Cheng 95], [Iyengar 89], [Nakao 93], [Seiss 91], [Touba 96], [Youseff 93]. Then the proposed method performs a post-processing step in which functional flip-flops are identified for driving the control points via logic cone analysis. The observation points are not modified. The dashed box in Fig. 5.2 indicates the post-processing flow that finds and replaces the control points to generate the netlist.

In a BIST application, the activation signal is controlled by a flip-flop scanned in with a pseudo-random value for each scan vector. During system operation, and the activation signal is set to its non-controlling value so that the functional logic value can pass through the control gate. Therefore, when a dedicated flip-flop is replaced by a functional flip-flop, it should keep the same property of not changing the system function. For this purpose, a global signal “*TP_Enable*” is introduced. *TP_Enable* enables and disables the control points. When *TP_Enable* is ‘1’, a control point is driven by a functional flip-flop in the proposed method. Note that this creates a new timing path from the functional flip-flop to the control point.

The functional flip-flops which are “logically” near the control point are chosen as candidates to replace a dedicated control point flip-flop for two reasons. The first reason is to minimize the length of the newly created test path from the candidate flip-flop to the control point. The second reason is that the transitions through the control

point will have roughly the same delay as those along the functional path from the selected functional flip-flop. As will be explained in detail in Sec. 5.3.2, the proposed method does not create any relationships between control points and unrelated registers, and hence no new timing constraints are introduced.

Since the proposed method does add additional primitive gates to a design, it can impact the testability of the design. Hence, the following rules need to be observed to minimize the number of redundant or untested faults introduced by circuit modification.

1. Maintain opposite path inversion parity along the paths from a functional flip-flop to a control point: There are two paths from a functional flip-flop to the control point. One is an existing functional path from a functional flip-flop to a control point and the other is the newly introduced path which is ANDed with the *TP_Enable* signal. Having opposite inversion parity along these two paths makes a path testable by appropriately applying either '0' or '1'. This is described in Sec. 5.3.3.1.
2. Check for illegal reconvergence from the candidate functional flip-flop: Hard to test faults could be blocked by a fanout of a test point if the functional flip-flop (which drives the test point) drives some gate in a fanout of a test point. This case needs to be avoided. This is described in Sec. 5.3.3.2.

5.3 DETAILS OF CONTROL POINT REPLACEMENT FLOW

The following subsections describe each of the steps in the proposed method for replacing the dedicated flip-flops with functional flip-flops to drive the control points.

5.3.1 Finding Candidate Functional Flip-flops

Fig. 5.3 is an example of conventional test point insertion. This circuit has flip-flops (denoted *A* to *I*) and combinational elements (denoted *G1* to *G17* and *Ctrl*). It has

one control point highlighted in gray color (*Ctrl*) and a dedicated flip-flop *I* drives the control point in test mode. Flip-flops *A* to *H* are the functional flip-flops that are used for a system operation. During the system operation, *Ctrl* is made transparent by resetting flip-flop *I* so that the value in *G10* can be transferred to the one input of *G12* without any change. And when the control point is activated, the output of gate *Ctrl* is fixed to a '1' (i.e., *control-1 point*). If an AND gate is used as *Ctrl*, the output of *Ctrl* is fixed to a '0' when it is activated (i.e., *control-0 point*).

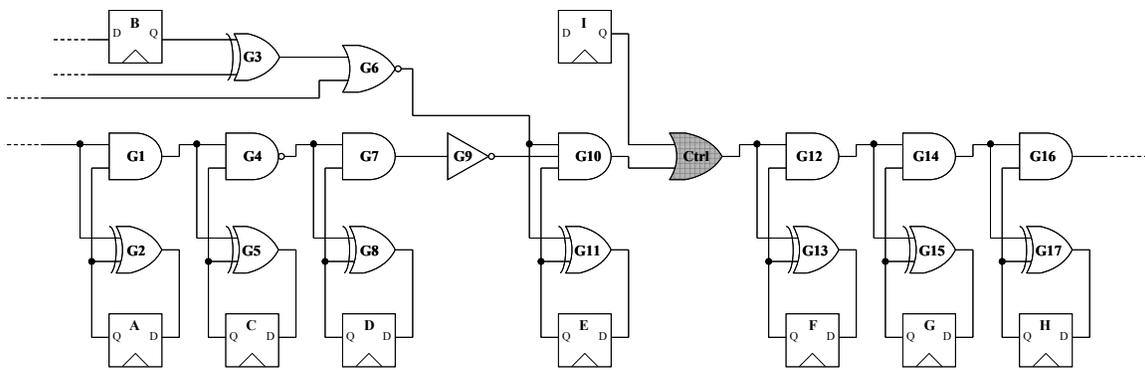


Figure 5.3 Example of a Circuit with Control Point Insertion (*Ctrl*)

To find the functional flip-flop for replacing the dedicated flip-flop, it is necessary to perform logic cone analysis. Logic cone analysis starts from the control point (*Ctrl*) and traces back to the flip-flops. In Fig. 5.3, the search initiates from *Ctrl*. Even though *I* is the first flip-flop found, since it is dedicated for the control point, *I* needs to be dropped from the search space. Hence, *G10* is the first gate visited. In a depth first search manner (a breadth first search can also be used), *G6* is visited next and then *G3* is visited. Flip-flop *B* is found along this branch from *Ctrl*. In the same fashion, flip-flop *A*, *B*, *C*, *D*, and *E* are found as candidates for replacing *I*.

While the nodes are traversed when searching, the inversion parity information is also checked. *G4*, *G6* and *G9* are inverting gates. Since *G6* is inverting, the flip-flop *B*

has odd inversion parity to the control point. Flip-flop *A*, *C* and *E* have even inversion parity, and *B* and *D* have odd parity along their paths to the control point. If both inversion parities are found along paths from the control point to one flip-flop, that flip-flop is discarded from the candidate list. For example, if during logic cone analysis, a flip-flop is found to have one path with one inversion, and another path with two inversions, it is not considered as a candidate. Some elements such as XOR gates and MUXes always have both non-inverting and inverting paths (dual polarity). Gates with dual polarity are considered as non-inverting gates. Further analysis on the dual polarity will be discussed in Sec. 5.5.

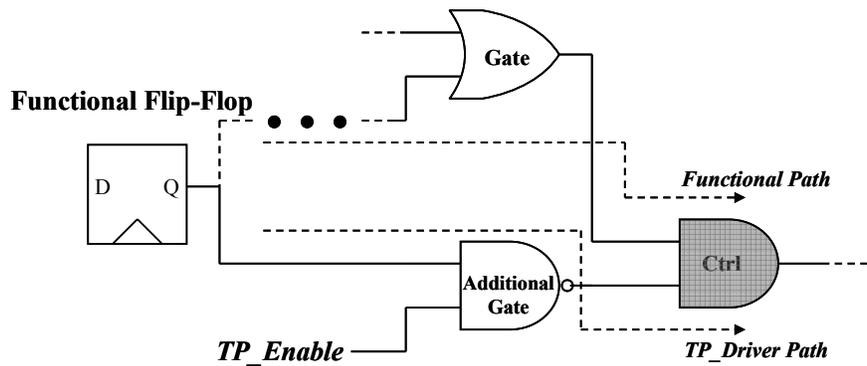
As shown in Sec. 5.2, a new timing path is a matter of concern in selecting a functional flip-flop to replace a dedicated flip-flop. Therefore, logical distance information needs to be considered so as not to introduce any delay paths that add performance overhead. The functional flip-flop distance to a control point can be measured based on the number of levels of logic. The logical distance is used to maximize the probability for the test point driver to be relatively close to the test point to minimize the length of the wires. The following shows the results of logic cone analysis for Fig. 5.3.

<u>Candidate Flip-Flop</u>	<u>Inversions</u>	<u>Logical Distance</u>
<i>A</i>	2	5
<i>B</i>	1	3
<i>C</i>	2	4
<i>D</i>	1	3
<i>E</i>	0	2

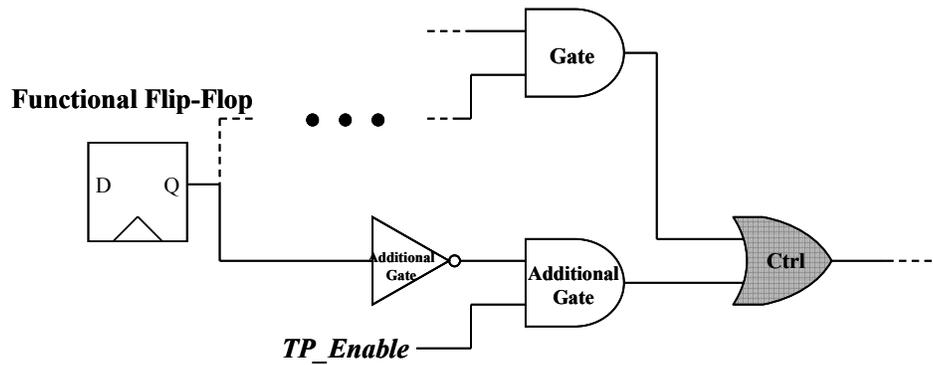
There may be cases when only one functional flip-flop is found as a candidate by logic cone analysis. This occurs when a test point has a single functional flip-flop in its

fan-in. This happens when the controllability to a certain value ('0' or '1') needs to be higher than 0.5. In this case, a dedicated flip-flop cannot be replaced.

5.3.2 Selecting Candidate Flip-Flop



(a) Conventional Control Point with Dedicated Flip-Flop



(b) Example of Proposed Control Point with Functional Flip-Flop

Figure 5.4 Conventional and Proposed Control Point

Assume that a dedicated flip-flop is replaced by one of the functional flip-flops among *A* to *E*. If a functional flip-flop directly drives the control point, it affects the system function. To hold the transparency property during system operation, one global signal called “*TP_Enable*” is introduced and it is deactivated during system operation. Fig. 5.4(a) shows an example of a conventional control point that uses a dedicated flip-

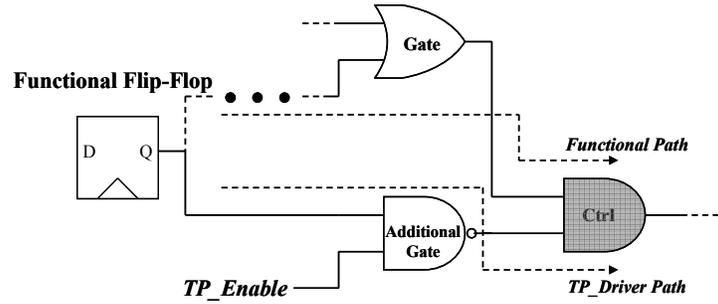
flop. Fig. 5.4(b) illustrates an example of the proposed control point that is driven by a functional flip-flop. The functional flip-flop not only drives the AND gate at the bottom but also operates as a test point driver via the additional gate path in Fig. 5.4(b) (this flip-flop can be named as *TP_Driver*). The *TP_Enable* signal can block the signal propagation by setting its value to '0'. This places a non-controlling value at the input of the control point. When *TP_Enable* is '1', *Ctrl* can have a value determined by a functional flip-flop.

In Sec. 5.3.1, *E* is found to be the closest flip-flop to the control point location with a distance '2' and it is now chosen as the *TP_Driver*. Since flip-flop *E* is already in the fan-in of the control point, no new timing relationships are created with flip-flops in the fan-out of the test point. Note that from a testability point of view, it would have been preferable to use a flip-flop that is not in the fan-in of the control point to avoid the correlation between the two inputs of the control point. However, new timing relationships would be created between functionally unrelated flip-flops and this is not acceptable. The consequences of the "no new timing relationships" rule are analyzed next.

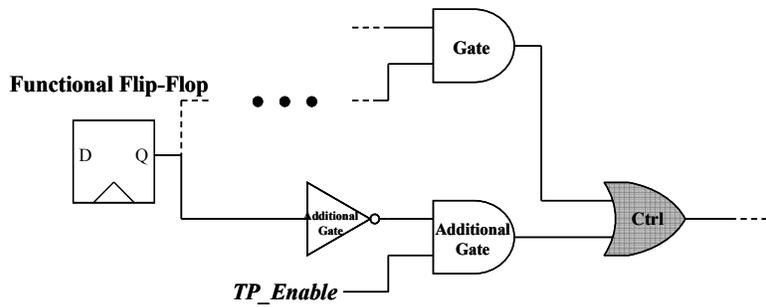
5.3.3 Testability Consideration

The proposed method modifies the CUT to try to maximize the random pattern testability. The following subsections describe the rules that need to be considered for improving testability.

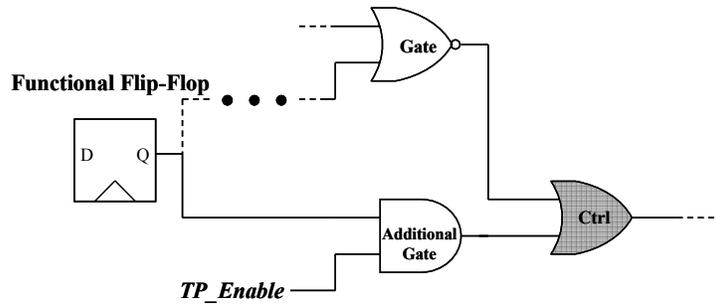
5.3.3.1 Path Inversion and Control Point Structures



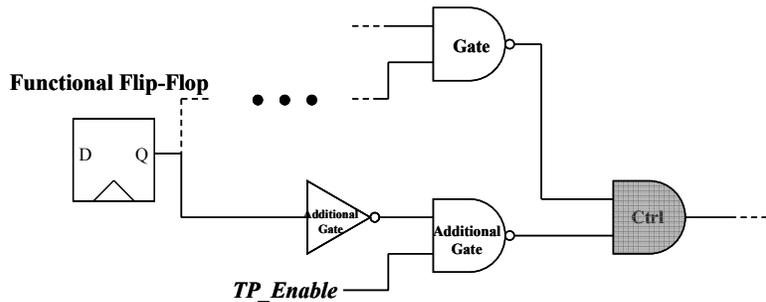
(a) *Type 1: Non-Inverting Functional Path with AND Ctrl*



(b) *Type 2: Non-Inverting Functional Path with OR Ctrl*



(c) *Type 3: Inverting Functional Path with OR Ctrl*



(d) *Type 4: Inverting Functional Path with AND Ctrl*

Figure 5.5 New Types of Control Point Structure for Different Path Inversion Parity

If a functional flip-flop is chosen to replace a dedicated flip-flop for the control point, a new path is created from the functional flip-flop to the control point. This new path will be referred to as the “*TP_Driver Path*”. The original functional path will be referred to as the “*Functional Path*”. A value in *Functional Path* can only propagate when *TP_Enable* is disabled in Fig. 5.4(b). However, the opposite inversion parity between the *TP_Driver Path* and *Functional Path* can enable propagation through *Functional Path* without disabling the test point. This increases the random pattern testability and helps to reduce the number of test patterns needed compared to having the same inversion parity along the two paths. Considering that either an AND or OR gate can be used for creating a control point, there are 4 types of control points that satisfy the inversion parity as shown in Fig. 5.5.

In Fig. 5.5(a) and 5.5(b), the *TP_Enable Path* needs to have inversion because *Functional Path* has a non-inverting path. Fig. 5.5(c) and 5.5(d) show the control point with an inverting path on *Functional Path*. When an inverter is added in the *TP_Driver Path* as in Fig. 5.5(b) and 5.5(d), either an inverter can be used or the flip-flop’s *Q_bar* can be connected to the additional gate.

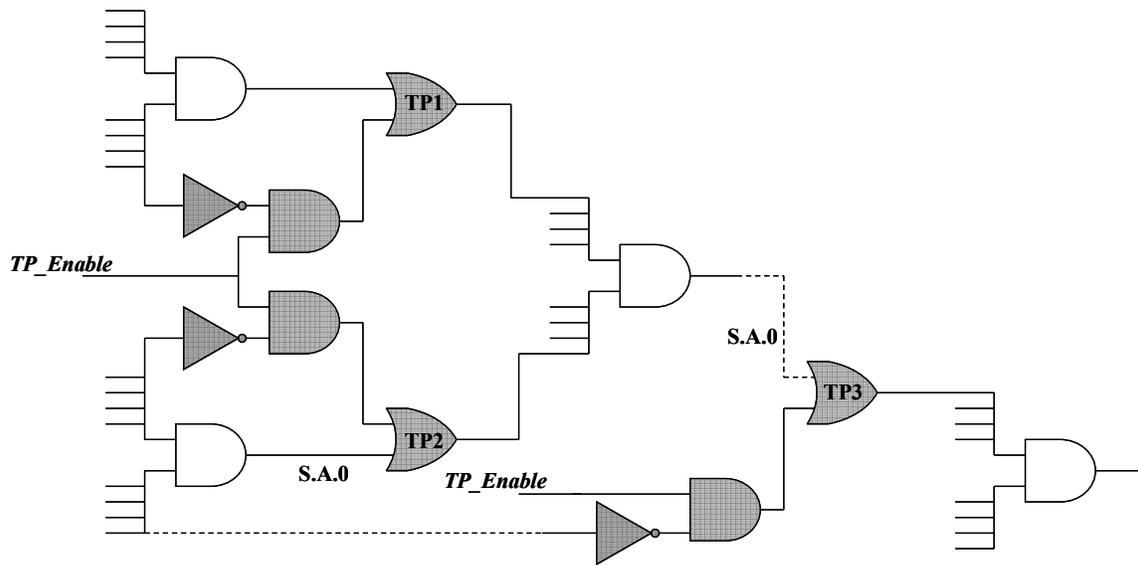


Figure 5.6 AND Tree Example with Proposed Control Point Structure

Fig. 5.6 shows an AND tree example with the proposed control point structure. Assume the path inversion is not considered, no inverter would be inserted on the *TP_Driver Path* when a dedicated control point driver flip-flop is replaced, as in Fig. 5.4(b). In this case, hard to test faults have a small probability of being able to propagate through the circuit if all test points are disabled ($TP_Enable = 0$). For example, the stuck-at-0 fault (S-A-0) at the input of TP2 is one of the hard to detect faults, and it would preferably need TP1 to be active in order to propagate S-A-0 through the AND gate which is located after TP1 and TP2, and then to propagate through TP3 and the remainder of the circuit. However, when path inversion is considered, the control point structure will solve this problem. S-A-0 at the input of TP2 could propagate even when TP_Enable is '1'. All inputs of the AND gate should be '1' to provoke S-A-0 at TP2, and it automatically sets the AND gate with a controlling value (0). This makes the control point disabled without setting TP_Enable to be '0'. TP_Enable reconvergence could be an issue in the proposed method, however, the path inversion

analysis solves this problem so that the testability is not degraded. Hence, they are detected relatively easier than not having the path inversion information. The other hard to detect faults in Fig. 5.6 are the S-A-1 faults on the TP_Enable input of the AND gates disabling the test points. Since it is unlikely to randomly detect the faults on TP_Enable branches, automatic test pattern generation (ATPG) patterns need to be used for detecting most of these faults. However, they represent a very small percentage of the total number of faults as we will see in Sec. 5.5.

5.3.3.2 Illegal Reconvergence

Logic cone analysis determines the functional flip-flop candidates for driving control test points. For testability, since there may be many connections from the functional flip-flops to other nodes in a circuit, it is necessary to check whether the fault propagation is blocked. Reconvergence from TP_Driver in the fanout of a control point can block the fault propagation. If any gate in the fanout of a control point is sourced by TP_Driver , it can obstruct fault propagation and it may result in the loss of testability.

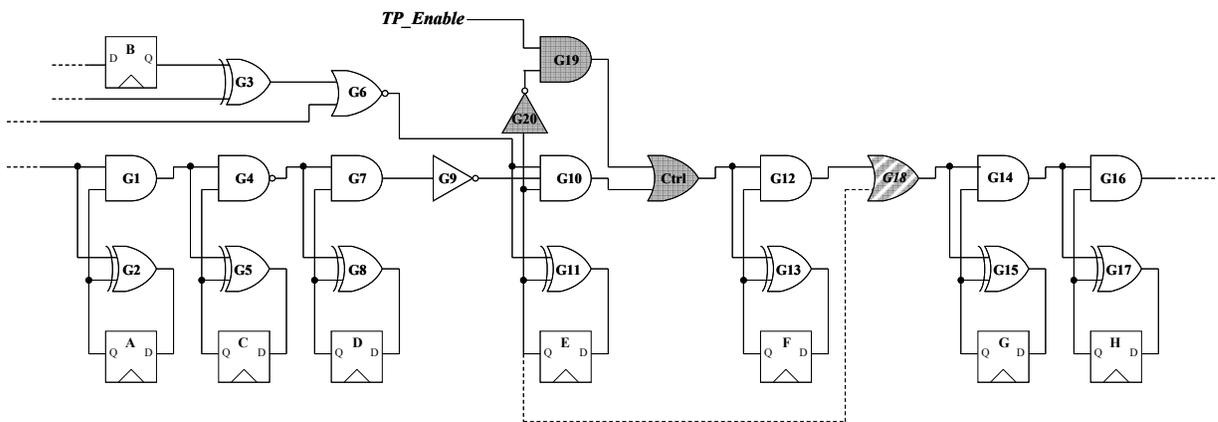


Figure 5.7 Example of a Circuit with Illegal Reconvergence

Fig. 5.7 illustrates an example which is almost the same as a circuit in Fig. 5.2 with the exception of OR gate $G18$. As shown in the previous sections, E is selected as a TP_Driver based on the distance and a *Type 2* control point is inserted to satisfy the inversion parity requirement. Assume E has a branch to $G18$ illustrated as a dashed line from E to $G18$. This forms reconvergence from E to a fanout of a control point. Due to the reconvergence, whenever E has '1', it drives $G18$ with a controlling value and this may block the fault propagation.

Illegal reconvergence analysis removes E from the candidate list, and the next closet flip-flop is chosen. In Fig. 5.7, B and D have a distance 3 and they do not introduce a longer timing path than the existing longest path. There is no reconvergence from B or D to the fanout of the control point, so they do not violate the illegal reconvergence condition. Since B and D have the appropriate inversion parity and an OR control point is used, a *Type 3* control point needs to be applied when replacing the dedicated flip-flop. If there is no flip-flop that satisfies the conditions for replacement, a dedicated flip-flop cannot be replaced.

5.4 TP_ENABLE SIGNAL PROBABILITY

Test points are generally assumed to be always enabled with a controllability of 0.5. Therefore, TP_Enable will generally have a value of '1'. However, the stuck-at-1 fault on TP_Enable can only be detected when the TP_Enable signal is set to '0'. To detect this fault, TP_Enable needs to take on a value of '0' some times. To investigate what the optimal signal probability for TP_Enable is, experiments were performed using different input size OR gates to bias the signal probability of TP_Enable . If two equiprobable pseudo-random signals are ORed together, the signal probability is increased to 0.75. In the general case, driving the TP_Enable signal by a k input OR gate achieves a $(2^k - 1)/(2^k)$ signal probability.

Different *TP_Enable* signal probabilities change the controllability on the control points and detectability of the stuck-at-1 fault on the *TP_Enable* signal. In Sec. 5.5, experimental results are shown for different signal probabilities for the *TP_Enable* signal.

5.5 EXPERIMENTAL RESULTS

In this section, experimental results are presented for six industrial designs and *OR1200* (OpenRisc Processor) [OR1200] and an *NOC* design [Jang 08]. The LogicVision testpointAnalyze tool [LogicVision] was used to determine the location of test points in each design. The post-processing tool described here was used to determine the functional flip-flops that could be used as test point drivers.

Table 5.1 Area Overhead Reduction Results

Design	Conventional Test Point Insertion		Proposed Control Point Replacement Method		Reduction Ratio (%)	Test Point Area Reduction (%)
	Num. Observation Points	Num. Control Points	Num. Dedicated Flip-Flops	Num. Functional Flip-Flops		
Design A	3	24	2	22	91.7 %	61.6 %
Design B	2	85	2	83	97.7 %	71.6 %
Design C	104	233	29	204	87.6 %	45.4 %
Design D	70	179	39	140	78.2 %	42.2 %
Design E	221	1424	794	630	44.2 %	28.7 %
Design F	129	371	13	358	96.5 %	53.7 %
OR1200	5	27	0	27	100 %	63.0 %
NOC	9	35	0	35	100 %	59.4 %

In Table 5.1, the number of dedicated flip-flops that are replaced by functional flip-flops using the proposed method is shown. The first column gives the design name. The number of observation points and control points calculated is shown in the second and third column, and the summation of both columns is the total number of test points. The fifth column shows the number of dedicated flip-flops that are replaced by functional flip-flops using the proposed method. As explained in Sec. 5.3, if there is only one functional flip-flop in the candidate list or no candidate meets the rules, a dedicated flip-

flop cannot be replaced. The number of dedicated flip-flops which could not be replaced is shown in the fourth column. The sixth column shows the reduction ratio which is computed as the number of functional flip-flops used to replace dedicated flip-flops (the fifth column) divided by the number of total control points (the third column). These results show a significant area reduction by replacing the dedicated flip-flop using the proposed method. The last column represents the test point area reduction ratio. 130nm TSMC technology is used for OR1200 and NOC synthesis. The proposed method achieves 63.0% and 59.4% of test point area reduction in OR1200 and NOC respectively, when the dedicated flip-flops are replaced by functional flip-flops. Because we do not have access to the netlist for Design A – F, their results are extrapolated based on the results from OR1200 and NOC. In OR1200 and NOC, each of the new control points driven by a functional flip-flop takes approximately 1/4 of the area of the original control points driven with a dedicated flip-flop. Therefore, the extrapolated area for Design A – F can be calculated based on the following equation.

$$\frac{New\ Area}{Old\ Area} = \frac{Nobs + Ndedicated + k * Nfunctional}{Nobs + Ndedicated + Nfunctional}$$

$$= 1 - Area_reduction$$

where *Nobs* denotes the number of flip-flops for observation points, and *Ndedicated* and *Nfunctional* indicate the number of dedicated flip-flops and functional flip-flops used for control point respectively. Since the *k* factor is approximately 0.25 for both OR1200 and NOC, we calculate the area reduction of Design A – F.

In Table 5.2, a fault coverage comparison is shown between the proposed test point implementation method and the standard LogicVision implementation. The number of test points inserted and the number of control points replaced are given in Table 5.1. The first column of upper and lower tables in Table 5.2 gives the design

name. The total number of faults is illustrated in the second column of the upper table. There are three different cases for which results were generated. These were when no test points are inserted (*NO TP*), test points are inserted with dedicated flip-flops (*Dedicated F/F*), and when the proposed method is used to replace dedicated flip-flops with functional flip-flops (*Proposed*). The proposed method was tried with three different *TP_Enable* signal probabilities ($1/2$, $15/16$, and $63/64$) to evaluate the random pattern testability. Since TPI adds extra gates in a design, more faults exist for the *Dedicated F/F* and *Proposed* cases than for *No TP*. When a dedicated flip-flop is replaced, the *type 1-4* control point structures in Fig. 5.5 are used. These structures add a few combinational gates in a design. Since the fault simulator does not consider internal faults of flip-flops, it appears that the proposed method has more faults than the standard implementation even though there are less. The third column of the upper table shows the number of redundant faults, and the second column of the lower table represents the number of aborted faults. Testability is a matter of interest in the proposed method and the experimental results show that one more redundant fault is found in Design B, and the rest of the cases have the same number of redundant faults as for the *Dedicated F/F* case. Because Designs A to F are random pattern resistant circuits, 100000 random patterns are applied to get the fault coverage in the third column of the lower table. Since the OR1200 and NOC designs are found to be relatively random pattern testable circuits, 2048 random test patterns are applied and the coverage is shown. The small fault coverage difference, about 0.05% ~ 0.1% from most of the benchmark circuits, between *Dedicated F/F* and *Proposed* with $15/16$ signal probability essentially corresponds to the number of faults added by the new test points that can only be detected when *TP_Enable* is '0'. Those faults, the faults on the *TP_Enable* branches, are very difficult to detect randomly and will require ATPG patterns. The Fault

coverage loss (0.4%) in Design F can be compensated by a combination of three options – applying more random patterns, calculating more top up patterns or adding more test points. In our experiment, either applying 100K more random patterns or 73 additional top up patterns could fully compensate the coverage loss and the coverage reached 98.27%.

Table 5.2 Testability Comparison of Proposed Method with Standard Implementation

Design	Num. of Faults			TP_ Enable Probability	Num. of Redundant Faults			TP_ Enable Probability
	No TP	Dedicated F/F	Proposed		No TP	Dedicated F/F	Proposed	
Design A	92631	92689	92726	1/2	186	186	186	1/2
			92739	15/16			186	15/16
			92741	63/64			186	63/64
Design B	20955	21131	21294	1/2	32	19	20	1/2
			21297	15/16			20	15/16
			21299	63/64			20	63/64
Design C	505254	506120	506457	1/2	2730	2559	2559	1/2
			506460	15/16			2559	15/16
			506462	63/64			2559	63/64
Design D	245688	246294	246474	1/2	1813	1690	1680	1/2
			246477	15/16			1683	15/16
			246479	63/64			1683	63/64
Design E	1665662	1299661	1300549	1/2	1813	1690	2136	1/2
			1300552	15/16			2134	15/16
			1300554	63/64			2134	63/64
Design F	500405	381139	381599	1/2	2706	1608	1647	1/2
			381602	15/16			1649	15/16
			381604	63/64			1649	63/64
OR1200	98690	98906	98984	1/2	2	0	0	1/2
			98987	15/16			0	15/16
			98989	63/64			0	63/64
NOC	56277	56383	56455	1/2	4	4	4	1/2
			56458	15/16			4	15/16
			56460	63/64			4	63/64

Design	Num. of Aborted Faults			TP_Enabled Probability	Fault Coverage (%)			TP_Enabled Probability
	No TP	Dedicated F/F	Proposed		No TP	Dedicated F/F	Proposed	
Design A	5	5	5	1/2	88.19	93.35	91.99	1/2
			5	15/16			93.28	15/16
			5	63/64			93.30	63/64
Design B	0	0	0	1/2	89.08	94.37	94.32	1/2
			0	15/16			94.82	15/16
			0	63/64			94.80	63/64
Design C	2738	601	600	1/2	95.68	98.90	98.84	1/2
			600	15/16			98.84	15/16
			600	63/64			98.83	63/64
Design D	936	683	709	1/2	95.05	98.71	98.61	1/2
			683	15/16			98.63	15/16
			690	63/64			98.61	63/64
Design E	1028	630	679	1/2	80.19	99.26	99.14	1/2
			681	15/16			99.23	15/16
			682	63/64			99.15	63/64
Design F	115	113	102	1/2	87.16	98.25	97.71	1/2
			114	15/16			97.86	15/16
			114	63/64			97.77	63/64
OR1200	15	22	28	1/2	93.18	98.96	98.44	1/2
			28	15/16			98.86	15/16
			30	63/64			98.86	63/64
NOC	214	0	13	1/2	97.78	99.41	99.40	1/2
			11	15/16			99.42	15/16
			11	63/64			99.42	63/64

An analysis of the dual polarity gates revealed that considering MUX primitives as non-inverting gates is not optimal. This is because paths going through the select input have an implied dual polarity. For example, in Fig. 5.5(a) illustrating a Type 1 control point, suppose that OR Gate (“Gate”) is a MUX primitive and its select input is directly connected to the candidate functional flip-flop output, then the stuck-at-0 fault on the select input becomes impossible to detect without setting TP_Enable to 0. Changing the control point for a Type 4 control point does not improve the situation as it makes the stuck-at-1 the hard to detect fault instead. Therefore, candidate flip-flops with a path going through the select input of a MUX primitive should be discarded. However, this non-optimal MUX primitives management had no impact on the experimental results of 5 of the 8 circuits since MUXes were implemented or modeled as AND/OR structures in

those circuits. Design D, E and F have approximately 13,000, 38161 and 15612 instances of MUXes modeled as MUX primitives, however, the results for these designs show that the testability seems similar to that of other designs. Hence, in the experiments, the testability is not significantly affected by this issue.

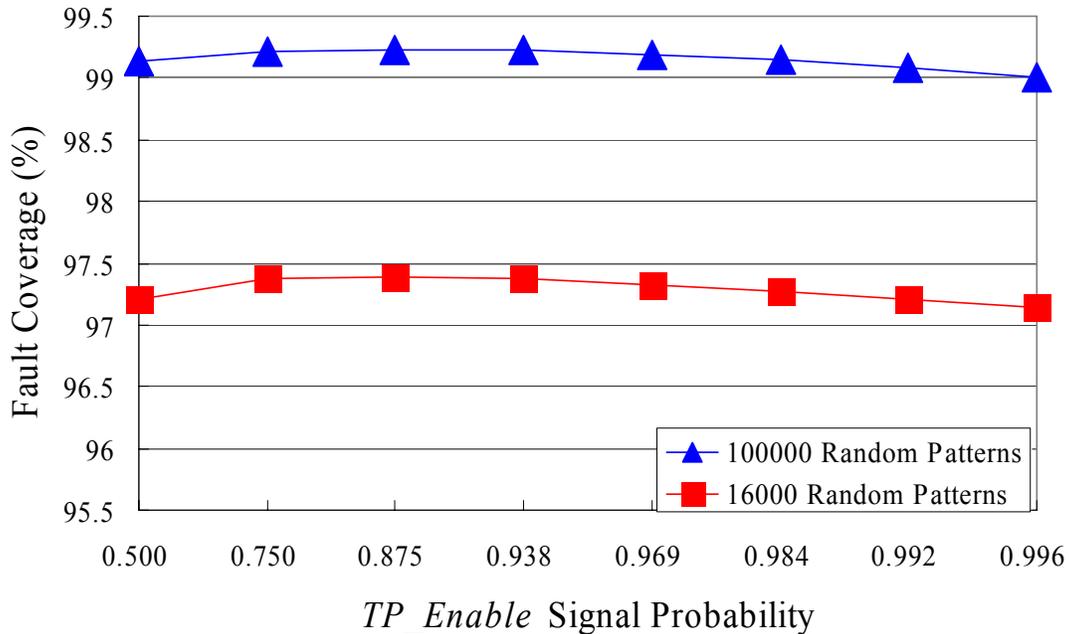


Figure 5.8 Testability vs. TP_Enable Signal Probability

To study how sensitive the fault coverage is to the TP_Enable signal probability, Design E is used with different signal probabilities of $(2^k - 1)/(2^k)$ for $k = 1$ to 8. 16000 and 100000 random patterns are applied to achieve random fault coverage. As can be seen in Fig. 5.8, the TP_Enable signal probability gives a significant improvement in the fault coverage. The fault coverage is increased about 0.5 % only by changing the signal probability. This is to be expected since the testpointAnalyze tool assumes that the TP_Enable probability is exactly 1. Both cases illustrate that there is saturation of the

coverage. Therefore, the probability of TP_Enable needs to be kept high, say 15/16 or 31/32, so that the efficiency of the original test point insertion method is not affected. In the proposed method, the maximum fault coverage is obtained in this way. Design A show that the coverage goes down a little when TP_Enable has a signal probability 15/16 compared with 63/64. This happens because of the noise related to vectors. When different vectors are applied to CUT, they can introduce the noise. This may result in the lower test coverage in the benchmark circuits.

5.6 CONCLUSIONS

The experimental results indicate that the methodology proposed in this chapter can significantly reduce the number of dedicated flip-flops for driving control points by replacing them with functional flip-flops. Significant area savings are therefore achieved while preserving the random pattern testability of the circuit and without introducing new timing constraints that would complicate timing closure. The test point area was typically reduced by half while the fault coverage loss during the random pattern phase was limited to less than 0.1% for most circuits. Several options were identified to compensate for a slightly higher coverage loss observed for 2 circuits.

The proposed method can be used to implement test points calculated with existing algorithms without having to modify the algorithms. The method is therefore neutral with respect to the handling of unknowns in the circuit and test power as it does not deal with the selection of the test points, only their implementation. It should also be noted that the new test point implementation method gives the flexibility of adding more test points to achieve even higher coverage or reduce test time.

The run times of our software implementing the test points were not monitored. The proposed method only involves static tracing of the fan-in and fan-out of gates which

are related to control points and very efficient algorithms are available for performing these tasks which are less complex than the algorithms used for test point selection itself.

Future work includes a more thorough investigation of circuit structures for which no suitable functional flip-flops could be identified to determine if a different test point implementation could be used. Also, a new implementation for observation points will be investigated. This will be useful for circuits for certain designs with a significant number of observation points.

Chapter 6: Reducing Test Point Area for BIST through Greater Use of Functional Flip-Flops to Drive Control Points

A new test point insertion method for pseudo-random built-in self-test (BIST) was proposed in [Yang 09] which tries to use functional flip-flops to drive control test points instead of adding extra dedicated flip-flops for driving the control points. This chapter investigates methods to further reduce the area overhead by replacing dedicated flip-flops which could not be replaced in [Yang 09]. A new algorithm (alternative selection algorithm) is proposed to find candidate flip-flops out of the fan-in cone of a test point. Experimental results indicate that most of the not-replaced flip-flops in [Yang 09] can be replaced and hence even more significant area reduction can be achieved with minimizing the loss of testability.

6.1 INTRODUCTION

Built-in self-test (BIST) embeds test pattern generation and output response analysis on-chip. It provides numerous advantages in terms of reducing test generation costs, reducing tester storage requirements, allowing the rapid application of many patterns to target non-modeled faults, test reuse, and in-field testing where there is no access to a tester. The most efficient BIST techniques are based on pseudo-random testing where the test patterns can be generated using a linear feedback shift register (LFSR) which has a very compact structure. However, the presence of *random-pattern-resistant (r.p.r.)* faults which have low detection probabilities may prevent pseudo-random BIST from achieving sufficiently high fault coverage. There are two general approaches for improving the fault coverage for pseudo-random BIST: (1) modifying the pattern generator so it generates patterns to detect r.p.r faults, and (2) modifying the circuit-under-test (CUT) to eliminate the r.p.r. faults by increasing their detection probability.

In pattern generator modification, a number of techniques have been proposed including weighted pattern testing [Pomeranz 92], [Bershteyn 93], [Kapur 94], [Jas 01], pattern mapping [Chatterjee 95], [Touba 95a, 95b], LFSR reseeding [Konemann 91, 01], [Hellebrand 92, 95], [Krishna 01], [Rajski 02] and others.

In CUT modification, test points are inserted [Eichelberger 83] to improve the fault coverage. Observability is enhanced by adding observation points and controllability on a particular node is enhanced by adding a control point. Control points insert AND or OR gates at a node and are activated by pseudo-random values from a dedicated flip-flop during BIST operation. Since test point insertion (TPI) adds extra gates in a design, area and performance overhead are issues. [Krishnamurthy 87] proved that finding optimal locations in circuits with reconvergent fanouts is NP-complete and hence there has been a lot of research on test point insertion techniques. Test point insertion methods based on fault simulation [Briers 86], [Iyengar 89], path tracing [Touba 96], and testability measures [Seiss 91] have been proposed.

In [Yang 09], a method was presented for reducing the area impact of TPI by removing dedicated flip-flops used for driving control points. Test points are inserted with any TPI algorithms and, in a post-processing step, dedicated flip-flops are replaced by functional flip-flops in a design. The replacement technique in [Yang 09] is constrained so that, by construction, it will not introduce any additional timing constraints and achieves basically the same fault coverage as conventional test point insertion using dedicated flip-flops. However, a drawback in [Yang 09] is that some dedicated flip-flops get marked as non-replaceable and thus limit the area overhead reduction that is achieved.

In this chapter, a new functional flip-flop selection replacement technique is proposed which provides a new selection algorithm to remove the not-replaced dedicated

flip-flops in [Yang 09]. Dedicated flip-flops cannot be replaced when no functional flip-flops satisfy the reconvergence rule and the path inversion parity rule and when control points requires controllability greater than 0.5. In the proposed method, an alternative selection algorithm tries to replace them by functional flip-flops without introducing new timing constraints.

Sec. 6.2 describes the overview of test point insertion method using functional flip-flops. Sec. 6.3 describes an alternative selection algorithm. Experimental results are shown in Sec. 6.4 and conclusions are given in Sec. 6.5.

6.2 OVERVIEW OF TEST POINT INSERTION USING FUNCTIONAL FLIP-FLOPS

This section gives a brief overview of the procedure for test point insertion using functional flip-flops to drive control points that was described in [Yang 09]. Area is generally the main issue for test point insertion. [Yang 09] showed a method for significant area reduction without losing testability and without introducing timing constraints. Dedicated flip-flops for control points are replaced by the functional flip-flops using logic cone analysis that considers the path inversion parity and distance information. To avoid additional timing constraints, the functional flip-flops in the fan-in of the control points are considered as candidate flip-flops for driving the control point because the new paths that are introduced cannot be longer than the existing functional paths by construction. Note, however, that when the dedicated flip-flops are replaced, primitive gates are added to a design and hence the testability may be affected by the circuit modification. The following rules were proposed in [Yang 09] to prevent the testability loss.

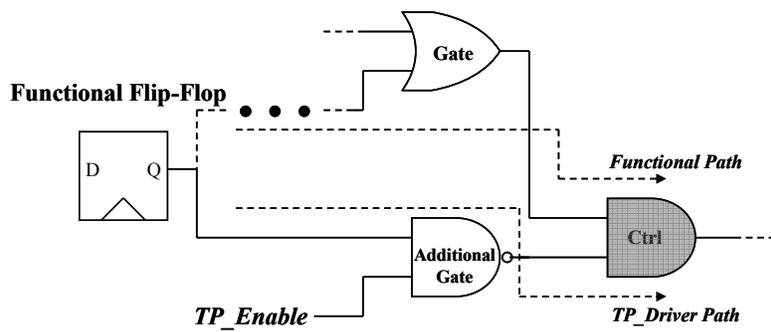
1. Illegal reconvergence from the candidate flip-flop must be checked - Reconvergence coming from the candidate functional flip-flop in the fanout

of a control point can block fault propagation. This needs to be considered and avoided.

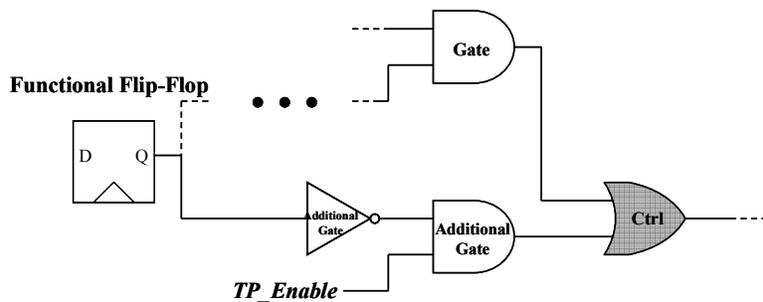
2. Opposite path inversion parity must exist along the paths from the functional flip-flop to the control point - Having opposite inversion parity along two paths (the existing functional path from the functional flip-flop to the control point and the new path created through the added test point enable, “TP_Enable”, primitive gate for the control point) ensures that a path is testable.

To take the two rules mentioned above into consideration, logic cone analysis is performed starting from the control point and searching the functional flip-flops in its fan-in. Both logical distance and path inversion information are checked. Distance information is used so as not to introduce any delay paths that impact timing. Therefore, the nearest functional flip-flop from the control point is considered first. As stated in rule 2, the testability can be maintained by strictly having opposite inversion parity along the existing functional path from the flip-flop and along the path through the TP_Enable gate. In the parity inversion analysis, if a flip-flop is found to have multiple existing functional paths with both inversion parities (i.e., one or more with an odd number of inversions, and one or more with an even number of inversions) then that functional flip-flop is discarded from the candidate list for driving the control point. The only functional flip-flops that can be considered as candidates are those for which all existing functional paths to the control point have the same inversion parity. Only for those flip-flops is it possible to satisfy rule 2 by making the newly created path through the TP_Enable gate have opposite inversion parity from the existing functional paths.

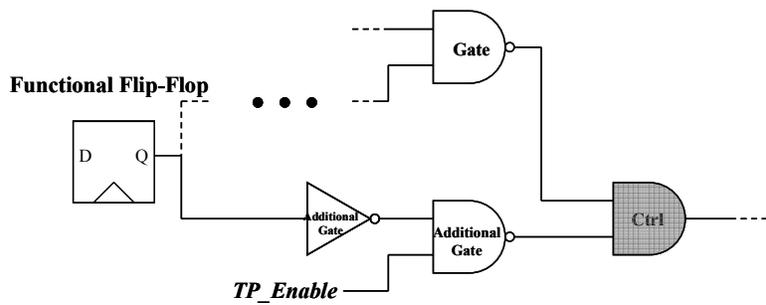
The purpose of the TP_Enable gate is that if a functional flip-flop is being used to activate a control point, it may affect the system function during normal operation. So the TP_Enable gate is needed to deactivate the control point at all times during normal operation. The TP_Enable gate can deactivate the control point by always driving one input of the control point gate with a non-controlling value ('1' for AND gate control point and '0' for OR gate control point). This modifies the CUT, and four types of control point structures are used in [Yang 09] to maximize the random pattern testability. Fig. 6.1 shows the new control point structures which include both control-0 points and control-1 points with both even and odd inversion parity through the TP_Enable gate.



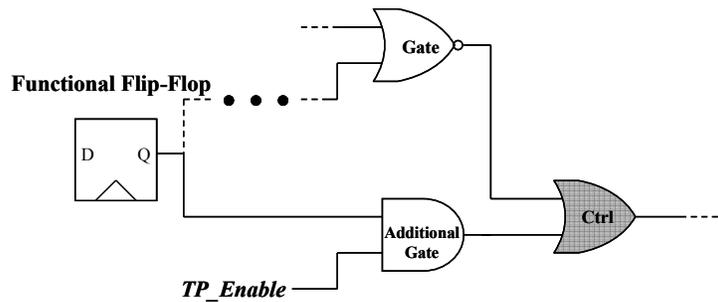
(a) Type 1 : Non-Inverting Functional Path and AND Ctrl



(b) Type 2 : Non-Inverting Functional Path and OR Ctrl



(c) Type 3 : Inverting Functional Path and AND Ctrl



(d) Type 4 : Inverting Functional Path and OR Ctrl

Figure 6.1 New Types of Control Point Structure for Different Path Inversion Parity

As mentioned, there are two paths existing from a functional flip-flop to a control point. Henceforth, in this chapter, the original functional path will be referred as the “*Functional Path*” and the new path created from a functional flip-flop to the control point will be referred as the “*TP_Driver Path*”. In a conventional control point structure, propagation is only enabled when the control point is driven by a dedicated flip-flop with a non-controlling value. However, the random testability is enhanced by the new control point structures in Fig. 6.1. Having opposite inversion parity between the two paths enables better propagation and helps to reduce the number of test patterns needed to achieve the random testability.

Dedicated flip-flops for the control points cannot be replaced in [Yang 09] when no functional flip-flop meet the two rules listed above. In particular, the opposite path

inversion parity rule can be one of the main limiting reasons why some dedicated flip-flops are not replaced. Also note that some test points have only a single functional flip-flop in its fan-in. This happens when the controllability of a certain node needs to be greater than 0.5. In this scenario, a dedicated flip-flop cannot be replaced. These not-replaced flip-flops could still have a large area impact for some designs.

In this chapter, a new algorithm is proposed to replace the not-replaced flip-flops in [Yang 09]. The proposed alternative selection algorithm reduces the area impact of test point insertion without introducing any additional delays.

6.3 ALTERNATIVE SELECTION ALGORITHM

This section describes the new algorithm for addressing the limitations mentioned in the previous section. Depending on the design, it may not be possible for some control points to find flip-flops that have functional paths to the control point that are either all even or all odd parity. Moreover, control points that are placed at nodes in AND and OR trees are more likely to have a skewed value on the functional path, either '0' or '1', so they have skewed controllability. The algorithm in [Yang 09] cannot replace the dedicated flip-flops in the above cases. The goal with the alternative selection algorithm described here is to relax the rules mentioned in Sec. 6.2, and be able to find more candidate functional flip-flops that can be used to replace the dedicated flip-flops to achieve the more area reduction.

A dedicated flip-flop for a control point with a single functional flip-flop in its fan-in cannot be replaced in [Yang 09]. Fig. 6.2 is an example of a small circuit with one control point. This circuit has flip-flops (denoted *A* to *F*) and combinational logic (denoted *G1* to *G6* and *Ctrl*). The control point highlighted in gray (*Ctrl*) is driven by two flip-flops (*A* and *B*) where *A* is a dedicated flip-flop for activating the control point.

Because there is only one functional flip-flop in the fan-in of the control point, the dedicated flip-flop *A* cannot be replaced by a functional flip-flop in [Yang 09].

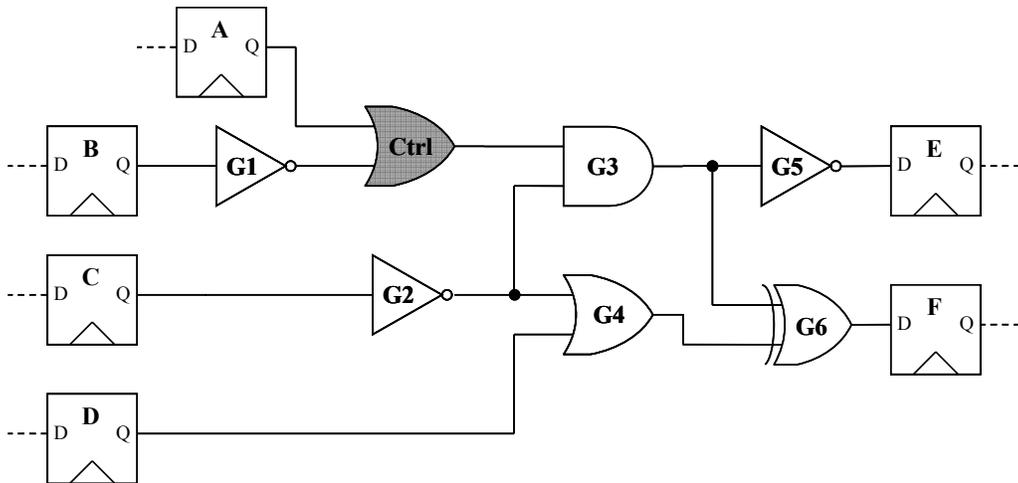


Figure 6.2 Example of a Circuit with One Control Point (*Ctrl*)

To replace the dedicated flip-flop in Fig. 6.2 where no functional flip-flops are available in the fan-in of test point, an alternative selection algorithm searches additional candidates from outside of test point's fan-in cone. Because the candidate flip-flops are searched outside of the fan-in of the control point, they need to be carefully chosen to avoid introducing new timing constraints.

Fig. 6.3 shows logic cones in a circuit. Through logic cone analysis starting from the test point, the fan-in cone of the test point can be determined. Functional flip-flops belonging to the fan-in (cross striped logic cone) are the candidates (*Current Set of Candidates*) for replacing a dedicated flip-flop. Since *Current Set of Candidates* belongs to a test point, functional flip-flops in *Current Set of Candidates* will be first considered as candidates. Some dedicated flip-flops may not satisfy the rules in Sec. 6.2 and are not replaced in [Yang 09]. If rules are relaxed and more candidates are found, there will be higher probability of having more number of dedicated flip-flops replaced.

Hence, as long as no redundant faults are introduced, a functional flip-flop which does not meet the rules can be considered for replacing a dedicated flip-flop.

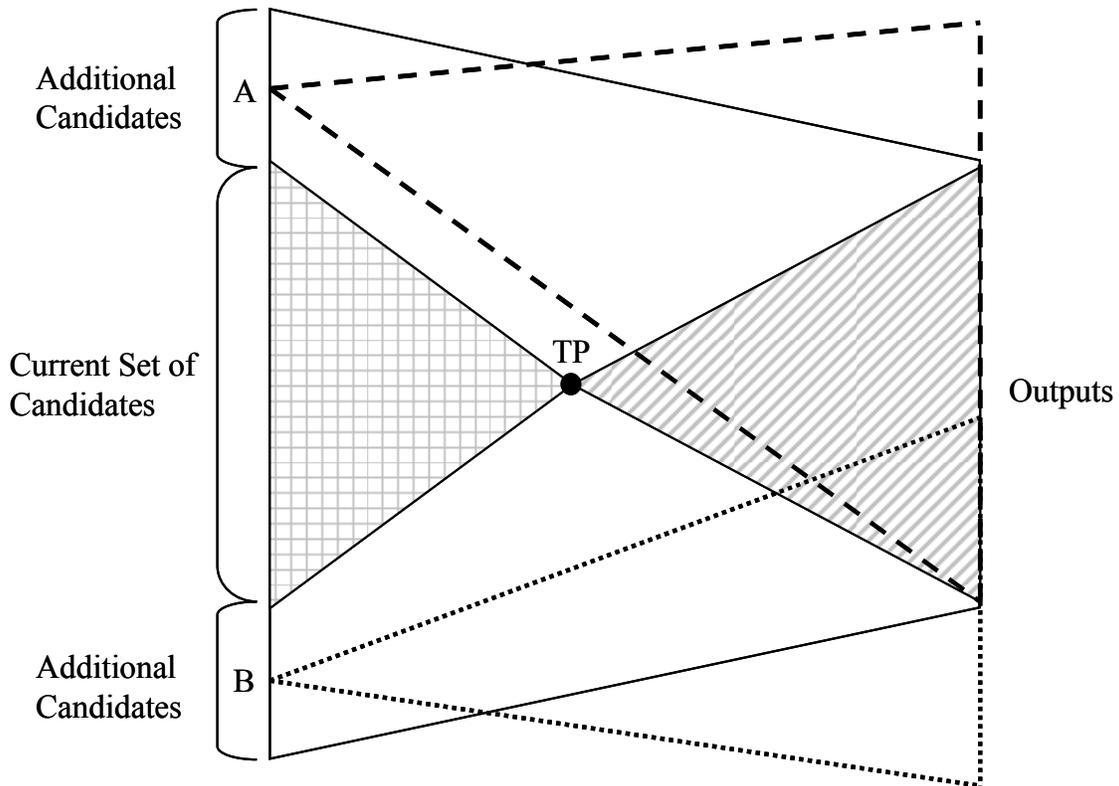


Figure 6.3 Alternative Selection Algorithm

Assume that there are no functional flip-flops available in the current set of candidates for performing the replacement. In this case, the alternative selection algorithm can be used to find possible functional flip-flops that can replace the dedicated flip-flop for the test point. The alternative selection algorithm finds a functional flip-flop that is not in the fan-in of the test point which may add additional timing constraints. Therefore, the selection algorithm needs to guarantee that there is no performance penalty by the proposed method.

The alternative selection algorithm starts from the test point. The nodes are traversed from the test point and the fan-out cone is generated. The fan-out cone is highlighted with dashed lines in Fig. 6.3. The proposed algorithm tries to find functional flip-flops that are related to the outputs in the test point logic cone. From each output of the fan-out cone, the logic cone analysis generates a fan-in cone. Once all the output fan-in cones are generated, it is possible to list all the inputs which are associated with outputs in the test point's fan-out cone. Some inputs are also found in the test point's fan-in cone while other inputs do not belong to the test point's fan-in cone. In Fig. 6.3, we can find a parallelogram which is the logic associated with the test point. *Current Set of Candidates* denotes the inputs of the test point's fan-in cone and *Additional Candidates* are the inputs which are not in the test point's fan-in cone, but found from the logic cone analysis from the outputs.

Once all the inputs which are related to the test point are found, a functional flip-flop needs to be selected to replace the dedicated flip-flop. To avoid introducing additional timing constraints, functional flip-flop candidates need to be considered in a logic parallelogram. In the example in Fig. 6.3, since only one functional flip-flop is found in the *Current Set of Candidates* and it is not replaceable, functional flip-flop candidates need to be selected from the set of *Additional Candidates*. There are two types of inputs in *Additional Candidates*. One is an input whose fan-out cone includes all the outputs of the test point's fan-out cone. The other type of input partially covers the outputs of the test point's fan-out cone. *A* in Fig. 6.3 has its logic cone drawn with a dashed line and it includes all the outputs of TP fan-out cone. The fan-out logic cone of *B* is marked with a dotted line, and some portion of the outputs belongs to it. Since candidate flip-flops are found outside of the test point's fan-in cone, the main concern is the additional timing constraints in the alternative selection algorithm. To guarantee no

performance penalty, it is necessary to find functional flip-flops that cover all the outputs of the test point's fan-out cone. Therefore, candidate B is not acceptable. Inputs that have all of the test point's fan-out cone's outputs as their outputs can be considered as candidates for the alternative selection and hence candidate A is acceptable. However, if there is no functional flip-flop found which covers all the outputs, a dedicated flip-flop cannot be replaced.

A functional flip-flop that is logically close to the control point is chosen to replace a dedicated flip-flop. Acceptable candidates' logic cones partly share the logic with the test point's fan-out cone. Instead of tracing back from the test point as in [Yang 09], for each acceptable candidate, propagation from the test point is performed until the overlapped gate element with the test point's fan-out logic cone is first found. To minimize the length of the newly created test path from the candidate flip-flop to the control point, the input which has the closest overlapped element is selected for replacing the not-replaced dedicated flip-flops that remain after using the method in [Yang 09].

In Fig. 6.2, flip-flops E and F are found in the test point's fan-out cone. First, back tracing from E generates a logic cone and inputs B and C are found. Secondly, F fan-in cone has B , C and D as its inputs. Since B directly belongs to the control point's logic cone, *Current Set of Candidates* is B . *Additional Candidates* are C and D . The fan-out cones of C and D are generated and need to be checked whether they cover all the outputs of the control point, E , and F . E is not included by fan-out cone of D and hence it is not guaranteed to avoid new timing constraints. However, C covers all outputs E and F , and $G3$ is the nearest overlapped element from the test point between C and the test point's fan-out cones. Therefore, C can be used to replace A . If there are multiple candidates found, the flip-flop which has the nearest overlapped element to the test point

is used. Fig. 6.4 shows a circuit when the dedicated flip-flop *A* for the control point in Fig. 6.2 has been replaced with functional flip-flop *C*.

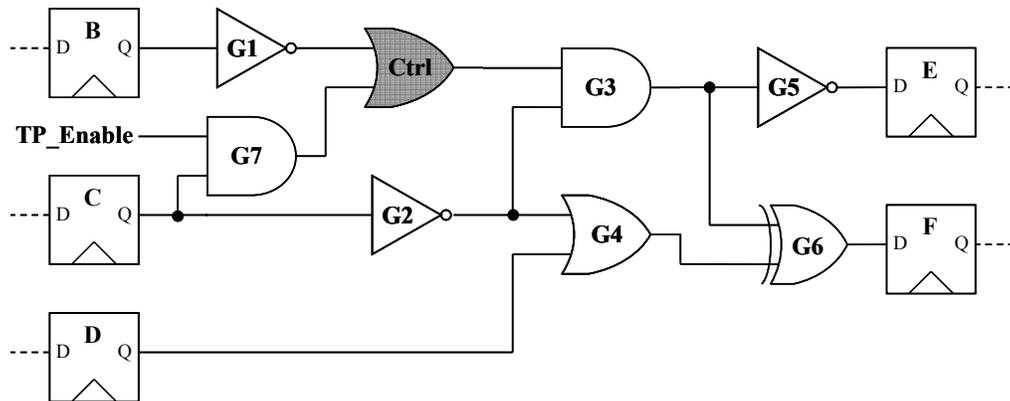


Figure 6.4 Dedicated Flip-Flop in Fig. 6.2 Replacement by a Functional Flip-Flop

6.4 EXPERIMENTAL RESULTS

In this section, experimental results are presented for industrial designs to evaluate the improvements that are obtained through the algorithm proposed here. The LogicVision testpointAnalyze tool [LogicVision] was used to determine the locations of test points in each design. Dedicated flip-flops for driving inserted control points are first replaced by the method in [Yang 09]. Then the proposed algorithm is used to try to increase the number of dedicated flip-flops that can be reduced without impacting delay and with minimizing the loss of testability.

Table 6.1 shows the number of dedicated flip-flops that are required for driving control points for each of the methods. The first column gives the design name. The number of observation points and control points inserted by the LogicVision tool [LogicVision] are shown in the second and third column. The sum of observation points and control points is the total number of test points inserted in a design. The fourth and fifth columns show the results in [Yang 09]. The number of dedicated flip-flops shows

the number of dedicated flip-flops for driving control points that could not be replaced and the number of functional flip-flops denotes the number of dedicated flip-flops that could be replaced. The last two columns show the results using the proposed method. The proposed method is applied on top of [Yang 09] and replaces dedicated flip-flops that are not replaced by [Yang 09].

Table 6.1 Dedicated Flip-Flop Replacement Comparisons

Design	Conventional TP Insertion		Replacement in [Yang 09]		Proposed Method	
	Observation Point	Control Point	Dedicated Flip-Flop	Functional Flip-Flop	Dedicated Flip-Flop	Functional Flip-Flop
Design A	70	179	39	140	2	177
Design B	129	371	13	358	8	363

Table 6.2 compares the ratios of dedicated flip-flop reduction and test point area reduction. The reduction ratio is computed as the number of functional flip-flops used to replace dedicated flip-flops divided by the number of total control points. For test point area reduction, the following equation is used:

$$\frac{New\ Area}{Old\ Area} = \frac{Nobs + Ndedicated + k * Nfunctional}{Nobs + Ndedicated + Nfunctional} = 1 - Area_reduction$$

As shown with 130nm TSMC technology in [Yang 09], each of the new control points sourced by a functional flip-flop takes approximately 1/4 of the area of the original control points driven with a dedicated flip-flop. Therefore, the following equation can be used to extrapolate the area reduction with 0.25 as a k factor.

Table 6.2 Improvement Comparisons

Design	Conventional TP Insertion		Replacement in [Yang 09]		Proposed Method	
	Reduction Ratio (%)	Test Point Area Reduction (%)	Reduction Ratio (%)	Test Point Area Reduction (%)	Reduction Ratio (%)	Test Point Area Reduction (%)
Design A	0	0	78.2 %	42.2 %	98.9 %	53.3 %
Design B	0	0	96.5 %	53.7 %	97.8 %	54.5 %

Table 6.3 shows a testability comparison with other test point insertion techniques (conventional method and a method in [Yang 09]). The fault coverage is shown when 100000 random patterns are applied. In Design A, most of the dedicated flip-flops are replaced, however, the testability that is achieved is a little lower than the conventional methods. Top-up patterns can be applied to achieve higher testability.

Table 6.3 Testability Comparisons

	Design Method	Design A	Design B
Fault Coverage	Conventional	98.71 %	98.25 %
	[Yang 09]	98.61 %	97.86 %
	Proposed	98.09 %	97.85 %

6.5 CONCLUSIONS

In this chapter, a new technique which helps to reduce the area impact of test point insertion is proposed. An alternative selection algorithm replaces the dedicated flip-flops with skewed controllability. Experimental results indicate that significant numbers of dedicated flip-flops are replaced by the proposed method and hence the area impact by a test point insertion can be noticeably reduced. Note that the proposed method can be incorporated with the existing test point insertion algorithms to minimize the area impact with minimizing the loss of random pattern testability.

Chapter 7: Conclusion and Future Work

7.1 CONCLUSION

As mentioned in Chapter 1, post-silicon debug has become a most time consuming challenge in the chip development cycle. This dissertation proposes several debug methodologies and test point insertion techniques to enhance the signal observability. Since design are very complicated, it is almost impossible to detect 100% of design bugs via pre-silicon verification techniques in the pre-silicon verification stage. And as process technology scales well below 100 nanometers, electrical bugs due to process variations increasingly become issues. The circuit behaviors could deviate from their specified functions in some operating regions. Circuit misbehaviors are caused by these bugs and the narrow signal observability hampers the post-silicon debug process. In this dissertation, new techniques to enhance signal observability are proposed.

In chapter 2, a new debug technique that utilizes the scan chains is presented. Unlike conventional scan-based methods, the proposed method avoids halting the system to read the internal system states. The proposed three step process can zero in on the first clock cycle in which an error is present with a small number of scan dumps. Non-destructive scan chains are reconfigured as multiple MISRs and they provide the internal system state information without halting the system operation. In chapter 3, two dimensional compaction method was proposed to increase the volume of meaningful debug data by selectively storing the data. This helps to improve the utilization efficiency of the trace buffer and enhances the signal observability by orders of magnitude as well. Because there are limited DFD hardware resources available, it is very important to determine the signals to observe for silicon debug. Automated selection of signals to observe is presented. Debugging capability can be maximized by

observing the right signals and the fault simulation and integer linear programming (ILP) guides the signal selection to enhance the observability. The investigation on test point insertion further extends research not only on signal observability but also signal controllability. In chapter 5, a new test point insertion method is proposed. Functional flip-flops are used to replace dedicated flip-flops for control points so that the area overhead is significantly reduced. The technique in chapter 6 replaces more dedicated flip-flops which were not replaced by the method in chapter 5. The methods in chapter 5, 6 replace the significant number of dedicated flip-flops for control points and achieve the area reduction without losing the testability.

7.2 FUTURE WORK

Some possible research directions can be identified for the post-silicon debugging techniques proposed in this dissertation.

One of the main challenges in the post-silicon debug is *In-System Silicon Debug*. In SOC (System On a Chip), there are multiple cores and they interact with each other when a program is executed on the system. This requires acquiring the debug data from interrelated cores. And when debugging a chip at the system level, the interactions among interrelated modules are invoked by event based transactions not by cycle based operations. In these *In-System Silicon debug*, the error manifests with non-deterministic behavior so that it may not be reproducible. One reason of the non-determinism is asynchronous communications among modules via buses. Hence, it is very difficult to narrow down the root cause of errors in time and location in the design. The proposed methods in this dissertation primarily focus on enhancing the signal observability by storing when the error is invoked in a system operation. The triggering logic which initiates the debugging data acquisition process helps to obtain real time information. Therefore, the future work involves an investigation of the triggering unit that signals to

start capturing the debug data. This will help to localize the debugging information in time space. Debugging data needs to be extracted from the modules so as to reduce the error location space. Distributed trace buffers can be used to record the histories of the executed programs. Debugging capability can be maximized by run time reconfigurable / programmable triggering logic such as assertion based trigger. Therefore, system level debugging technique using trigger logic can be further investigated for future work.

In test point insertion, more area reduction can be achieved via investigation of circuit structures for which no suitable functional flip-flops could be replaced. And the future work also involves an observation point implementation which reduces the BIST area.

Bibliography

- [Abramovici 05] Abramovici, M., and Y.-C. Hsu, "A New Approach to Silicon Debug," *Proc. of Int. Silicon Debug and Diagnosis Workshop (SDD)*, 2005.
- [Abramovici 06] Abramovici, M., P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A Reconfigurable Design-for-Debug Infrastructure for SoCs," *Proc. of Design Automation Conference*, pp. 7-12, 2006.
- [Anis 07a] Anis, E., and N. Nicolici, "On Using Lossless Compression of Debug Data in Embedded Logic Analysis," *Proc. International Test Conference*, pp. 1-10, 2007.
- [Anis 07b] Anis, E., and N. Nicolici, "Low Cost Debug Architecture using Lossy Compression for Silicon Debug," *Proc. of Design, Automation, and Test in Europe*, pp. 1-6, 2007.
- [Barnhart 01] Barnhart, C., V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "OPMISR: the Foundation for Compressed ATPG Vectors," *Proc. of International Test Conference*, pp. 748-757, 2001
- [Bershteyn 93] Bershteyn, M., "Calculation of multiple sets of weights for weighted random testing," *Proc. of International Test Conference*, pp. 1031-1040, 1993.
- [Briers 86] Briers, A. J., and K.A.E. Totton, "Random Pattern Testability by Fast Fault Simulation," *Proc. of International Test Conference*, pp. 274-281, 1986.
- [Brglez 89] Brglez, F., D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. of International Symposium on Circuits and Systems*, pp. 1929-1934, 1989.
- [Carbine 97] Carbine, A. and Feltham, D., "Pentium®Pro Processor Design for Test And Debug," *Proc. of International Test Conference*, pp. 294-303, 1997.
- [Chatterjee 95] Chatterjee, M., and D.K. Pardhan, "A Novel Pattern Generator for Near-Perfect Fault-Coverage," *Proc. of VLSI Test Symposium*, pp. 417-425, 1995.
- [Cheng 95] Cheng, K.-T., and C.J. Lin, "Timing –Driven Test Point Insertion for Full-Scan and Partial-Scan BIST," *Proc. of International Test Conference*, pp. 506-514, 1995.
- [Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self Test," *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.
- [GLPK] <http://www.gnu.org/software/glpk/glpk.html>

- [Gu 06] Gu, X., Wang, W., Li, K., Kim, H. and Chung, S., "Re-Using DFT Logic for Functional and Silicon Debugging Test", *Proc. of International Test Conference*, pp. 648-656, 2006.
- [Hammond 89] Hammond, B.J., "Development in Logic Analyzers," *Proc. of IEE Colloquium on Instrumentation in Electronic Product Manufacture*, pp. 2/1- 2/3, 1989.
- [Hammond 90] Hammond, B.J., "Testing Time-Advances in Logic Analysis," *IEE Review*, 36(3)103-106, 1990.
- [Hellebrand 92] Hellebrand, S., S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Register," *Proc. of International Test Conference*, pp. 120-129, 1992.
- [Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Trans. on Computers*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.
- [Hopkins 06] Hopkins, A., and K. McDonald-Maier, "Debug Support for Complex Systems on-Chip: A Review," *IEEE Proc. on Computers and Digital Techniques*, Vol 153, No. 4, pp. 197-207, Jul. 2006.
- [Hsu 06] Hsu, Y.-C., F. Tsai, W. Jong and Y.-T. Chang, "Visibility Enhancement for Silicon Debug," *Proc. of Design Automation Conference*, pp. 13-18, 2006.
- [ITRS 05] "The International Technology Roadmap for Semiconductors," Semiconductor Industry Association, 2005.
- [ITRS 07] "The International Technology Roadmap for Semiconductors," Semiconductor Industry Association, 2007.
- [Iyengar 89] Iyengar, V.S., and D. Brand, "Synthesis of Pseudo-Random Pattern Testable Designs," *Proc. of International Test Conference*, pp. 501-508, 1989.
- [Jang 08] Jang, W., Ding, D. and Pan., D., "A Voltage-Frequency Island Aware Energy Optimization Framework for Networks-on-Chip", *Proc. IEEE/ACM Int. Conference on Computer-Aided Design*, 2008.
- [Jas 01] Jas, A., and C.V. Krishna, and N.A. Touba, "Hybrid BIST based on Weighted Pseudo-Random Testing: A New Test Resource Partitioning," *Proc. of VLSI Test Symposium*, pp. 2-8, 2001.

- [Josephson 04] Josephson, D. and Gottlieb, B., "The Crazy Mixed up World of Silicon Debug," *Proc. of IEEE Custom Integrated Circuits Conference*, pp. 665-670, 2004
- [Kapur 94] Kapur, R., S. Patil, T.J. Snethen, and T.W. Williams, "Design of an efficient weighted random pattern generation system," *Proc. of International Test Conference*, pp. 491-500, 1994.
- [Ko 08] Ko, H. F., and Nicolici, N., "Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation", *Proc. of Design, Automation, and Test in Europe*, pp. 1298-1303, 2008.
- [Konemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. of European Test Conference*, pp. 237-242, 1991.
- [Konemann 01] Koenemann, B., C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheeler, "A SmartBIST variant with guaranteed encoding," *Proc. of VLSI Test Symposium*, pp. 325-330, 2001.
- [Krishna 01] Krishna, C.V., A. Jas, and N.A. Touba, "Test Vector Encoding Using Partial LFSR Reseeding," *Proc. of International Test Conference*, pp. 885 – 893, 2001.
- [Krishnamurthy 87] Krishnamurthy, B., "A Dynamic Programming Approach to the Test Point Insertion Problem," *Proc. of Design Automation Conference*, pp. 695-704, 1987.
- [Kuppuswamy 04] Kuppuswamy, R., DesRosier, P., Fletham, D., Sheikh, R. and Thadikaran, P., "Full Hold-Scan Systems in Microprocessors : Cost/Benefit Analysis," *Intel Technology Journal*, Volume 08, Issue 01, 2004.
- [Lai 05] Lai, L., J.H. Patel, T. Rinderknecht, and W.-T. Cheng, "Hardware efficient LBIST with complementary weights," *Proc. of International Conference on Computer Design*, pp. 479-481, 2005.
- [LogicVision] ETAnalysis Tools Reference Manual, software version 6.0a, 2007.
- [Nakao 99] Nakao, M., S. Kobayashi, K. Hatayama, and K. Iijima, "Low Overhead Test Point Insertion for Scan-Based BIST," *Proc. of International Test Conference*, pp. 384-357, 1999 Scan-Based BIST,"
- [OR1200] OPENCORES, <http://www.opencores.org>
- [Park 08] Park, S.-B., and Mitra, S., "IFRA: Instruction Footprint Recording and Analysis for Post-Silicon Bug Localization in Processors", *Proc. of Design Automation Conf.*, pp. 373-378, 2008.

- [Pomeranz 92] Pomeranz, I., and S.M. Reddy, "3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 7, pp. 1050-1058, Jul. 1993.
- [Rajski 02] Rajski, J., J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, T. Kun-Han, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eider, and Q. Jun, "Embedded deterministic test for low cost manufacturing test," *Proc. of International Test Conference*, pp. 301-310, 2002.
- [Seiss 91] Seiss, B.H., P. Trouborst, and M. Schulz, "Test Point Insertion for Scan-Based BIST," *Proc. of European Test Conference*, pp. 253-262, 1991.
- [Shen 99] Shen J., and Abraham, J. A., "Verification of Processor Microarchitectures," *Proc. of VLSI Test Symposium*, pp. 189-194, Apr. 1999.
- [Tamarapalli 96] Tamarapalli, N., and J. Rajski "Constructive Multi-Phase Test Point Insertion for Scan-Based BIST," *Proc. of International Test Conference*, pp. 649-658, 1996.
- [Touba 95a] Touba, N. A., and E. J. McCluskey, "Transformed Pseudo-Random Patterns for BIST," *Proc. of VLSI Test Symposium*, pp. 410-416, 1995.
- [Touba 95b] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. International Test Conference*, pp. 674-682, 1995.
- [Touba 96] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," *Proc. of VLSI Test Symposium*, pp. 2-8, 1996.
- [Vermeulen 01] Vermeulen, B., Oostdijk, S. and Bouwman, F., "Test and Debug Strategy of the PNX8525 Nexperia™ Digital Video Platform System Chip", *Proc. of Int. Test Conference*, pp. 121-130, 2001.
- [Vermeulen 02] Vermeulen, B., Waayers, T. and Goel, S.K., "Core-Based Scan Architecture for Silicon Debug", *Proc. International Test Conference*, pp. 638-647, 2002.
- [Wunderlich 96] Wunderlich, H.-J., and G. Kiefer, "Bit-Flipping BIST," *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pp. 337-343, 1996.
- [Yang 08a] Yang, J.-S., and Touba, N. A., "Expanding Trace Buffer Observation Window for In-System Silicon Debug through Selective Capture", *Proc. of IEEE VLSI Test Symposium*, pp. 345-351, 2008.

- [Yang 08b] Yang, J.-S., and Touba, N. A., “Enhancing Silicon Debug via Periodic Monitoring”, *Proc. of IEEE Symposium on Defect and Fault Tolerance*, pp. 125-133, 2008.
- [Yang 09] Yang, J.-S., B. Nadeau-Dostie, and N.A. Touba, “Test Point Insertion Using Functional Flip-Flops to Drive Control Points,” *Proc. of International Test Conference*, 2009.
- [Yen 06] Yen, C-C., Lin, T., Lin, H., Yang, K., Liu, T. and Hsu, Y.-C., “Diagnosing Silicon Failures Based on Functional Test Patterns”, *International Workshop on Microprocessor Test and Verification*, pp. 94-98. 2006.
- [Youssef 93] Youssef, M., Y. Savaria, and B. Kaminska, “Methodology for Efficiently Inserting and Condensing Test Points,” *IEEE Proceedings Computers and Digital Techniques*, Vol. 140, pp. 145-160, May, 1993.

Vita

Joon-Sung Yang was born and raised in Seoul, Korea. He graduated Sehwa High School in 1996 and earned his Bachelors in 2003 from YONSEI University, Seoul, Korea, where he majored in Electrical Engineering. He worked in Samsung Electronics from 2003 to 2005. He joined the University of Texas at Austin in 2005, where he did his MS in Computer Engineering with a thesis on VLSI Testing in 2007. He received scholarship from Korea Science and Engineering Foundation and best paper award at the International Symposium on Defect and Fault Tolerance in 2008. He is currently pursuing his Ph. D. in Computer Engineering with a focus on post-silicon techniques debug by hardware insertion. His primary areas of research interests include DFD (Design for Debug), DFT (Design for Testability) and test data compression.

Permanent Address : 202 Hanavilla 48 Bundangdong
Bundanggu, Sungnamsi
Kyunggi, Korea (463-831)

This dissertation was typed by Joon-Sung Yang.