On Multiplexed Signal Tracing for Post-Silicon Debug

Xiao Liu and Qiang Xu
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {xliu,qxu}@cse.cuhk.edu.hk

ABSTRACT

Trace-based debug solutions facilitate to eliminate design errors escaped from pre-silicon verification and have gained wide acceptance in the industry. Existing techniques typically trace the same set of signals throughout each debug run, which is not quite effective for catching design errors. In this work, we propose a multiplexed signal tracing strategy that is able to significantly increase debuggability of the circuit. That is, we divide the tracing procedure in each debug run into a few periods and trace different sets of signals in each period. A novel trace signal grouping algorithm is presented to maximize the probability of catching the propagated evidences from design errors, considering the trace interconnection fabric design constraints. Experimental results on benchmark circuits demonstrate the effectiveness of proposed solution.

1. INTRODUCTION

Today's complex integrated circuit designs increasingly rely on post-silicon validation to eliminate bugs that escape from pre-silicon verification [2, 6, 8]. One effective post-silicon debug technique is to monitor and trace the behavior of the circuit under debug (CUD) during its normal operation. As shown in [1], for million-gate industrial designs with trace-based debug support, it is common to tap thousands of "key" signals in the circuit and select a subset of them (say, 32 signals) to trace concurrently in each debug run[1] [9, 11]. An interconnection fabric is used to link the large number of tapped signals to the trace buffers and/or trace ports with limited bandwidth [10].

Existing trace-based solutions typically trace the same set of selected signals throughout each debug run. It might not be a problem for applying trace-based solution on dedicated purposes, e.g., to reconstruct instruction flow [12] or to monitor bus communication protocol [3]. When debugging general logic circuits, however, this kind of "static" signal tracing methods limits the visibility of the design since it only provides limited part-view of the CUD, while the states of other tapped signals are not visible. Consequently, if the error effects do not manifest themselves on those signals that are currently under trace, more debug runs need to be conducted, which greatly increases the bug localization effort. Some recent works try to address this problem by expanding the visibility from traced information to other signals based on the concept of "state restoration" [9, 11]. While interesting, the effectiveness of these methods for identifying design errors is limited since errors may actually occur only on those expanded signals, which are different from their restored values.

Intuitively, for a permanent design error that has been activated, if the trigger mechanism is set properly and its effect can be propagated to one or more tapped signal, there should be a high possibility to catch it when the number of trace cycles is sufficiently long and further increase of trace cycles does not improve the debuggability much. Based on the above, in this work, we propose a multiplexed signal tracing strategy that is able to significantly improve the debuggability of the circuit, by leveraging the fact that the number of tapped signals is much more than the number of signals that can be traced concurrently. That is, we divide the tracing procedure in each debug run into a few periods and intelligently trace a different subset of accessible signals (i.e., tapped signals) in each period.

With multiplexed tracing, the possibility of catching the evidence of design errors can be dramatically increased since we now have a better view of the overall CUD by observing more signals in each debug run. How to selectively group the tapped signals for tracing in each period, however, becomes a critical question in this strategy. To tackle this problem, we define a new evaluation metric namely *design error visibility* (*DEV*) to measure the tracing quality and use it as a guidance in our proposed heuristic for trace signal grouping. With random design errors injected in benchmark circuits, experimental results show that our solution is much more effective than existing "static" tracing techniques and multiplexed tracing with randomly-grouped trace signals, and the hardware cost introduced by the proposed multiplexed tracing scheme is nearly negligible.

The remainder of this paper is organized as follows. Section 2 reviews related works and presents the motivation of our work. In Section 3, we define the proposed metric used to evaluate the effectiveness for capturing the evidence of design error. In Section 4, we describe the proposed methodology in detail. Experimental results on benchmark circuits are shown in Section 5. Finally, Section 6 concludes this work.

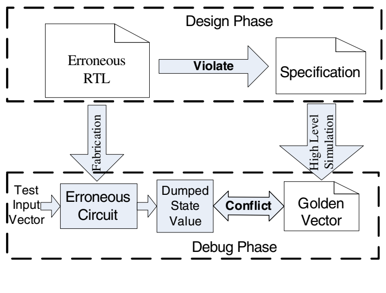


Figure 1: Post-Silicon Debug for Design Errors.

2. PRELIMINARIES AND MOTIVATION

Design errors are caused by erroneous design process and they violate the pre-defined specification of IC products. Due to the ever-increasing complexity of IC designs, there is a growing number of such errors left in the first silicon, requiring post-silicon debug to catch them [14]. As shown in Fig. 1, during debug phase, certain test input vectors fed into the fabricated chip can activate the bug so that its error effects manifest themselves. Since the circuit under debug is a piece of silicon that has already been fabricated, the main challenge here is that bugs may take long time to be activated and there is only limited visibility of its internal signals. Consequently, an essential step is to capture the error evidences within the chip, so that designers can quickly focus on a small region of the CUD, and then apply various diagnosis methods (e.g., [5]) to root-cause the bug and fix it.

One widely-used technique to mitigate this problem is to reuse the CUD's existing test structure (e.g., scan chains) to run/stop its operation and observe whether the values in the circuit's storage elements

[1]It is impossible to trace all the tapped signals at the same time due to the limited trace bandwidth.

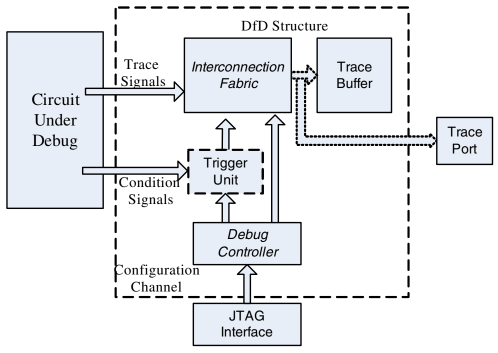
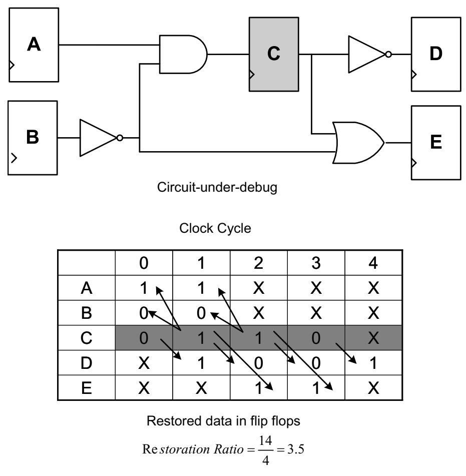


Figure 2: Trace-Based Debug Infrastructure.

are the expected values [8, 15]. However, this low-cost technique provides little help for tracking those tricky bugs that takes a long period of operation to manifest themselves. In addition, it is a challenging problem to repeat the CUD's error occurrence procedure in a deterministic cycle-accurate manner.

To tackle the above problems, trace-based post-silicon debug solutions are proposed to observe the states of selected internal signals in the CUD at real-time. Fig. 2 depicts a conceptual hardware infrastructure for trace-based debug techniques. As signal tracing involves non-trivial design-for-debug (DfD) overhead, only a small portion of "key" signals in the circuit are tapped, and in each debug run, a subset of the tapped signals are traced concurrently. An interconnection fabric is used to link the large number of tapped signals to the trace buffers/ports. The trigger unit controls the start and stop of the tracing, in which the triggering mechanism can be configured through JTAG interface through the debug configuration channel [16].



	0	1	2	3	4
A	1	1	X	X	X
B	0	0	X	X	X
C	0	1	1	0	X
D	X	1	0	0	1
E	X	X	1	1	X

$$Restoration\ Ratio = \frac{14}{4} = 3.5$$

Figure 3: State Restoration Example.

Selecting which signals to tap in the design is a key issue for effective debug. To maximize the visibility of the CUD, some researchers proposed to select signals based on the "state restoration" concept [9, 11, 13]. An example is shown in Fig. 3. Since some states can be restored with logic implication from traced FF(C), 14 states are restored from 4 traced ones. Using such restoration capability as evaluation metrics to guide trace signal selection, however, may miss finding certain bugs. Consider the example in Fig. 3 again. Suppose the OR gate is mistakenly replaced by another gate (due to some design errors), it is only beneficial if we trace FF(E) to capture the evidence induced by this error. We cannot identify this error by tracing other FFs (e.g., FF(C)) and applying *golden* netlist-based logic implication to root cause it, even if the corresponding restoration ratio is high. In [17], the authors proposed a different trace signal selection method, targeting on minimizing the latency for capturing the propagated evidences after bugs are triggered. The method, however, is based on the assumption that the states' sequence for exposing error is available, which may not be obtained as early as the design phase. Furthermore, another key issue of debuggability on how to determine a subset of

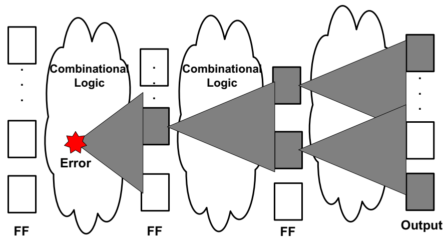


Figure 4: Example of Design Error Evidence Propagation.

accessible signals to trace in the debugging phase remains untouched.

Tricky design errors may take millions of cycles to be activated during debug runs [4], when the CUD is accidentally operated into "corner cases". If the trigger mechanism is properly designed, however, the error can generate its evidences from time to time during the running procedure. In addition, as demonstrated in Fig. 4, the evidences will further propagate forwardly and leave their tracks on FFs (e.g., the FFs in bold in Fig. 4). This evidences can also be masked by the controlling values of side-inputs in the process of going through a logic gate. As a result, some FFs cannot capture evidences (e.g., the FFs in blank in Fig. 4). With this observation, if we are able to observe the error evidences by properly tracing relevant FFs for some time, the region with error is greatly zoomed in, and hence the root-cause effort is significantly reduced. Besides, the tracing time is not necessary to be as many as possible. In [13], the authors proposed to select and trace two sets of signals in odd time-frames and even time-frames respectively. Such multiplexed tracing method, however, requires interconnection fabric to redirect debug data flow in every clock cycle, which involves large DfD overhead and high power consumption.

In trace-based debug solutions, the number of accessible signals is much more than the number of signals that can be traced concurrently in each cycle. Hence, for a suspicious region that is likely containing errors, it is possible to divide the tracing procedure into different periods and intelligently select a group of "key" signals from accessible ones for tracing in each period. This strategy can enhance the capability of capturing error evidences for the region. Motivated by above, we propose the multiplexed signal tracing method as follows.

Total Design Error Visibility (TDEV), which indicates the overall capability of detecting design errors in the suspicious region

3. DESIGN ERROR VISIBILITY METRIC

As discussed in Section 2, it is essential to introduce a reasonable metric to evaluate the effectiveness on capturing the evidences from possible design errors within a suspicious region. We start with emulating the actual behavior of error evidence propagation, as demonstrated in Fig. 4. To be specific, when a design error is activated in functional mode, the phenomenon that a correct '0' is replaced by an erroneous '1' or a correct '1' is replaced by an erroneous '0' occurs on at least one node of the circuit. Then the erroneous value propagates in the circuit and might be captured by FFs or outputs. Based on this observation, we first introduce *Evidence Impact* (denoted by $EI1/EI0$) to represent the probability that the evidence is propagated to a certain FF (see Table 1). Different from similar concept of fault detection probability in manufacturing test [7], for design error we cannot predict the error occurrence probability, hence Evidence Impact is set to '1' at the possible root-cause location. With the forward propagation, the evidence tends to be masked because some other side-inputs can be controlling values. To indicate this weakening effect, we then introduce a series of *weakening parameters (WP)* for various types of gates and express them as the following equations.

$$WP_{\text{and/nand}} = \prod_{i=1}^{n_{\text{input}}-1} P_i(1) \tag{1}$$

$$WP_{\text{or/nor}} = \prod_{i=1}^{n_{\text{input}}-1} P_i(0) \tag{2}$$

$$WP_{\text{not/xor/xnor}} = 1 \tag{3}$$

Selected Visibility ($SV1/SV0$)	For selected trace nodes, $SV1 = SV0 = 1$; otherwise $SV1 = SV0 = 0$
Design Probability ($P1/P0$)	The probability that the node is '1'/'0' in functional mode
Evidence Impact ($EI1/EI0$)	The probability that evidence '1'/'0' is propagated on one FF
Evidence Visibility ($EV1/EV0$)	The probability that the evidence '1'/'0' is visible to be captured on one traced FF
Design Error Visibility ($DEV1/DEV0$)	The probability that the design error with evidence '1'/'0' is visible with traced signals
Total Design Error Visibility ($TDEV$)	The capability that design errors in the suspicious region are visible with traced signals

Table 1: Terminologies for Visibility Calculation.

where, n_{input} is the number of inputs to the gate and $P_i(0/1)$ is the probability of the logic value (0/1) on side-input i of the gate. Several methods can be utilized to obtain the above probability. One method is to run simulation with operational input sequence and then dump internal values to estimate it. Alternatively, we can simply use structural analysis to calculate the probability forwardly from primary inputs, by assuming some control inputs are pre-set as 0/1 to ensure the CUD is working in functional mode, while all other inputs are with the probability 0.5 to be value 0/1.

With these notations, after the evidence passes through a gate, it is weakened as $EI_{out} = EI_{in} \times WP$. Re-convergent fan-out may cause multiple propagated evidences to propagate through the same gate. To capture this effect, we introduce the expression $EI_{out} = \bigcup EI_{in_{evi.}} \times WP_{non-evi.}$, within which the $WP_{non-evi.}$ is determined by the inputs without propagated evidences.

If there is a chance that the evidence is captured by a certain FF, we need to trace this FF such that the event that the evidence reaches the FF can be observed. We therefore need to define the selected visibility (SV) to represent the monitor capability of the traced signals (the detailed definition are shown in Table 1).

Sequentially, we use Evidence Visibility (EV) to indicate the probability that the evidence is visible to be captured on a under-traced FF, which is given by

$$EV1 = SV1 \times EI1 \tag{4}$$

$$EV0 = SV0 \times EI0 \tag{5}$$

Since the evidences generated from an error can be captured by one or more traced FFs and the observation of any evidence is helpful for detecting the error, we regard the events that each evidence is captured by traced signals as mutually independent. With this assumption, the probability that a possible design error with evidence '1'/'0' is detected with traced signals can be expressed as

$$DEV1 = \bigcup_{k=1}^{n_{\text{FFcap}}} EV_k 1 \tag{6}$$

$$DEV0 = \bigcup_{k=1}^{n_{\text{FFcap}}} EV_k 0 \tag{7}$$

where n_{FFcap} is the number of FFs that capture error evidences.

Moreover, in the context of multiplexed tracing, we have several tracing periods and the evidences' capture in any period is helpful for detecting the corresponding error. To capture this effect, we can rewrite Eq. (8)-(9) as follows.

$$DEV1 = \bigcup_{j=1}^{m_{\text{period}}} \bigcup_{k=1}^{n_{\text{FFcap}}} EV_k 1 \tag{8}$$

$$DEV0 = \bigcup_{j=1}^{m_{\text{period}}} \bigcup_{k=1}^{n_{\text{FFcap}}} EV_k 0 \tag{9}$$

where m_{period} is the number of tracing periods. Again, the evidences' captures between different period are regarded as independent events.

For a possible design error, we can use the summation of its design error visibilities with respect to evidence '1' and '0' to evaluate the detection capability for the error. The summation of this quantity over all possible errors on suspicious cells is finally defined as Total Design Error Visibility (TDEV), which indicates the overall capability of detecting design errors in the suspicious region, and can be expressed as

$$TDEV = \sum_{i=1}^{n_{\text{cell}}} (DEV0 + DEV1) \tag{10}$$

where n_{cell} is the number of cells where possible errors can occur (namely, suspicious cells) in the suspicious region.

By emulating evidence propagation behavior, this metric provides a reasonable estimation of the propagation impact for each possible error, which facilitates to identify those FFs with high chances to detect errors. At the same time, by combining the visibilities of all suspicious cells (see Eq. (8)-(11)), we inherently balance the error detection capability for all suspicious cells in signal grouping. To be specific, every time we choose a new trace signal, the benefits of potential trace signals are evaluated by the induced gain on $TDEV$. Thus, generally speaking, the visibilities of the possible errors that already have high DEV tends to result in less $TDEV$ gain with new tracing signal; while the errors with low DEV tend to provide more $TDEV$ enhancement. Consider a simple case as an example. Given two suspicious cells a and b, at the end of the first period we have $DEV1(a) = 0.9$, $DEV1(b) = 0.1$, $DEV0(a) = 0$, and $DEV0(b) = 0$. In the second period, suppose we have two options to get 0.5 $DEV1$ for a and b for this period. Combining the detection capability contributed by the two periods with Eq. (10), the resulting $DEV1(a) = 0.95$ and $DEV1(b) = 0.55$. With Eq. (12), we will select those trace signals that enhance the error detection capability of suspicious cell b because they bring more $TDEV$ gain. From the above, the metric $TDEV$ inherently guides the selection of trace signals to cover possible errors with low detection capability during signal grouping procedure.

4. PROPOSED METHODOLOGY

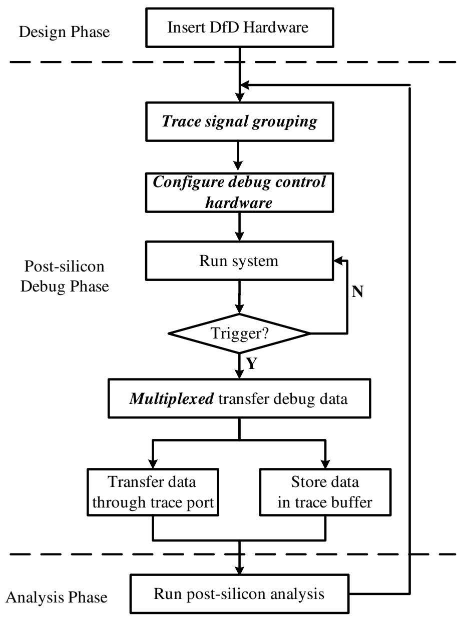
The design flow of the proposed multiplexed signal tracing method is described in Fig. 5. During design phase, supporting DfD circuitries to facilitate multiplexed tracing (e.g., interconnection fabric and debug controller as shown in Fig. 2) are inserted in the design (detailed in Section 4.1). Then, during the post-silicon debug phase, for a particular suspicious region relevant to one or more trigger conditions, we use an off-chip algorithm to determine signal grouping for maximizing error detection capability in that region (detailed in Section 4.2). The arrangement is loaded into on-chip debug controller through JTAG interface to facilitate multiplexed trace control.

When certain pre-defined condition is triggered in a debug run, the debug controller starts to trace data in a multiplexed fashion. The dumped information are then analyzed off-line by designers to root cause design errors. The debug process is terminated if we can successfully find the bugs. Otherwise, we either try to zoom in the suspicious region with the help of captured evidences or switch to the other part of the CUD when no error evidence is detected during the previous debug run. In both cases, the signal grouping algorithm is used again to obtain the new set of traced signals and the associated configuration is loaded into debug controller for another debug run. The above process is conducted iteratively until all design errors are found and eliminated.

4.1 Supporting DfD Hardware for Multiplexed Signal Tracing

To facilitate multiplexed signal tracing that are composed of multiple tracing periods, a few modifications are required on top of the conventional trace-based DfD infrastructure, as indicated in Fig. 6.

First of all, *shadow* registers are added into the configuration unit, which determines which signals are transferred through interconnection fabric. Within each tracing period, the shadow latches in these registers can be loaded with the configuration data for next tracing period without intervening the normal trace data transfer. Then, when the new period starts, all the *shadow* registers are updated by a global enable signal from debug controller to configure the trace interconnection fabric simultaneously so as to transfer data from another group



Design Phase
Insert DfD Hardware
Trace signal grouping
Configure debug control hardware
Post-silicon Debug Phase
Run system
N
Trigger?
Y
Multiplexed transfer debug data
Transfer data through trace port
Store data in trace buffer
Analysis Phase
Run post-silicon analysis

Figure 5: Proposed Trace-Based Debug Scenario.

of accessible signals to trace buffers/ports. It is important to note that, the time required to load the shadow register determines the minimum number of cycles of each tracing period.

In addition, a small RAM needs to be introduced into the on-chip debug controller to store the configuration data and a timer that controls the data loading into *shadow* registers and assert the global enable signal at the beginning of each tracing period. The controller can be configured through JTAG interface.

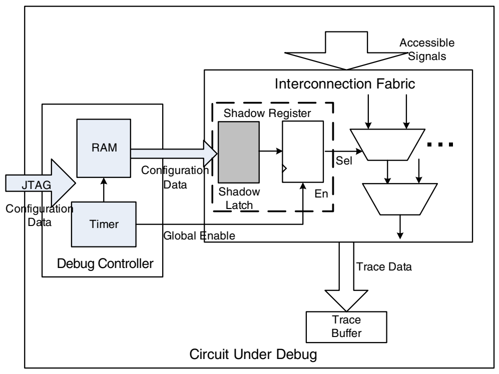


Figure 6: Diagram of Supporting DfD Hardware.

4.2 Signal Grouping Algorithm

As not all signals can be traced concurrently due to the existence of the trace interconnection fabric (e.g., any signals going through the same multiplexer cannot be traced concurrently), it is essential to develop an algorithm to judiciously group signals in each tracing period to maximize the error detection capability.

Based on the metric introduced in Section 3, the above problem is formulated as: Given

- Suspicious region under debug;
- Concurrent tracing constraint from interconnection fabric;
- Relevant accessible signals[2] determined by designers;
- Trace buffer size;
- The number of tracing periods;

To maximize TDEV (i.e., the total design error visibility) of the circuit under debug.

We tackle this problem in two steps. Firstly, with a set of targeted cells in suspicous region, we extract the relevant FFs that have the potential to catch evidences and estimate corresponding evidence impact by following their evidences' propagation tracks. In this step, For each suspicious cell, we initialize $EI0/EI1$ to be 1, which is for error evidence 1 and 0, respectively. We then propagate them forwardly. The impact is weaken during propagation so that the process will be terminated when the impact is close to 0. It can also be stopped by bounding the number of relevant FFs to a pre-defined threshold due to the memory cost, as capturing the propagated error evidences on a few nearby FFs is sufficient to detect the root-cause error. After that, we propose a heuristic signal grouping algorithm to maximize TDEV for the general type of interconnection fabric as follows.

[2]Not all accessible signals are relevant for each debug run.

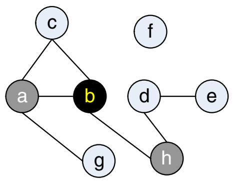
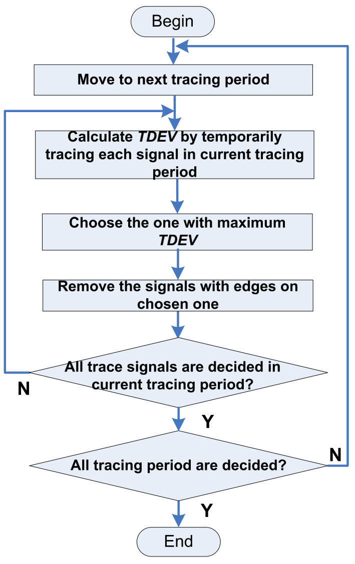


Figure 7: General Interconnection Fabric: An Example.

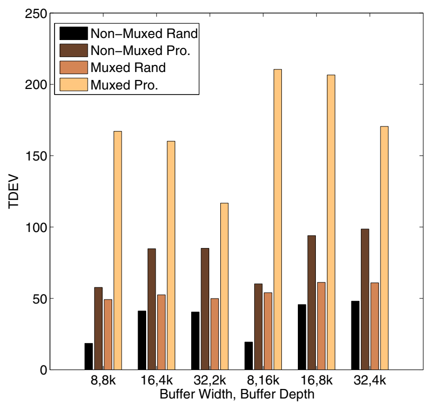


Begin
Move to next tracing period
Calculate TDEV by temporarily tracing each signal in current tracing period
Choose the one with maximum TDEV
Remove the signals with edges on chosen one
All trace signals are decided in current tracing period?
N
Y
All tracing period are decided?
N
Y
End

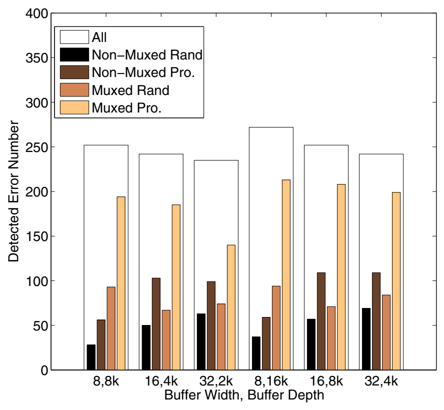
Figure 8: Flowchart of Signal Grouping with General Interconnection Fabric.

A general trace interconnection fabric can be the widely-adopted MUX network, the one with high flexibility introduced in [10] or any type of design specific structure. We represent the concurrent tracing relationship within the fabric as a graph, as the example shown in Fig. 7. Within the graph, a vertex denotes each accessible signal while an edge denotes the two connected signals cannot be traced together (e.g., they go through the same multiplexer). Hence a unified MUX network can be represented by a few cliques with no edge in between in the graph.

With the above graph representation, we propose a greedy method as depicted in Fig. 8 to solve the problem, wherein we incrementally select trace signals period by period to maximize TDEV. To decide which signals to trace in each sample period, we firstly estimate the resulted TDEV by temporarily choosing each relevant accessible signal remaining in the graph. Then, the one with the maximum $TDEV$ is chosen (e.g., b in Fig. 7). After that, we remove all nodes connected with the already-chosen one (e.g., a and h in Fig. 7) from further selection in the current tracing period, as these signals cannot be traced together with the selected one in this period. The procedure for each sample period ends when the number of selected signals reaches trace buffer width. As discussed in Section 3, the metric TDEV itself inherently guide us to maximize the probability of detecting design errors within the suspicious region.

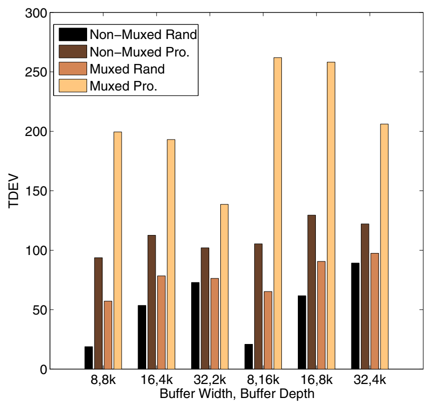


(a) TDEV Comparison

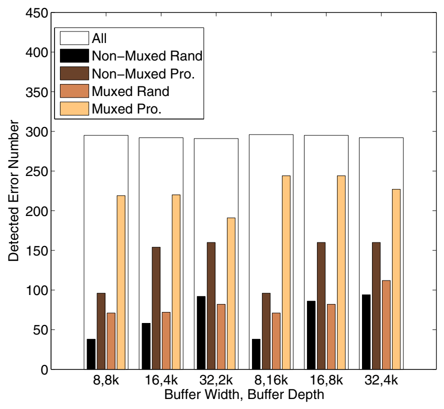


(b) Detected Error Number Comparison

Figure 9: Experimental Results for s38417.



(a) TDEV Comparison



(b) Detected Error Number Comparison

Figure 10: Experimental Results for s13207.

5. EXPERIMENTAL RESULTS

5.1 Experiment Setup

We conduct experiments on ISCAS'89 benchmark circuit s13207 and s38417 to evaluate the effectiveness of the proposed solution, and we consider the general interconnection fabric with 100 random selected accessible signals. As discussed earlier, based on the trace interconnection fabric implemented in the CUD, we can automatically construct the corresponding relation graph (see Fig. 7) to represent incompatibility among trace signals. Since this is not available to us, for the sake of simplicity, we randomly insert edges in the relation graph in our experiments.

To verify whether the tracing method can observe the errors' evidences or not, we randomly generate 1000 errors based on the widely-used "cell replacement" fault model in the literature [5]. Each time, we inject one of the errors in the circuit into the original netlist to obtain the erroneous netlist. Simulation is then conducted to dump the actual states. As no dedicated trigger mechanism is used in our experiments, the state dumping starts from the beginning of the simulation. These states are compared against the "golden vector" obtained from the simulation with the original netlist to get the propagated error evidences, by finding the difference between actual states and "golden vector". Finally, with different signal tracing methods, the evidence can be treated as visible if the capture FF is currently traced at the particular cycle. We can compare the number of visible errors from various signal tracing methodologies to demonstrate their effectiveness on error detection.

5.2 Experimental Results

Fig. 9 and Fig. 10 present our experimental results for s38417 and s13207, respectively. We set the length for each tracing period to *1k* (i.e., 1024) clock cycles. We also set the trace buffer size to be *64k* and *128k* in total and vary their width and corresponding depth to obtain the results with different buffer usage strategy. For example, the *64k* buffer can be used as $8 \times 8k$ or $16 \times 4k$. The buffer usage strategies are marked on X-axis. Hence, the left half of the figure shows the results for the *64k* buffer, while the right half is for the *128k* buffer. Buffer depth infers the number of tracing periods, e.g., when the buffer depth is set as *8k*, we have $\frac{8k}{1k} = 8$ tracing periods.

We compare the multiplexed tracing signals grouped with the proposed method (denoted by "Muxed Pro.") with other solutions, including non-multiplexed tracing signals grouped randomly (denoted by "Non-Muxed Rand"), non-multiplexed tracing signals grouped by proposed method (denoted by "Non-Muxed Pro.") and multiplexed tracing signals grouped randomly (denoted by "Muxed Rand"). All of them target the same suspicious region covering all the injected errors, assuming all the accessible signals are relevant. The results of the proposed evaluation metric and the number of detected errors resulting in evidences on traced information are denoted by "TDEV" and "Detected Error Number" in these tables, respectively.

First of all, the trend with our proposed evaluation metric $TDEV$ in Fig. 9 and Fig. 10 is roughly consistent with that of the number of detected errors, which demonstrates the effectiveness of this metric. At the same time, there are a few exceptions (e.g., for s13207 when the buffer is configured as $8k \times 16$). This is because, the $TDEV$ is for evaluating the error detection quality for the whole suspicious region, while the errors that are actually caught are just part of the signals in the suspicious region and they are also related to the particular debug run, which is unknown during the design stage.

In Fig. 9(b) and Fig. 10(b), the white bars with the legend "All" indicate the number of errors that result in evidences on accessible signals of the CUD. We can observe that only part of the injected errors can result in evidences on the trace-based DfD structure (24.2% to 29.6%). This is because: (i). the injected errors can only be activated by certain input sequence, which may not occur during the tracing cycles. (ii). the evidences generated by errors can be masked during the propagation so that they will not result in evidences on any FFs. (iii). The trace-based DfD structure only allows to trace 100 out of 1636 and 638 FFs for circuit s38417 and s13207, respectively, and hence those signals that capture error evidences may not be accessible. We also notice that the number of detected errors grows slowly with the increase of trace cycles. This phenomenon can be attributed to the fact that some errors are activated by certain condition that rarely occurs. In this case, the extension of tracing procedure can result in higher error detection chance. To note, the number of detected errors indicated with “All” tends to be higher than the one with the optimal trace signal grouping solution. This is because the DfD structure constraint only a small part from accessible signals to be traced. We can observe from results that our solution by tracing part of accessible signals can detect up to 82.7% of all detectable errors (as shown in Fig. 10 (b)), which demonstrates the effectiveness of our method. We also observe that most evidences captured by accessible signals are within two sequential levels from the root-cause, showing the effectiveness of trace solutions on error localization.

From the results, we can observe that the proposed multiplexed tracing solution outperforms significantly over the non-multiplexed one with the same signal grouping used in our first tracing period. Even for the closest case having two tracing periods, utilizing multiplexed tracing can still detect on average 27.8% more errors than the other. Even with random grouping, multiplexed tracing detects on average 37.3% more errors than the non-multiplexed one. When comparing against the case by randomly grouping a set of signals for each tracing period, as shown from Fig. 9 and Fig. 10, on average the proposed one detects 1.75 times more errors than the random solution. Even by comparing the non-multiplexed tracing with signals selected with proposed method against the multiplexed one with random grouping, the first one detects more errors in many cases (e.g., the result in Fig. 10 (b)).

The computational time of the proposed method is acceptable. It takes only tens of seconds for all cases on s38417 and less than ten seconds for s13207. For s38417, a large share of the computational effort is spent on error effect estimation, its runtime is almost independent of buffer usage strategy. While for s13207, the runtime almost grows linearly with the number of tracing periods for s13207. This is because it does not take much time for error effect estimation due to the small size of s13207, and hence most effort is spent on signal grouping.

Finally, to evaluate the area cost for the proposed solution, we implement and synthesize the DfD hardware using a commercial tool for multiplexed tracing and non-multiplexed tracing, respectively, with different buffer configurations. As shown in Table 2, the additional DfD area overhead (in NAND2 gate equivalent) of the proposed solution is quite small, less than 3.5% of the DfD cost for conventional non-multiplexed tracing. In particular, we can observe that within the DfD structures to facilitate multiplexed tracing, the storage element in debug controller dominates the extra DfD cost, while most of the hardware cost for conventional tracing is from trace buffer. Hence, for different trace buffer utilization types of the same capacity (e.g., $8 \times 8k$, $16 \times 4k$ and $32 \times 2k$), the original DfD cost remains almost constant, while the size of storage elements in our debug controller decreases linearly by the number of tracing periods (e.g., 8 to 4). Hence, the hardware overhead decreases correspondingly as shown in Table 2. It also explains why we find similar overhead when we double the trace buffer capacity (e.g., $8 \times 8k$ to $8 \times 16k$), as the required storage element in debug controller also doubles its size with the increase of tracing periods (e.g., 8 to 16). To note, the number of tracing periods can be flexibly determined to tradeoff debuggability and DfD cost.

6. CONCLUSION AND FUTURE WORK

In this work, we propose a novel multiplexed signal tracing method to maximize the design error detection capability, under various trace interconnection fabric constraints. Experimental results on ISCAS’89 benchmark circuits demonstrate the effectiveness of our solution. For future work, we plan to target on two problems: (i). How to select trace signals during design phase to facilitate better design error detection; (ii). Based on the error detection information that only provides *part-view* of the CUD, how to conduct effective diagnosis to root-cause the error.

Buffer Type	$8 \times 8k$	$16 \times 4k$	$32 \times 2k$	$8 \times 16k$	$16 \times 8k$	$32 \times 4k$
# of Period	8	4	2	16	8	4
$\Delta(\%)$	3.50	1.78	0.90	3.47	1.76	0.89

$\Delta = \frac{\textit{Extra Hardware Cost}}{\textit{Conventional Hardware Cost}} \times 100\%$

Table 2: Area Overhead of Proposed Multiplexed Tracing.

7. REFERENCES

[1] M. Abramovici. In-System Silicon Validation and Debug. *IEEE Design & Test of Computers*, 25(3):216–223, May-June 2008.

[2] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 7–12, July 2006.

[3] M. Boule, J. Chenard, and Z. Zilic. Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 613–620, 2007.

[4] K. H. Chang, V. Bertacco, and I. L. Markov. Simulation-based Bug Trace Minimization with BMC-based Refinement. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 1045–1051, 2005.

[5] K. H. Chang, I. L. Markov, and V. Bertacco. Fixing Design Errors with Counterexamples and Resynthesis. In *Proceedings IEEE Asia South Pacific Design Automation Conference (ASP-DAC)*, pages 944–949, 2007.

[6] A. B. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems On-chip: A Review. In *IEE Proceedings, Computers and Digital Techniques*, pages 197–207, July 2006.

[7] S. K. Jain and V. D. Agrawal. Statistical Fault Analysis. *IEEE Design & Test*, 2(1):38–44, 1984.

[8] D. Josephson and B. Gottlieb. Debug Methodology for the McKinley Processor. In *Proceedings IEEE International Test Conference (ITC)*, pages 451–460, October 2001.

[9] H. F. Ko and N. Nicolici. Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1298–1303, 2008.

[10] X. Liu and Q. Xu. Interconnection fabric design for tracing signals in post-silicon validation. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 352–357, 2009.

[11] X. Liu and Q. Xu. Trace signal selection for visibility enhancement in post-silicon validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1338–1343, 2009.

[12] S. B. Park and S. Mitra. IFRA: Instruction footprint recording and analysis for post-silicon bug localization in processors. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 373–378, 2008.

[13] S. Prabhakar and M. S. Hsiao. Multiplexed Trace Signal Selection Using Non-Trivial Implication-Based Correlation. In *Proceedings International Symposium on Quality of Electronic Design (ISQED)*, pages 697–704, 2010.

[14] P. Rashinkar, P. Paterson, and L. Singh. System-on-a-chip Verification: Methodology and Techniques. In *Kluwer Academic Publishers*, 2002.

[15] G. Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pages 892–902, September 1999.

[16] B. Vermeulen and S. K. Goel. Design for Debug: Catching Design Errors in Digital Chips. *IEEE Design & Test of Computers*, 19(3):37–45, May 2002.

[17] J. S. Yang and N. A. Touba. Automated Selection of Signals to Observe for Efficient Silicon Debug. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 79–84, 2009.