Dynamic Selection of Trace Signals for Post-Silicon Debug

Kanad Basu, Prabhat Mishra
Computer and Information Sc. and Engg.
University of Florida, USA
email: {kbasu, prabhat}@cise.ufl.edu

Priyadarsan Patra
Post-Si Validation Architecture
Intel Corporation, USA
email: priyadarsan.patra@intel.com

Amir Nahir, Alon Aadir
IBM Research
Haifa, Israel
email: {nahir, adir}@il.ibm.com

Abstract— **Post-silicon validation is one of the most expensive and complex tasks in today's System-on-Chip (SoC) design methodology. A major challenge in post-silicon debug is limited observability of the internal signals. Existing approaches address this issue by selecting a small set of useful signals. These signal states are stored in an on-chip trace buffer during execution. The applicability of existing methods is limited to a specific debug scenario where every component has equal importance all the time. In reality, a verification engineer would like to focus on a specific set of components (functional regions). Some regions can be ignored in a certain duration during execution due to clock gating and other considerations. Similarly, certain regions may be well verified datapath and less likely to have errors compared to other control-intensive regions. In this paper, we propose an efficient signal selection algorithm and a low-overhead trace controller design that would enable verification engineers to dynamically select a set of trace signals for improved error detection. Our experimental results using both ISCAS'89 benchmarks and Opencores circuits demonstrate that our approach can detect up to 3 times more errors compared to existing techniques.**

I. Introduction

An important challenge during post-silicon debug is the limited observability of internal signals since the chip has already been manufactured. Design overhead considerations limit the number of signals that can be traced or stored in an on-chip trace buffer. To improve the observability, existing techniques [1], [2], [4], [6] select a small set of useful signals during design time. The states of these signals are stored in an on-chip trace buffer during execution. The data from the trace buffer is then used for debugging using an off-line debugger. Unfortunately, the applicability of the existing methods is limited for various reasons. First, these methods treat each component (functional regions) of the design as equally important from debug perspective and therefore selects signals that are globally beneficial based on restoration capability. In other words, it assumes uniform "spatial" and "temporal" distribution of errors. In reality, certain regions may not be relevant in a certain duration for various reasons. For example, a set of cores in a multicore architecture may be in power saving mode (using clock gating) during certain timeframe. Therefore, no error is possible in those cores during that timeframe. Similarly, certain regions (such as well verified datapath) are less likely to have errors compared to other control-intensive regions. In general, only a small set of regions may be relevant during a certain duration for debugging an error. Therefore, a verification engineer would like to have knobs that allow him to trace a different set of signals at different timeframe.

In this paper, we propose an efficient signal selection algorithm and associated trace controller design that would enable verification engineers to dynamically trace different set of signals for improved error detection. This paper makes two important contributions. We propose a region-aware signal selection algorithm (RSS) that selects useful signals during design time (using static analysis) based on the knowledge of functional regions and associated error zones. We also develop a low-overhead dynamic signal tracing (DST) hardware to enable designers to trace different set of signals during execution based on active (relevant) functional regions. This lays emphasis on the errors in active zones in the circuit that can be detected using a specifically selected set of trace signals. We have developed an efficient spatio-temporal solution for dynamic trace signal selection. Our experimental results using both ISCAS'89 benchmarks and Opencores circuits demonstrate that our approach is able to detect up to 3 times more signals compared to existing state-of-the-art techniques.

The rest of the paper is organized as follows. Section II describes related work on signal selection. Section III compares signal restoration versus error detection. Section IV formulates the dynamic signal selection problem followed by the description of our region based signal selection algorithm in Section V. Section VI describes our dynamic signal tracing technique. Section VII presents our experimental results. Finally, Section VIII concludes the paper.

II. Related Work

Limited observability is a primary concern during post-silicon debug. Various signal selection techniques [1], [2], [4], [3], [6], [7] have been proposed for post-silicon validation and debug which focus on signal restoration across the entire circuit and are not designed for error detection. Recent approaches emphasize detection of timing errors [11], [12]. All of these approaches perform static signal selection i.e., the same set of signals are traced during the entire execution. The procedure developed by Prabhakar et al. [10] alternates between two sets of signals in alternate cycles. As a result, it is a very specific case of temporal distribution of errors without any consideration for spatial distribution. A multiplexed signal selection for error detection was proposed by Liu et al. [5]. Their approach is an ad-hoc signal selection heuristic based on error visibility metric. Their approach does not consider the challenges associated with dynamic signal selection in the presence of spatial and temporal distribution of errors. Lee et al. [9] described a dynamic signal sequence slicing approach in which they tried to generate the input sequence values that would result in an error rather than focusing on the trace signals. Recently, Han et al. [8] proposed a dynamic signal selection algortihm which focuses on signal resotration rather than error detection. The major limitation of their approach is that it depends on only the test vectors and does not take into account the currently active regions in the circuit, that is, it has no information on the current state of the circuit. In this paper, we propose a comprehensive spatio-temporal solution for dynamic trace signal selection to allow designers to choose different set of signals at different duration.

III. Signal Restoration versus Error Detection

This section compares two widely used metric, *state restoration* and *error detection*. Majority of the existing signal selection approaches [2], [4] try to focus on restoration of unknown signal using the knowledge of known signal states. Let's consider an *AND* gate with two inputs a and b and the output c. If the state of a is traced and found as 0, it is obvious that the state of c is also 0. Similarly,

This work was partially supported by NSF grant CCF-1218629.

if the state of c is traced and found as 1, it is obvious that both a and b will have states 1 each.

Let us consider an example circuit in Figure 1 comprising of 12 flip-flops. Tracing states of flip-flops A and L in cycle t, helps to restore the state of flip-flop D in cycle $t+1$, since the input to flip-flop D is an *OR* of the outputs of A and L. Similarly, since flip-flops C and H are connected by a *NOT* gate, tracing H in cycle t provides the state of C in cycle $t-1$. It should be noted that signal restoration can proceed in both forward (input to output) and backward (output to input) direction. For example, the restoration of D from A and L is in forward direction, while the restoration of C from H is in backward direction.

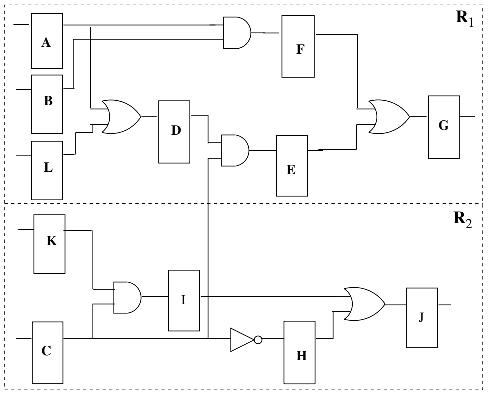

Fig. 1. Example circuit with two regions and 10 flip-flops

In case of error detection only backward restoration is meaningful. In Figure 1, when flip-flop D is in error, the bug can only propagate along a forward direction to its fan-out cone towards flip-flops E and G. Therefore, in order to detect the error in D, we have to trace either of these two flip-flops, and then apply backward restoration to restore D (error detection). Tracing the inputs of D, or those flip-flops that are in its fan-in cone (A and L) do not help. Given that a designer is interested in detecting an error, it is meaningful to focus directly on error detection metric instead of restoration performance.

IV. Problem Formulation

The goal of this paper is to develop an efficient dynamic signal selection technique to maximize detection of currently active errors[1] in a circuit. For example, if, the number of errors detected by signal i is represented by $detect_i$ and n is the trace buffer width, our goal is to select n signals such that $\sum_{i=0}^{n} detect_i$ is maximized. Various industrial studies highlight the fact that error locations are not uniformly distributed across the circuit, instead they are clustered in multiple small zones. In this paper, we call them error-prone zones (or *error zones*, in short). We assume that error zones during post-silicon validation closely resemble those in the pre-silicon phase. We divide the circuit into multiple parts where each part contains one or more error zones. We call these parts as *functional regions* (or *region*, in short). Information about error zones and regions are assumed to be provided by the pre-silicon engineer. A natural boundary for a region would be the component boundary of an SoC. For example, each core in a multicore SoC can form a region. If one component has multiple error zones, we may even divide that component into multiple regions following some functional boundary. For example, a processor core can be divided into two regions, one covering fetch and decode units and the other covering the rest. In our construction, an error zone is completely contained inside a region of the circuit.

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

There is a trade-off between number of regions versus error zones. One region per error zone may create too many regions (partitions) and lead to unacceptable computational complexity and hardware overhead. On the other hand, a large region with many disjoint error zones may reduce the effectiveness of dynamic signal selection.

Let us consider a circuit represented by the rectangle in Figure 2. The entire circuit is divided into m regions named R_1 to R_m. Each region can have one or more disjoint error zones. For the ease of illustration, we assume one error zone per region. It does not lose any generality since one error zone can be viewed as a composition of multiple disjoint error clusters. For example, the error zone Z_{R_1} for region R_1 consists of two disjoint error clusters in Figure 2. These two clusters together form the error zone Z_{R_1}.

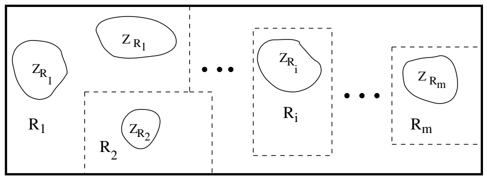

Fig. 2. Illustrative example showing regions and error zones

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

V. Region-Based Signal Selection (RSS)

Algorithm 1 describes our region based signal selection algorithm (*RSS*) for selecting useful signals during design time. The first step creates a graph-based model of the circuit. Next, for each region it computes the error propagation probability (defined in Section V-B) from each node in the error zone to the other nodes in the entire region. Finally, for each region the most profitable n signals are selected. The remainder of this section describes these steps in detail.

A. Graph Based Modeling of Circuits

The first step of Algorithm 1 is to construct a graphical representation of the circuit. We explain this step using our example circuit in Figure 1. Each signal in the circuit is represented by a node and any data flow between two nodes represented by an edge. The edge is irrespective of the type of gate between two nodes. The graphical representation is shown in Figure 3. For example, flip-flops C and H are connected by a *NOT* gate, hence, the two nodes representing them have an edge connecting them. Directed arrows signify the error propagation direction. R_1 and R_2 represent two different functional

Algorithm 1: Region-based Signal Selection (*RSS*)

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

regions of the circuit with respective error zones, Z_{R_1} and Z_{R_2}. Let us consider the region R_1. The probable sources of error are the nodes A and B. Any errors in these nodes can propagate to the other nodes in R_1 which are in their respective fan-out cones. Therefore, the error at A can propagate to F,D,E and G. We would like to compute the possibility of an error at any of these two probable erroneous nodes (A and B) to propagate to the other nodes. We call this probability as *error propagation probability*, as described next.

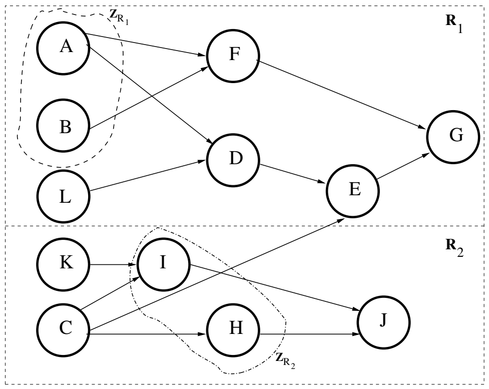

Fig. 3. Graphical representation of Figure 1 with two regions

B. Error Propagation Probability Computation

We first describe how to compute error propagation probability through single gates. Error propagation probability is defined as the probability of an error present at an input of a gate being propagated to its output. Error propagation probability over multiple gates will be explained later. We compute the probability of an error at one of the inputs getting propagated to the output of an individual gate.

Individual Gates: To compute the probability of error propagation, we first consider a multi-input *AND* gate. Let the inputs be named i_1, i_2,...,i_n and the output o_1 respectively. Let us assume an error occurs at one of the inputs, say, i_1. We want to compute the probability of the error to be propagated to o_1. In order for any error (0/1 or 1/0) to propagate to o_1, it is necessary that all the other inputs of the *AND* gate be tied at 1. If any of the other inputs is at state 0, the output will always be at a state of 0, irrespective of i_1, hence, the error gets undetected. Here, we assume all the other inputs to the *AND* gate are independent. Therefore, the probability that all of them are 1 simultaneously is the product of each of the individual probabilities. Let $p^1_{i_k}$ be the probability that input i_k is at state 1. Therefore, the probability that all the other inputs are at 1 is $P^1_{i_1} = \prod_{2\leq k\leq n} p^1_{i_k}$ which is the probability that an error at i_1 will get propagated to o_1, that is the error propagation probability through the *AND* gate. Similar computations can be performed for a *NAND* gate.

The computations for an *OR* gate follows the approach similar to an *AND* gate. Let the inputs be named i_1, i_2,...,i_n as before, and the output o_2 respectively. Let's consider the error propagation from i_1 to o_2. In order to propagate an error in i_1 to o_2, all the other inputs of the *OR* gate must be held at a state of 0. The probability that an input i_k is held at 0 is $p^0_{i_k}$. The joint probability that all the inputs other than i_1 is held at 0 is $P^0_{i_1} = \prod_{2\leq k\leq n} p^0_{i_k}$ which is the error propagation probability from i_1 to o_2. Similar computations can be performed for a *NOR* gate. For any one input and one output node (such as flip-flop and *NOT* gate), the error propagation probability is always 1.

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

Independent Paths through Multiple Gates: An independent path is one which passes across a set of logic gates with each gate being visited at most once. We explain the independent path scenario using Figure 3. In this example, we assume that each internal signal can be in a state of 0 or 1 with a probability of 50%. This assumption is for the ease of illustration; in our implementation as well as during experiments (presented in Section VII), we use profiling information to determine the state probability. The edge from A to E is an independent path, since there exists only one path from A to E, via D. Since there is only an *OR* gate between A and D, the probability of an error at A getting propagated to D is the probability of L (the other input to the *OR* gate) being in a state of 0, which is 0.5 in this case. Hence, the error propagation probability between A and D is 0.5. Similarly, since there is only a 2-input *AND* gate between D and E, the error propagation probability between D and E is 0.5. Since none of the signals are visited more than once, the overall error propagation probability between A and E is the product of these two, which is 0.25. In general, if there are $n+1$ signals in an independent path between nodes s and d, with their intermediate error propagation probabilities being p_1, p_2,..., p_n, the overall error propagation probability across the path is $P_{(s,d)} = \prod_{1\leq k\leq n} p_k$

Dependent Paths through Multiple Gates: A dependent path is one in which while moving from a source node to a destination node, at least one of the internal nodes is visited more than once[2]. We explain the error propagation probability computation using Figure 3. There exists two independent paths between nodes A and G. One edge is (A,F,G) while the other is (A,D,E,G), both branching out at A and combining at G. In order to compute the error propagation probability across the path between A and G, we need to compute these independent path values separately. For the path (A,F,G), the error propagation probability is the product of the probabilities between the paths (A,F) and (F,G), both of which, for obvious reasons are 0.5. Thus, the error propagation probability of path (A,F,G) is 0.25. On the other hand, since the path (A,D,E,G) passes through 3 independent two-input gates, the eventual error propagation probability is 0.125. The error propagation probability through path

[2]The same computations are used even if a combinational gate is visited twice instead of a flip-flop.

(A,G) can be computed as $p_{(A,G)} = max(p_{(A,F,G)}, p_{(A,D,E,G)})$. This is because during computation, the effect of two different paths are already taken into account, and a path with a higher probability of detecting an error will always dominate. In general, if there are n independent paths e_1, e_2,..,e_n between two nodes s and d, then the error propagation probability of the paths between s and d, $p_{(s,d)} = max(p_{e_1}, p_{e_2}, ..., p_{e_n})$

C. Signal Selection Based on Node Values

In this section, we describe the final step in our signal selection algorithm. The first node chosen for tracing in a region is the one with the highest node value. The value of a node is the sum of error propagation probabilities of all paths in which the node is the destination. For example, in Figure 4, if we concentrate on Region R_1, the node value of E will depend on paths (A,D,E), (L,D,E) and (D,E). Since in this example only A and B are possible error locations for R_1, the relevant path would be (A,D,E); the paths (L,D,E) and (D,E) are not relevant because D and L are not in error zone. Therefore the node value of E will be the sum of error propagation probability across the path (A,D,E), that is, 0.25. We can have similar computations for other regions. The node values of all the nodes in R_1 are shown in Figure 4. Each node value is represented by a number beside it.

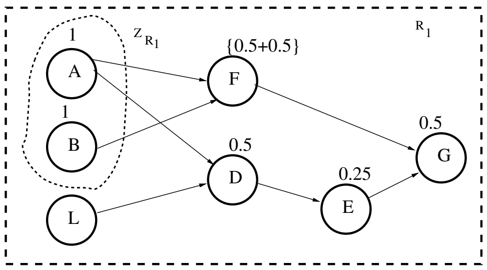

Fig. 4. Node values for region R_1 in Figure 3

In Figure 4, three nodes (A, B and F) have highest node values of 1. A and B are not valid choices since any of them cannot detect the error in other node, whereas F can detect error in both A and B with 50% probability. Therefore, we choose F as the first node to trace. The subsequent signals should be carefully selected to enhance the error detection in the region. Contributions from signals which have a high error detection probability from the already selected signals should be deleted. Step 4.4 of Algorithm 2 is used for this purpose. The basic idea is that if an already selected node (e.g., F) can detect an error (e.g., in A) with equal or higher probability than another node (e.g., D), then the overlap from the other node should be removed. For example, since F can detect an error in A with 50% probability, the contribution claimed by D (also 50% for A) should be deleted from D during the next iteration. This process continues until n best signals are selected for each region or there are no more signals to be selected.

VI. Dynamic Signal Tracing (DST)

Algorithm 2 describes our dynamic signal tracing (DST) procedure for improved error detection. The inputs to the algorithm are the chip design, trace buffer size, active regions and associated relevance and signal lists. *Relevance* of a region indicates how important the region is in error detection, or the possibility of finding an error in that region compared to other regions. The relevance information is provided by the pre-silicon verification engineer based on percentage of errors found in the error zone in that region (compared to other error zones) during pre-silicon validation[3]. If no information is

[3]The probability of errors present in each zone during the post-silicon phase is assumed to be similar.

Algorithm 2: Dynamic Signal Tracing (DST)

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

available, we can consider the size of the error zone in that region as relevance. If the trace buffer size is n, and there are m active regions in the circuit, during design time our RSS procedure (Algorithm 1) will select $m \times n$ signals. During execution, our DST procedure needs to choose n signals from these $m \times n$ signals that are most profitable at a certain duration depending on the k ($1 \leq k \leq m$) active regions.

Since we have to select n out of $m \times n$ signals for tracing, it is reasonable to adopt n multiplexers, each of which will provide a signal corresponding to the trace buffer output. The main problem is to divide the $m \times n$ signals among the n multiplexers so that all possible combinations of trace signals can be achieved. An obvious but expensive solution would be to use n multiplexers each having all the $m \times n$ signals as input and 1 output.

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

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

Under each multiplexer, all these signals except the one from the same region will be stored. For example, consider Figure 5 with m =3 and n = 3. The original design of each multiplexer is 9×1 whereas our optimized design is 5×1, providing reduction of $\frac{9}{5}$ = 1.8.

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

The overall structure of our proposed design is shown in Figure 6. Here we consider a design with n multiplexers that would produce n

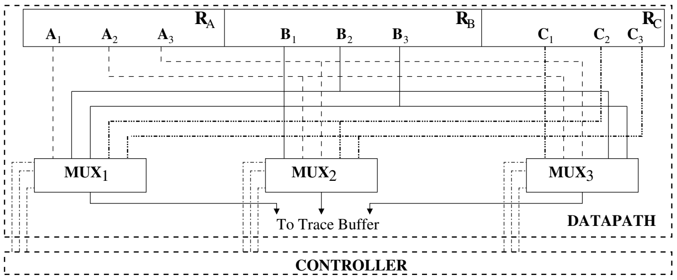

Fig. 5. Optimized trace controller design for m = 3 and n = 3

TABLE I

TABLE FOR $n = 2$ AND $m = 2$

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

trace signals. The output of the multiplexers are fed to a trace buffer. The trace controller provides the control signals to the multiplexers based on the logic mentioned above. The trace controller operates under the same clock as the Design Under Test (DUT). An external knob is applied on the trace controller (generally by the validation engineer) which contains information about the currently active error zones in the circuit.

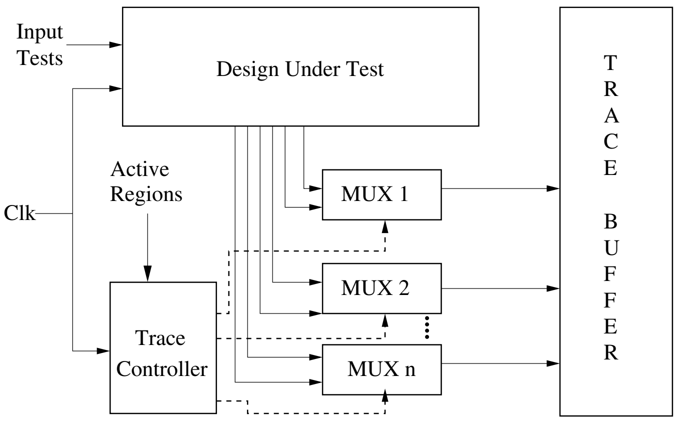

Fig. 6. Proposed dynamic signal tracing hardware

VII. EXPERIMENTS

A. Experimental Setup

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

$$EDR = \frac{Number\ of\ Errors\ Detected}{Number\ of\ Detectable\ Errors}$$

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

- **GSS:** This approach represents the existing techniques that focus on global signal selection (GSS) without any knowledge of error zones or active regions, it assumes that the errors are uniformly distributed across the circuit. The signals are selected using an approach similar to [4]; the only difference being that we have considered error detection and not restoration.
- **EZ-GSS:** We extend the existing methods with the knowledge of the error zones to evaluate their effectiveness in handling error zones. We call this approach as error-zone aware global signal selection (EZ-GSS). This is a static signal selection assuming all zones are active.
- **RSS+DST:** Our approach is essentially a combination of region-aware signal selection (RSS) and dynamic signal tracing (DST).

B. Results for Two Regions

For each of our experimental circuits, we created two regions each having one error zone. In the first set of experiment, we assume both zones are active. In this case, $EZ-GSS$ and $RSS+DST$ are same, since we have to consider both zones for signal selection even during DST. The results are shown in Figure 7. As expected, our approach performs better than GSS, with the maximum improvement being 1.75 times, since our approach lays more emphasis on the error zones.

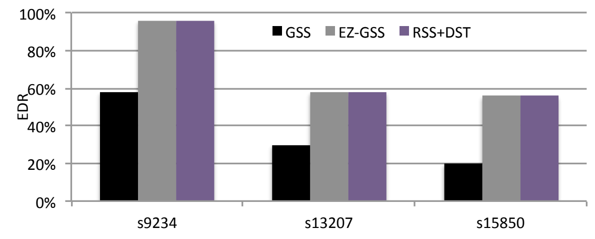

Fig. 7. Comparison of EDR performance when both regions are active

In the next experiment, we assume one of the two error zones are active at a particular time. Now, we would like to compare the EDR performance of our three approaches, GSS, $EZ-GSS$ and $RSS+DST$. The results are shown in Figure 8. GSS performs the worst among the three since it has no knowledge of where the error is located or which region is active. $EZ-GSS$ performs better than GSS but has no knowledge of active regions. $RSS+DST$ performs the best since it dynamically selects signals with the complete knowledge of currently active error zones, perform best. Please note that, there are two possible scenarios for one active region: R_1 is active or R_2 is active. We present the average of both of these cases. The maximum improvement obtained by our approach against GSS is almost 3 times.

We would now like to observe the performance of our approach on some real circuits obtained from the Opencores[13] website. We choose three circuits for our purpose, namely RS232 Uart, OPB Onewire and i2cslave. These will be referred to as *uart*, *one* and *slave*, respectively for further discussion in this section.

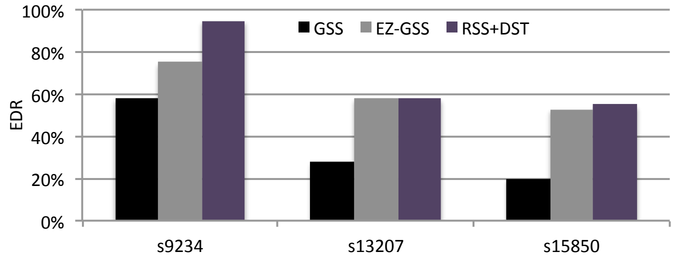

Fig. 11. Comparison of *EDR* performance when two regions are active

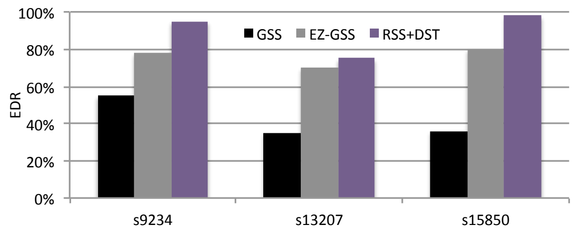

Fig. 8. Comparison of *EDR* performance when only one region is active

We synthesized these using Synopsys Design Compiler to obtain the gate-level netlist from the RTL descriptions. For each of these circuits, we consider two error regions of which one is active at a time. The results are shown in Figure 9. As expected, for all three benchmarks, our proposed methods $EZ-GSS$ and $RSS+DST$ performs much better than GSS. $RSS+DST$ performs best in all cases; however for *one* performance of $EZ-GSS$ and $RSS+DST$ are similar. This is because of all the signals selected using $RSS+DST$, the ones which can detect most of the errors are selected using $EZ-GSS$ as well.

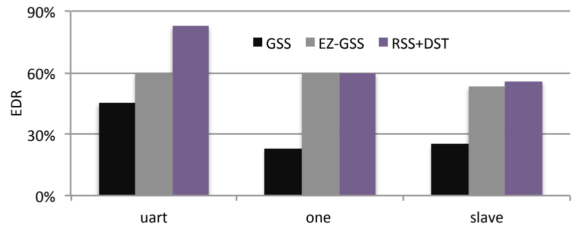

Fig. 9. Comparison of *EDR* performance on the Opencores circuits

C. Results for Three Regions

In these experiments, we created three regions for each circuit. In the first experiment, we assume only one of the three regions are active. The *EDR* performance comparison using ISCAS '89 benchmarks is shown in Figure 10. The $RSS+DST$ numbers are the average of three possible scenarios of one active region in the circuit. Up to 3 times improvement is obtained by our approach compared to GSS, while compared to $EZ-GSS$, $RSS+DST$ has a maximum improvement of 1.56.

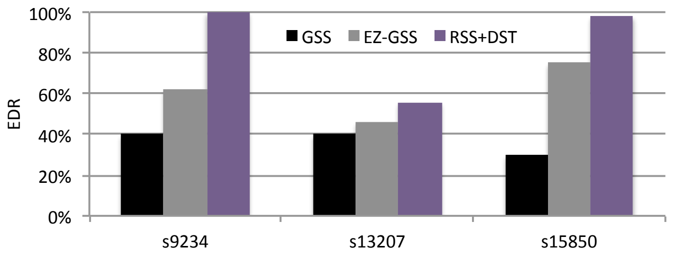

Fig. 10. Comparison of *EDR* performance when one region is active

In the next set of experiments, we compare when two of the three regions are active. The results are shown in Figure 11. The $RSS+DST$ numbers are the average of three possible scenarios of two active regions in the circuit. $RSS+DST$ performs the best among the three approaches, with the maximum improvement obtained 2 times (compared to GSS) and 1.3 times (compared to $EZ-GSS$). We also performed experiments for 4 regions and made similar observations. Due to space limitations, we omit those results.

D. Hardware Overhead

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

VIII. Conclusion

Limited trace buffer size constraints the number of signals that can be observed during post-silicon debug. Existing trace signal selection techniques operate on the basic assumption that errors are uniformly distributed across the circuit. During design time, our region-aware signal selection approach selects beneficial signals for each region based on information regarding error zones. During execution, our dynamic signal tracing controller enables designer to trace a different set of signals based on regions that are relevant (active) during a certain duration. Our experimental results demonstrated that our approach can detect significantly more (up to 3 times) errors compared to existing approaches.

References

[1] H. Ko et al., "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *TCAD*, pp. 285–297, 2009.
[2] X. Liu et al., "Trace signal selection for visibility enhancement in post-silicon validation," in *DATE*, 2009, pp. 1338–1343.
[3] S. Prabhakar et al., "Using Non-Trivial Logic Implications for Trace Buffer-based Silicon Debug," in *ATS 2009*, pp. 131–136.
[4] K. Basu et al., "Efficient Trace Signal Selection for Post Silicon Validation and Debug," in *VLSI Design*, 2011.
[5] X. Liu et al., "On multiplexed signal tracing for post-silicon debug," in *DATE*, 2011, pp. 1 – 6.
[6] D. Chatterjee et al., "Simulation-based signal selection for state restoration in silicon debug," in *ICCAD*, 2011, pp. 595–601.
[7] M. Li et al., "A hybrid approach for fast and accurate trace signal selection for post-silicon debug," in *DATE*, 2013, pp. 485–490.
[8] K. Han et al., "Dynamic Trace Signal Selection for Post-Silicon Validation," in *VLSI Design*, 2013, pp. 302–307.
[9] Y. Lee et al., "On-chip dynamic signal sequence slicing for efficient post-silicon debugging," in *ASPDAC*, 2011, pp. 719–724.
[10] S. Prabhakar et al., "Multiplexed trace signal selection using non-trivial implication-based correlation.," in *ISQED*, 2010, pp. 697 – 704.
[11] J. S. Yang et al., "Automated selection of signals to observe for efficient silicon debug," in *VTS*, 2009.
[12] H. Shojaei et al., "Trace signal selection to enhance timing and logic visibility in post-silicon validation", in *ICCAD 2010*, pp. 68 – 72
[13] www.opencores.org