Emphasizing Functional Relevance Over State Restoration in Post-Silicon Signal Tracing

Debjit Pal, *Student Member, IEEE*, Sai Ma, and Shobha Vasudevan, *Senior Member, IEEE*

***Abstract*—The state restoration ratio (SRR) has been a de facto standard for evaluating the quality of signals selected for post-silicon tracing and debug. In this paper, we establish that SRR is intrinsically unsuitable as a metric for evaluating trace signal quality, as it captures neither the higher-level functionality of the design nor the constraints and requirements on trace signals. We present an algorithm, based on PageRank [PageRank on Netlist (PRoN)], for post-silicon trace signal selection. PageRank is *not* designed to maximize SRR and is applied to the circuit netlist. We demonstrate that optimizing for SRR typically generates signals that are functionally irrelevant to the design and unusable for debug, for a comprehensive set of SRR-based techniques. We assess the scalability of different signal selection algorithms by applying them to an industrial scale OpenSPARC T2 design. Our results show that our PRoN algorithm consistently outperformed other techniques with respect to scalability and functional relevance of signals selected. It also has higher restorability than the other algorithms, despite not being optimized for that metric.**

***Index Terms*—Behavioral coverage, OpenSPARC T2 system-on-chip (SoC), PageRank, signal selection, state restoration ratio (SRR).**

I. Introduction

POST-SILICON validation is a critically important [30] and expensive activity, accounting for the majority of the validation expense in modern system-on-chip (SoC) designs [37].

A fundamental problem of post-silicon validation is limited observability and control. Only a few hundred among millions of internal signals can be traced or controlled during silicon execution. These signals must be selected *a-priori* through analysis of pre-silicon design collateral, so that the design can be instrumented with hardware to route them to an observation point (e.g., pins and trace buffers). If critical signals are missed, their omission can be identified only during post-silicon validation, as an inability to root-cause an observed failure. At that point, rectifying the omission would require a significant change in the observability architecture together with silicon respin, which is not feasible.

Manuscript received March 16, 2018; revised August 14, 2018 and October 15, 2018; accepted December 5, 2018. Date of publication December 14, 2018; date of current version January 18, 2020. This paper was recommended by Associate Editor S. Gao. *(Corresponding author: Debjit Pal.)*

The authors are with the Electrical and Computer Engineering Department, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: dpal2@illinois.edu; saima1@illinois.edu; shobhav@illinois.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2887047

Given the severity of the impact of missing observability, there has been significant research in the "signal selection problem," or maximizing design visibility as necessary for post-silicon debug under observability restrictions [10], [16], [17], [20], [21], [25]. While there are significant differences in the specific approaches proposed, virtually all related work uses the same metric, called the *state restoration ratio* (SRR), for evaluating approaches. The SRR measures the number of design states reconstructed from the signals observed; a set S of signals is considered superior to another set S' if more design states can be inferred from observing S than S' (Section II-C). Most signal selection algorithms include heuristics to efficiently identify signals that maximize SRRs.

In this paper, we establish that despite its wide use as a de facto standard in signal selection research, SRR is, in fact, a poor metric for determining the quality of post-silicon trace signals.

Why is SRR a poor metric for qualifying post-silicon trace signals? Post-silicon validation exercises deep and interesting functional behaviors of the design, e.g., booting of an operating system, running of target user-level applications, and execution of different power-management mode. For each such behavior, certain specific design states are critical, while others might be irrelevant. A metric for post-silicon trace signals should reward (or highly favor) the selection of signals that facilitate understanding, interpretation, and validation of such high-level functionality during silicon execution. On the other hand, restorability as a metric takes a "myopic" view of state reconstruction, attempting to maximize restoration of gate-level states without proper prioritization.

An attempt to maximize SRR frequently results in ignoring the natural design structure and context/conditions of monitoring signals, causing the selected signals to be functionally irrelevant to the design.

In previous work [26], we endeavored to increase the functional relevance of selected signals by departing from the SRR optimizing strategy of prior art. Instead, our approach was to let the design structure indicate importance of signals. Our algorithm is based on the Google PageRank algorithm [5], as applied to the circuit netlist. The algorithm gives us a rank ordering among important signals for tracing. We compared the signals selected by our method with the signals from SRR-based techniques. We used pre-silicon simulation coverage metrics to establish functional relevance of selected signals. We performed signal selection experiments on a USB design [8], which has substantially more complex behavior

than ISCAS89 benchmarks used in the literature until then. Our results showed that compared to SRR-based methods, our method selected signals with high functional relevance. Further, we plotted SRR against the behavioral coverage achieved by the signals selected by SRR optimizing methods [10] and our method. We found that high SRR values do not correlate to high behavioral coverage.

Scalability is an important concern in automatic post-silicon trace signal selection as its methods work on a fine-grained netlist level. A modern SoC contains hundreds of different intellectual property (IP) blocks [1] with millions of logic elements such as flip-flops (see Fig. 6). SRR-based methods [10], [16], [22], [33] update the rate of restorability of each flip-flop in the design in each iteration based on the currently selected trace signals. For a large-scale design such as a modern SoC, this iterative update is computationally expensive and has a chance to run out of time and/or memory. This considerably limits the scalability of the state-of-the-art algorithms. On the other hand, our PageRank-based algorithm as applied to netlist (PRoN) avoids the restorability computation altogether, relying on design structure and connectedness as the guideline for signal selection. This is a cheaper operation.

In this paper, we focus on scalability, and demonstrate experiments at an industrial scale. We show results on the publicly available multi-core SoC design, OpenSPARC T2 [4]. OpenSPARC T2 contains several heterogeneous IPs and reflects many of the complex features of an industrial SoC design. We selected several large and complex modules from OpenSPARC T2 that contain up to 14,000 flip-flops and up to 74,000 logic elements for our experiment. The scale and complexity of these design modules are several orders of magnitude greater than those of the traditional ISCAS89 benchmarks used in signal selection literature. This added complexity helps to illustrate the divergence between gate-level state restorability and functional behavior.

Our experiments on OpenSPARC T2 design modules showed that state-of-the-art signal selection techniques [10], [16], [22], [33] could not finish signal selection for designs consisting of no more than 2800 flip-flops due to timeout and large peak memory usage (up to 30 GB). Our PRoN algorithm was able to select trace signals for designs containing approximately 14,000 flip-flops within 13 s with a peak memory usage of up to 1.5 GB. Our results showed that PRoN has much better scalability than other state-of-the-art signal selection algorithms for industrial scale designs.

While the original PageRank was sufficiently accurate for our ISCAS, and USB experiments in [26], application of this algorithm to the large scale OpenSPARC T2 introduced problems that needed to be addressed at scale. Complex interconnections such as feedback and feed-forward loop structures among flip-flops and deep hierarchical signal connections from instantiated modules to the top module exposed issues in the original PageRank algorithm. The density and complexity of connections in the real design caused PageRank to infer that the outputs were the most important. For our purpose, tracing an output signal does not add any value, since it is observable anyway. In this paper, therefore, we modify PageRank to correctly rank internal signals for large, complex designs. We will refer to this modified PageRank as PageRank on netlist (PRoN) hereafter.

In this paper, we also provide a more comprehensive experimental study to compare the quality of the selected trace signals in terms of behavioral coverage by using total restorability-based [10], [26], hybrid-analysis-based [22], ILP-based [31], [33], and simulation-based [16] signal selection algorithms.

Our experimental results, when we applied the algorithms to OpenSPARC T2, were in conformance with the results presented in our previous work [26]. We compared our PRoN method with the only two SRR-based techniques that could finish for at least some of the OpenSPARC T2 design modules. *The behavioral coverage of the signals selected by our PRoN method consistently outperformed (up to 50.94% more) the signals selected by SRR optimizing methods [10], [22].* Further, we showed that signals selected by PRoN executed up to 4.59% more design paths than did signals selected by SRR-optimizing methods on large-scale designs. PRoN achieves higher path coverage for the signals than selected by the SRR-optimizing methods due to enhanced PageRank metric as it prefers flip-flops that are highly connected and part of many design paths.

For completeness, we determined the extent of restorability achieved by all the algorithms, including PRoN. Interestingly, the signals selected by PRoN although not optimized for SRR, often achieve up to 7.3× (on an average 3.15×) higher restorability on large scale designs compared to signals selected by SRR-optimizing methods.

Our contributions over [26] are as follows. First, we demonstrate the scalability and viability of our PRoN signal selection algorithm on the OpenSPARC T2 SoC design modules containing up to 14,000 flip-flops and up to 74,000 logic elements. To the best of our knowledge, this is the largest scale application of netlist level signal selection approaches demonstrated in literature. Second, we enhance the PageRank algorithm so it can select high quality tracing signals at scale. Third, we provide a comprehensive comparison of our PRoN technique with all the signal selection-based techniques (and tools) available in the public domain. This provides conclusive empirical evidence for the typical functional irrelevance of signals selected by state-of-the-art SRR-based methods.

II. Preliminaries

A. PageRank Algorithm

Google PageRank algorithm ranks a Web page as important if it is hyperlinked from many important Web pages. This ensures that not all hyperlinks have equal weights. PageRank computes an importance score for each Web page based on its incoming hyperlinks. Let p denote a Web page. Let $B(p)$ denote the set of pages that have an outgoing link to p, and let $F(p)$ denote the set of pages to which p has outgoing links. Let ϵ be a constant between 0 and 1.0 and let n be the number of Web pages. The PageRank $\mathrm{PR}(p)$ of p is defined as

$$\mathrm{PR}(p) = (1-\epsilon) \sum_{p_i \in B(p)} \frac{\mathrm{PR}(p_i)}{|F(p_i)|} + \frac{\epsilon}{n}. \tag{1}$$

The first term in (1) represents the probability that a random surfer will navigate to a Web page. If the surfer is caught in a cycle of Web pages, then it is unlikely that he or she will continue in the cycle forever. The second term accounts for the surfer's eventual departure from the cycle and navigation to a random Web page.

B. Hardware Signal Tracing

Hardware tracing is one among many different design-for-debug architectures that are used to address the observability limitation during post-silicon debug. The basic idea is to trace a set of signals during post-silicon execution and store them in on-chip memory called the *trace buffer*. Trace signal selection needs to maintain various design constraints of which trace buffer size directly translates to the area, power, and routing congestion. As a result, only a few hundred among millions of signals can be traced for a few thousand cycles.

A trace buffer has two parameters.

1) *Width*, i.e., the number of bits of signals that can be traced simultaneously.
2) *Depth*, i.e., the number of cycles for which signals values can be traced.

In post-silicon debug, unknown signal values can be reconstructed from the traced values in two ways—forward restoration and backward justification [19].

C. Signal Reconstruction and SRR Calculation

A metric used frequently to measure quality of the selected signals is the SRR [10]. SRR is defined as *the sum total of the number of signals traced and the number of signals restored expressed as a fraction of the number of signals traced*, i.e., SRR = (total number of signals traced + total number of signals restored)/(total number of signals traced).

We calculate SRR for the simple circuit shown in Fig. 1. Let us assume that the trace buffer can record values of two signals. The restored values of the other signal states that use the method of [10] are shown in Table I. The signals that are chosen via total restorability computations are *A* and *C*. The selected signals are shown in gray. Since ten signal values are traced and 22 values are restored, the SRR with this selection is 3.2.

D. Simulation-Based Coverage Metrics

In this section, we define several simulation-based coverage metrics that we use in our experimental results.

Line coverage is defined as the fraction of total design statements (like *blocking, nonblocking*, and *assign* statements) that are executed in a design simulation. **Branch coverage** is defined as the fraction of total branches (like *if-else* and *case statements*) that are executed in a design simulation. **Condition coverage** is defined as the fraction of conditions of all branches that are executed in a design simulation. **Path coverage** is defined as the fraction of total design paths that are executed in a design simulation. **Toggle coverage** is defined as the fraction of total bits of a wire/register that change from a value of zero (1'b0) to one (1'b1) and back from one (1'b1) to zero (1'b0) in a design simulation. A bit is said to be fully covered when it toggles back and forth at least once.

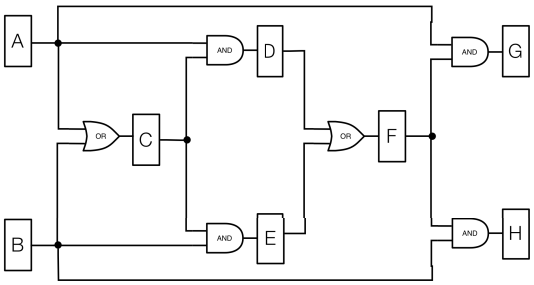

Fig. 1. Example circuit [10].

TABLE I
STATE RESTORATION FOR CIRCUIT SHOWN IN FIG. 1 APPLYING ALGORITHM OF [10]

Signal	Cyc1	Cyc2	Cyc3	Cyc4	Cyc5
A	0	0	0	0	1
B	1	0	1	0	X
C	1	1	0	1	0
D	X	0	0	0	0
E	X	1	0	0	0
F	X	X	1	0	0
G	X	0	0	0	0
H	X	X	0	1	0

III. INADEQUACY OF SRR AS METRIC

A. Motivating Example

In this example we provide a comparison of selected signals corresponding to interesting high-level behavior between SRR-based signal selection methods and our proposed PRoN method. We show that using LC3B [2], a 16-bit academic processor in which we attempt to reconstruct the micro-architectural state.

We applied the SRR-based signal selection technique SigSeT_1 [10] that is designed to maximize the SRR [10]. It selects the complete ISDU finite state machine (FSM) of LC3B state registers, some bits of the program counter (PC), and some bits of the instruction register (IR) at the top of the list. With this set of signals, we can recreate a few control states, but not the rest of the processor state. Without the complete PC and IR, it is not possible to determine which instruction will be processed and fetched from memory next.

As a point of contrast with the above results, consider the performance of our PRoN algorithm that does not seek to maximize SRR (see Section IV) on the same example. PageRank selects all of the ISDU FSM state registers, all 16 bits of PC and IR as complete words, and NZP branching registers. This is sufficient to check the sequence of states in the design, the opcode and operands fetched, all transitions in the control state machine, and branching behavior. PRoN ranks all of the control signals with high priority, while ranking eight 16-bit data registers lower. This helps in reconstructing the micro-architectural state of LC3B.

The above example suggests a key problem with the utility of SRR as a metric: it treats all gate-level design states

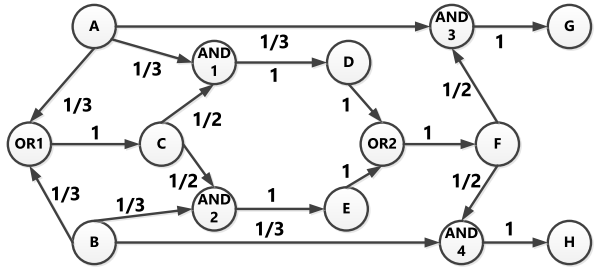

Fig. 2. Circuit network $\mathcal{G}$ for the example circuit of Fig. 1. Each node in $\mathcal{G}$ is a logic element and each edge in $\mathcal{G}$ follows the connectivity in the circuit. Each edge of $\mathcal{G}$ is annotated with importance contributions from each node.

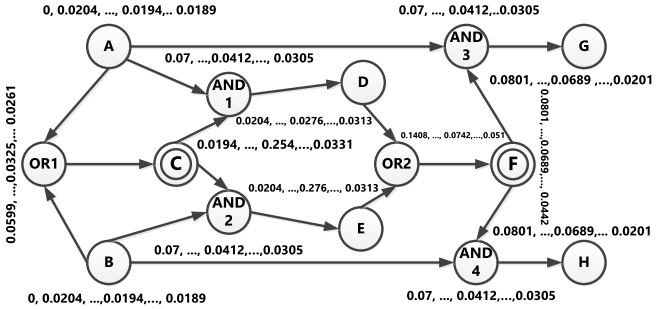

Fig. 3. Nodes of $\mathcal{G}$ of Fig. 2 annotated with importance values in successive iterations and the final importance value as calculated by PRoN. PRoN selects flip-flop F and C (shown in double circle) to trace as trace buffer width is 2.

as "equals." Reconstructing any specific design state is not considered more valuable than reconstructing any other state. However, practical debugging experience suggests that some signals are inherently more valuable for validation and debug than others. Also, some signals can provide useful state information only in the presence of some other signals as well. For example, reconstructing only the lower-order bit of a PC provides little information on program behavior or execution flow, while reconstructing all bits of the PC can provide significant insight. Consequently, signals selected to optimize SRR do not necessarily facilitate debug.

B. Deconstructing SRR Inadequacies

We find that SRR is not useful for signal selection for designs with the following features.

1) **Large arrays**, where SRR tends to reconstruct individual array element values.
2) **On-chip instrumentation**, since SRR selects signals insufficient to validate instrumentation.
3) **Complex SoC protocols**, since SRR favors signals in larger IP blocks and misses smaller ones, routers and other important signals.

IV. PageRank-Based Trace Signal Selection Algorithm

We apply the PageRank algorithm [5], [13] to the circuit netlist. Algorithm 1 details the PRoN algorithm. In this section, we apply it on the example circuit shown in Fig. 1.

1) ***Network Construction:*** We parse the synthesized netlist of an RTL design to construct a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the connectivity between different logic elements, where every $v \in \mathcal{V}$ represents a logic element and every directed edge $(v_i, v_j) \in \mathcal{E}$ represents a connection between the logic elements v_i and v_j. Fig. 2 shows the directed graph for the example circuit in Fig. 1.
2) ***PageRank Value Calculation:*** After constructing the directed graph for the circuit, we apply PageRank algorithm to compute the importance of each node. The directed graph in Fig. 2 has 14 logic elements (eight sequential elements and six logic gates). Each node transfers its importance equally to the nodes to which it links. For example, node A has three out-links, so it will transfer 1/3 of its importance to each of the nodes OR1, AND1, and AND3. In general, if a node has n out-links, it will pass on $1/n$ of its importance to each of the nodes to which it is linked. Following this importance transition rule, we annotate every edge of the graph in Fig. 2 with the corresponding importance value.

Algorithm 1 Pseudo-Code of PRoN Algorithm

```
procedure PRoN(G, G', error, ϵ)
G = (V, E), G' = (V, E') {if (v_i, v_j) ∈ E, then (v_j, v_i) ∈ E'}
error: error bound for rank matrix convergence
ϵ: damping factor
prank_1 ← PageRank(G, error, ϵ)
prank_2 ← PageRank(G', error, ϵ)
for v in G do
    prank_hm(v) ← HM(prank_1(v), prank_2(v)) {HM: Harmonic Mean}
end for
```

Initially, we assume an equal rank for each of the nodes, i.e., if there are n nodes in the network, every node will have a rank of $1/n$. In Fig. 2, each node has a rank of 1/14. As each incoming link increases the rank of a node, we update the rank of each node by adding the importance of the incoming links. We continue this until the rank of all of the nodes stabilizes. We use a standard error tolerance value in the PageRank algorithm, which is 1e-6, to check for convergence in the power iteration process. If the PageRank values across two iterations is within this error tolerance, the rank of nodes is assumed to have stabilized and is returned. In the example network, nodes G and H do not have any outgoing links, and PageRank refers to them as *dangling nodes*.

Dangling nodes would cause the final rank of each node to converge to 0, and the importance of these nodes cannot be propagated further. Since dangling nodes and disconnected components are quite common in the Internet as well as in common circuits, a positive constant between 0 and 1.0 (typically 0.15) is introduced, which is the damping factor ϵ [27]. We add a virtual directed edge from G and H to every other node in the network and assign ϵ to every outgoing edge from G and H.

After adjustment of the dangling nodes, we recalculate the rank of each of the nodes in the graph until the PageRank value stabilizes. For our example, the initial value, intermediate

value, and final value of the PageRank of each node is shown in Fig. 3.

Let $0 < \epsilon < 1$ be a constant source of importance. Let $\mathbf{r}^k$ denote $\mathbf{r}$ in the k-th iteration of the rank computation. Let $\mathbf{A}$ be the adjacency matrix of size $n \times n$ where each $A(a_i, a_j)$ is the ratio between the number of right references to variable i in all assignments to variable j to the number of right references to variable i in all assignments. Let $\mathbf{r}_i^0 = (1/n)$. We compute the importance score of each of the variables according to the following equation:

$$\mathbf{r}^{k+1} = (1-\epsilon)\mathbf{A}\mathbf{r}^k + \frac{\epsilon}{n}. \tag{2}$$

A. Enhancing the Ranking Metric of Selected Signals

PageRank algorithm implicitly adjusts for the in-degree of each node. When the same principle applied on a circuit netlist graph with loop structures among flops, PageRank algorithm tends to select output signals from the IP modules as high-ranking signals. Since output signals are connected to many internal signals, they inherit their importance values from these signals. This gives the PageRank algorithm a *false* sense of importance. For the signal selection application, selecting output signals is not useful, since these are already observable. Our objective in signal selection is to select important internal signals for observation.

To resolve that concern, we enhance the signal-ranking metric of the PageRank algorithm. We calculate a *reverse PageRank* for each of the nodes in the circuit graph. We create a graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ for the original circuit graph $\mathcal{G}$. For each directed edge $e \in \mathcal{E}$ connecting a pair of nodes (v_i, v_j) in the original circuit netlist graph $\mathcal{G}$, we create a directed edge $e' \in \mathcal{E}'$ connecting the same pair of nodes (v_j, v_i) in the graph $\mathcal{G}'$. Then we calculate a PageRank score for each of the graph nodes in $\mathcal{G}'$. We use $p\text{rank}_1(v)$ to denote the PageRank of a node in $\mathcal{G}$, and $p\text{rank}_2(v)$ to denote the PageRank of a node in $\mathcal{G}'$. Intuitively, we see that PageRank algorithm will assign high importance values to the nodes in $\mathcal{G}'$ that have high in-degrees from other important nodes. The in-degree of a node in $\mathcal{G}'$ maps to the out-degree of the same node in the $\mathcal{G}$. To combine $p\text{rank}_1(v)$ and $p\text{rank}_2(v)$, we calculate their harmonic mean (HM) following the idea of the *importance* metric in [23]. We use $p\text{rank}_{\text{hm}}$ to denote this metric

$$p\text{rank}_{\text{hm}}(v) = \frac{1}{\frac{1}{p\text{rank}_1(v)} + \frac{1}{p\text{rank}_2(v)}}.$$

By virtue of HM, $p\text{rank}_{\text{hm}}$ will assign high ranks to the flip-flops that have high values for both $p\text{rank}_1$ and $p\text{rank}_2$. Intuitively, this means that $p\text{rank}_{\text{hm}}$ selects flip-flop nodes that are connected to many other important flip-flop nodes via incoming edges and can propagate their values to many other flip-flops via outgoing edges. This indeed resolves our original concern about PageRank algorithm's selection of output nodes for tracing. For the output nodes of a circuit netlist, the in-degree is very high but the out-degree is very low. On the other hand, for the input nodes of a circuit netlist, the out-degree is very high but the in-degree is zero. Therefore, $p\text{rank}_{\text{hm}}$ will not select either outputs or inputs for tracing. Instead, it prefers important internal nodes of the design. In our running example, PRoN selects flip-flop F and C (see Fig. 3) to trace which are neither outputs nor inputs of the design rather they are internal to the design. Our SRR results for OpenSPARC T2 IP modules given in Section VI-B. Table V supports that conclusion.

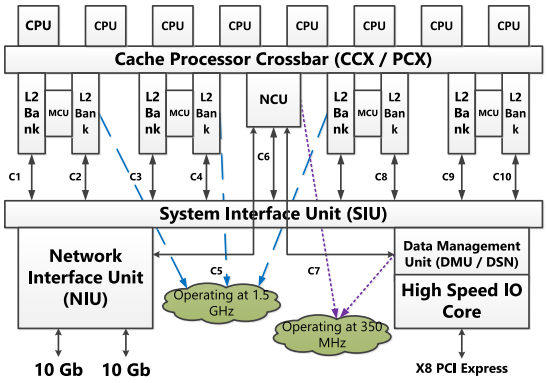

Fig. 4. Block diagram of OpenSPARC T2 processor. NCU: non-cacheable unit and MCU: memory controller unit [4].

Module Name	NoS	LoC
mcu_rdpctl_ctl	29	2872
dmu_dsn	14	3156
dmu_rmu	43	3376
pmu	32	4008
dmu_clu	21	5211
dmu_cmu	15	5260

Module Name	NoS	LoC
ncu_fcd_ctl	85	8427
dmu_ilu	57	10489
mcu_drif_ctl	113	16493
ncu_scd_ctl	262	23882
dmu_imu	239	77230

Fig. 5. Details of each of the OpenSPARC T2 design modules used in our experiment. **NoS**: no. of submodules in the design module excluding standard library cells. **LoC**: total lines of code excluding standard library cells.

B. Functionally Relevant Signal Selection by PRoN

The PRoN algorithm analyzes the structure of the circuit netlist and selects a signal that is important in the design, based on which other important signals that signal is connected to. If a variable is well-connected to other connected variables, it is highly likely that variable forms an important part of the design function. In a well-designed hardware, design structure should be closely related to functionality, for optimal performance of the design implementation. We believe variable importance is a metric that transitions quite faithfully between the structure and the function of a design, thereby capturing how a design structure, does in fact, correspond to the functionality. Our results in Tables VI–VIII support this intuition.

V. Experimental Setup

A. Design Testbed

We primarily use the publicly available USB 2.0 [8], ISCAS89 benchmarks, and multi-core OpenSPARC T2 SoC [4] to demonstrate our results. Comparing the testbeds, we observe that the ISCAS89 benchmarks have no more than 1700 flops, and the USB despite being more complex, synthesizes to around 1800 flops. In contrast, the OpenSPARC T2 is a large, industry scale design with high complexity. We describe

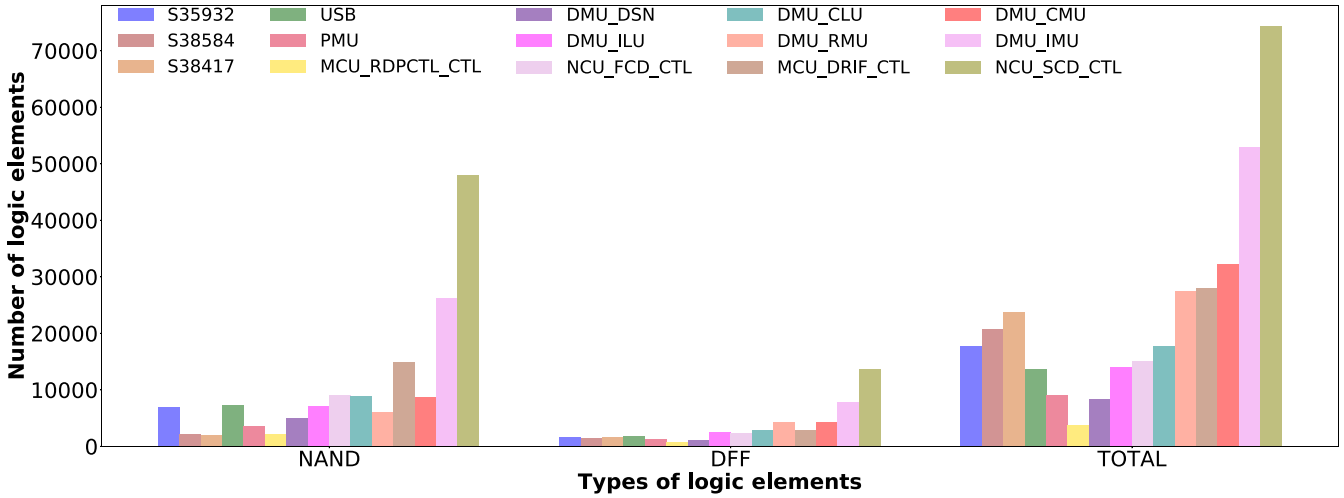

Fig. 6. Comparison of number of different logic elements in ISCAS89 benchmarks, USB 2.0, and OpenSPARC T2 modules.

the experimental setup with respect to the OpenSPARC T2 in the rest of this section.

OpenSPARC T2[1] is a multi-core SoC containing several heterogeneous IPs and many of the complex design features of an industrial SoC design. Fig. 4 shows an IP-level block diagram of OpenSPARC T2. For these experiments, we used several larger and complex IP modules of the OpenSPARC T2 design. Fig. 5 details number of submodules and the total number of line of codes of each of the T2 design modules that we used in our experiment. Details of the ISCAS89 benchmarks, USB 2.0, and several large and complex modules of OpenSPARC T2 in terms of the total number of flip-flops and the logic elements are shown in Fig 6. ISCAS89 benchmarks contain up to 1728 flip-flops and 23 815 total logic elements whereas the OpenSPARC T2 design modules contain up to 13 746 flip-flops and 74 350 total logic elements. The presence of a several orders of magnitude more flip-flops and logic elements in the different design modules of OpenSPARC T2 make these modules functionally complex and larger than the three largest designs of the ISCAS89 benchmark and the USB design.

B. Testbenches

To simulate and collect trace signal values from each of the OpenSPARC T2 design modules, we used our own constrained-random testbenches [6] written in SystemVerilog [7] as per the design specification. We could not use any tests that are included in the OpenSPARC T2 regression suites, since those tests were meant to simulate the whole SoC. We used SystemVerilog monitors during simulation and recorded trace signal values into an output trace file.

C. Tools Used for Comparison

We compared the scalability and quality of the selected trace signals of our PRoN method against those of several other state-of-the-art algorithms. We used SigSeT_1 [10], SigSeT_2 [33], HybrSel [22], and AASR [16]. Since SigSeT_1, SigSeT_2, HybrSel, and PRoN accept designs in ISCAS89 format, we converted the USB and OpenSPARC T2 design modules into ISCAS89 netlist format for comparison among these algorithms. We synthesized the USB and OpenSPARC T2 design modules by using the Synopsys Design Compiler with the NanGate 45-nm library [3], and constrained the library such that the synthesized DC netlist contained only basic logic gates like AND, OR, NOT, NAND, NOR, and D flip-flop (DFF). We then converted the DC netlist into the ISCAS89 format. For AASR [15], we used the GTECH 180 nm library that is included in the Synopsys Design Compiler package, since AASR can only parse design netlists consisting of GTECH library logic elements.

[1]OpenSPARC T2 source: http://www.oracle.com/technetwork/systems/opensparc/opensparc-t2-page-1446157.html.

TABLE II
RUNTIME AND MAXIMUM MEMORY USAGE OF SIGSET_1 [10], HYBRSEL [22], AND PRON DURING THE SIGNAL SELECTION PHASE ON ISCAS89 BENCHMARKS AND USB. THE BENCHMARKS ARE ARRANGED IN INCREASING ORDER OF TOTAL NUMBER OF LOGIC ELEMENTS. **T**: RUNTIME IN SECONDS AND **MEM**: PEAK MEMORY USAGE IN MB

Bench mark	Logic elements		SiGSeT_1		HybrSel		PRoN	
	DFF	Total	T	Mem	T	Mem	T	Mem
USB2.0	1757	13601	181	385	735.36	1204.28	2.01	805.7
s35932	1728	17793	9.52	498	17.6K	389	7.74	275.8
s38584	1452	20705	150	285	8.94K	287	7.86	298.81
s38417	1636	23815	208	702	19.4K	359	9.54	326.9

D. Execution Platform

All experiments on the ISCAS89 designs and USB were run on an AMD Opteron 8-core 22xx processor with 15 GB of RAM. All experiments on the OpenSPARC T2 design modules were run on an Intel Xeon CPU E3-1240 8-core processor running at 3.4 GHz with 16-GB RAM. In most of our experiments, we used simulation-based coverage metrics for behavioral coverage, including line coverage, condition coverage, branch coverage, toggle coverage, FSM coverage, and path coverage.

TABLE III
RUNTIME AND PEAK MEMORY USAGE OF SIGSET_1 [10], HYBRSEL [22], AASR [16], SIGSET_2 [33], AND PRON DURING THE SIGNAL SELECTION PHASE ON OPENSPARC T2 DESIGN MODULES. OPENSPARC T2 DESIGN MODULES ARE ARRANGED IN INCREASING ORDER OF TOTAL NUMBER OF LOGIC ELEMENTS. **DFF**: TOTAL NUMBER OF D FLIP-FLOPS IN A DESIGN MODULE, **TOTAL**: TOTAL NUMBER OF LOGIC ELEMENTS IN A DESIGN MODULE; **T**: RUNTIME IN SECONDS; **MEM**: PEAK MEMORY USAGE IN MB; ⊠: SIGNAL SELECTION FAILURE DUE TO EXCESSIVE PEAK MEMORY USAGE (30 GB OR MORE); ⊖: SIGNAL SELECTION FAILURE DUE TO AN ERROR; ⊡: SIGNAL SELECTION FAILURE DUE TO TIMEOUT (1800 S); AND ⊘: SIGNAL SELECTION FAILURE DUE TO TIMEOUT (7200 S)

Module Name	Logic Elements		SiGSeT_1		HybrSel		AASR		SigSeT_2		PRoN	
	DFF	Total	T	Mem	T	Mem	T	Mem	T	Mem	T	Mem
mcu_rdpctl_ ctl	722	3720	1.13	50.21	107.11	1050.13	⊘	125.85	⊘	113.59	1.01	9.41
pmu	1351	9145	4.07	325.26	320.33	1193.32	⊘	151.8	⊘	113.59	2.01	746.51
dmu_dsn	1108	8409	3.01	351.62	556.20	932.67	⊘	138.22	⊘	113.59	2.01	724.26
dmu_ilu	2517	13948	⊖	⊖	1556.0	1205.31	⊘	174.66	⊘	145.19	2.01	806.0
ncu_fcd_ ctl	2397	15169	20.02	3069.11	771.79	1014.32	⊘	175.12	⊘	152.67	2.01	816.48
dmu_clu	2893	17743	55.06	8271.8	1479.0	1214.27	⊘	180.53	⊘	176.64	3.01	842.3
dmu_rmu	4279	27538	2.02	⊠	⊡	1240.87	⊘	209.87	⊘	297.77	3.01	949.66
mcu_drif_ ctl	2954	27945	272	⊠	⊡	1243.38	⊘	192.58	⊘	251.14	5.01	981.33
dmu_cmu	4246	32234	96.47	⊠	⊡	1240.87	⊘	215.6	⊘	345.49	8.01	1199.83
dmu_imu	7895	52962	245	⊠	⊡	1240.84	⊘	280.9	⊘	462.51	9.01	1249.83
ncu_scd_ ctl	13746	74350	255	⊠	⊡	196.18	⊘	393.64	⊘	645.69	12.01	1389.58

VI. EXPERIMENTAL RESULTS

A. Scalability of Different Signal Selection Algorithms

In this experiment, we show scalability in terms of runtime and peak memory usage of different signal selection algorithms based on SRR including SigSeT_1 [10], SigSeT_2 [33], HybrSel [22], and AASR [16]. We compare these algorithms to our PRoN algorithm.

For this experiment we use the three biggest designs of the ISCAS89 benchmark (namely s3592, s38417, and s38584), and USB 2.0 design, we compare SigSeT_1, HybrSel, and our PRoN algorithm (Table II). On the OpenSPARC T2, we compare many more SRR-based signal selection tools with our PRoN algorithm. The tools under comparison are SigSeT_1, SigSeT_2, AASR, HybrSel, and PRoN. We run experiments on 11 large and complex design modules of the OpenSPARC T2 and compare runtime and peak memory usage [Table III and Fig. 7(a)–(c)].

We consider a trace buffer width of 256 bits and trace buffer depth of 512 cycles. To record the maximum memory usage for ISCAS89 benchmarks and USB, we used the Massif tool in Valgrind [9]. For OpenSPARC T2 design module, we use the `datetime` package of Python to measure runtime. We use a virtual memory monitor written in Python to monitor the peak virtual memory usage of each algorithm during signal selection for OpenSPARC T2 design modules. We iterate PageRank until the values of the ranking matrix are stabilized. For each algorithm we set a timeout limit of 7200 s. HybrSel, and AASR iteratively updates the restorability rate of each state element based on the current signal selection, a computationally intensive approach that is time consuming.

Note that in Table II, PRoN uses considerably large peak memory usage for USB due to large fanouts of most of the logic elements (often more than five) causing high outdegree for many nodes in the $\mathcal{G}$ for USB. PageRank of a node thus propagates to many other connecting nodes requiring large number of iterations to converge. This causes high memory usage for PRoN for USB design.

We make the following observations from Tables II and III, and Fig. 7(a)–(c). From Fig. 7(a), we find that SigSet_1 could not complete signal selection for designs consisting of more than 2800 flip-flops because of its large peak memory usage of 30 GB or more. In Fig. 7(b), we note that HybrSel failed to complete signal selection for any design containing more than 2900 flip-flops when the allowed time limit was varied up to 7200 s in steps of 1800 s. Hence, in Table III, we report the timeout value for HybrSel as 1800 s. Both SigSeT_2 and AASR failed to complete signal selection for any OpenSPARC T2 designs within the allowed time limit of 7200 s. None of the SRR-based signal selection algorithms finished for designs greater than 17 743 logic elements.

In contrast, *our PRoN algorithm was able to complete signal selection within 13 s and with a peak memory usage of up to 1.5 GB for the largest OpenSPARC T2 design module consisting of 13 746 flip-flops*.

HybrSel, AASR, and SigSeT_2 update the rate of restorability of each flip-flop in the design in each iteration based on currently selected signals. For a large number of flip-flops of T2 design modules, this iterative update was computationally intensive and took more time and memory, and often failed to complete signal selection in a reasonable amount of time. PRoN is able to scale because it analyzes topography of the design identifying important variables.

Since two of the tools, namely AASR and SigSeT_2 do not complete signal selection on any of the OpenSPARC T2 design modules, our comparison of selected signals in forthcoming experiments is limited to the two SRR-based tools that completed. Among them, since SigSeT_1 and HybrSel do not complete for any modules larger than the `dmu_clu`, we limit comparisons to the top six modules listed in Table III.

This experiment shows that PRoN signal selection algorithm scales to industry standard large-scale designs compared to the state-of-the-art SRR-based signal selection techniques.

B. Comparison of Algorithms With Respect to Restorability

In this experiment, we compare the restorability (measured by SRR) achieved by algorithms designed to optimize the

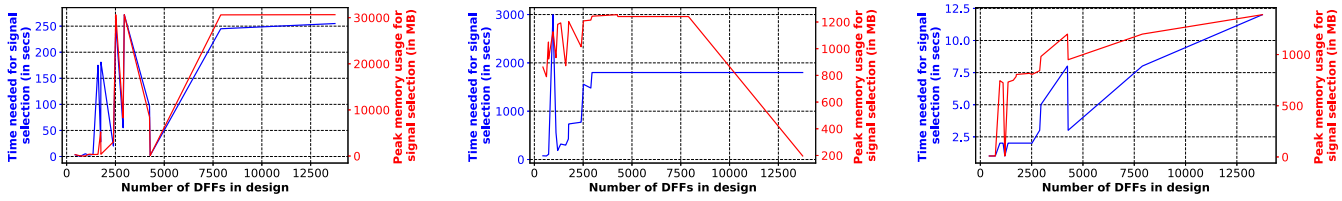

Fig. 7. Scalability of different signal selection algorithms in terms of runtime (in seconds) and peak memory usage (in MB) for 11 different OpenSPARC T2 design modules using (a) SigSeT_1 [10], (b) HybrSel [22], and (c) PRoN.

TABLE IV
COMPARATIVE ANALYSIS OF SRR USING SIGSET_1 [10], HYBRSEL [22], AND PRON ON ISCAS89 BENCHMARKS AND USB 2.0

Benchmark	SiGSeT_1	HybrSel	PRoN
s35932	4.7	4.7	4.7
s38417	4.0	3.8	3.9
s38584	4.7	4.6	4.7
USB2.0	3.7	3.5	3.5

TABLE V
COMPARATIVE ANALYSIS OF SRR FOR SIGNALS SELECTED IN TABLE III ON OPENSPARC T2 DESIGN MODULES. **M1**: PMU. **M2**: MCU_RDPCTL_CTL. **M3**: DMU_DSN. **M4**: DMU_ILU. **M5**: NCU_FCD_CTL. **M6**: DMU_CLU. **RAND**: SRR CALCULATED USING TRACE VALUES OBTAINED FROM DESIGN SIMULATION USING RANDOM STIMULUS; **SIM**: SRR CALCULATED USING TRACE VALUES OBTAINED FROM DESIGN SIMULATION USING CONSTRAINED RANDOM STIMULUS; ⊗: NO SRR VALUES AS SIGSET_1 FAILS TO SELECT SIGNALS FOR DMU_ILU (TABLE III); ⊘: SRR CALCULATION FAILED USING RANDOM STIMULUS FOR NCU_FCD_CTL; §: HIGHEST SRR ACHIEVED USING RANDOM STIMULUS; AND ¶: HIGHEST SRR ACHIEVED USING CONSTRAINED RANDOM STIMULUS

Module Name	SiGSeT_1		HybrSel		PRoN	
	Rand	Sim	Rand	Sim	Rand	Sim
M1	3.83	2.55	14.54§	2.18	8.46	7.62¶
M2	2.97	2.27	4.06	3.32¶	29.65§	1.72
M3	13.61	6.52	14.1	2.92	14.14§	8.03¶
M4	⊗	⊗	33.16§	16.8	29.65	19.55¶
M5	⊘	1.97	⊘	6.71¶	⊘	3.98
M6	37.2	8.98	37.49§	9.83¶	35.99	8.41

SRR metric with our PRoN algorithm, that is not designed to optimize this metric. Since SRR is the de facto standard to measure goodness of selected signals, we evaluate our algorithm according to this metric for the sake of completeness.

To calculate SRR values for the ISCAS89 benchmarks and USB 2.0 design, we simulate the designs using randomized testbenches. We use the top 20% of the signals selected by each method for each benchmark and restore signals for 5000 cycles.

For the OpenSPARC T2 design modules, since we construct SystemVerilog testbenches (see Section V), we could use signal values from simulation traces in addition to randomized signal values to calculate the SRR. We use a trace buffer width of 256 bits and a trace buffer depth of 512 cycles for both simulation-value-based and randomized-value-based SRR calculations.

Tables IV and V show comparative analysis of SRR values for different algorithms on ISCAS89, USB, and different OpenSPARC T2 design modules, respectively.

SRR calculation involves forward propagation and backward justification [10] for selected trace signals. In several cases, signal restoration tool was not able to compute SRR on the selected signals from Table III. In one case, none of the randomized signal values converged, possibly because the number of signals to be restored overflowed and eventually restoration process ran out of memory.

On the ISCAS89 benchmarks, none of the methods outperformed the others. PRoN's SRR value is lower than that of SigSeT_1 and HybrSel. On OpenSPARC T2 design modules, HybrSel and PRoN *consistently performed better* than SigSeT_1. Table V shows that PRoN achieves the highest SRR values for three IP modules while using simulation-based trace values (¶ in column 7), and for two IP modules while using randomized trace values (§ in column 6). This is interesting, given that PRoN is not optimized for SRR.

We note that the SRRs of the ISCAS89 benchmarks in Table IV are significantly lower than the values reported in previous papers [10], [16], [22]. SRR is a ratio and is defined as (total number of signals restored + total number of signals traced)/(total number of signals traced). Previous work [10], [16], [22] used a fixed-length trace buffer of size 8/16/32, and therefore the denominator is 8/16/32. If the average number of signals restored is 1000, the RR value will be 126/63/32. We select approx 350 signals in each design, making our denominator very large. So even with 1200 signals restored, the SRR value is small. In Table V, when trace signal values from simulation are used for restoration compared to randomized values, the SRR values of the T2 benchmarks were 4× smaller. The reason is that randomization assigns a concrete binary value of 1'b1 or 1'b0 to every selected trace signal at each cycle, effectively maximizing the SRR value. In a simulation, that is more reminiscent of the real scenario, there are cycles in which a 1'bX (an unknown value) is assigned to a traced signal. An unknown value does not help to restore any other new signal values, effectively reducing the SRR values. A traced signal may have 1'bX if it is part of a control bus or a data bus. Whenever the control or data bus enable signal is de-asserted, the control bus or the data bus does not have a concrete binary value, causing the trace value of the traced signal to become 1'bX.

Section VI-A concerns the scalability of each of the signal selection methods. AASR and HybrSel simulate the design

TABLE VI
COMPARATIVE ANALYSIS OF TRACE SIGNALS FROM SIGSET_1 [10] AND PRON WITH RESPECT TO SIMULATION-BASED COVERAGE METRICS FOR DIFFERENT USB MODULES. **U0**: USBF_UTMI_IF. **U1**: USBF_PL, **U2**: USBF_MEM_ARB. **U4**: USBF_RF. **U5**: USBF_WB. **L**: LINE COVERAGE, **C**: CONDITION COVERAGE, **F**: FSM COVERAGE, **B**: BRANCH COVERAGE, **O**: OVERALL COVERAGE. ⊘: VCS DOES NOT REPORT THE COVERAGE VALUE

Module Name	SigSeT_1					PRoN				
	O	L	C	F	B	O	L	C	F	B
u0	29.73	35.81	28.89	15.75	38.35	49.61	71.81	38.89	28.75	58.97
u1	17.71	22.94	15.02	7.44	25.43	33.94	52.94	22.02	15.33	45.46
u2	41.44	33.45	⊘	⊘	49.43	68.75	75.00	⊘	⊘	62.50
u4	17.71	30.98	1.67	⊘	20.48	33.70	60.98	2.45	⊘	37.68
u5	23.27	31.90	13.49	15.37	32.33	40.14	56.34	19.23	31.67	53.33

TABLE VII
COMPARATIVE ANALYSIS OF TRACE SIGNALS FROM SIGSET_1 [10], AND PRON WITH RESPECT TO SIMULATION-BASED COVERAGE METRICS FOR DIFFERENT OPENSPARC T2 DESIGN MODULES. **M1**: PMU. **M2**: MCU_RDPCTL_CTL. **M3**: DMU_DSN. **M4**: DMU_ILU. **M5**: NCU_FCD_CTL. **M6**: DMU_CLU. **L**: LINE COVERAGE. **C**: CONDITION COVERAGE. **F**: FSM COVERAGE. **B**: BRANCH COVERAGE. **T**: TOGGLE COVERAGE. **P**: PATH COVERAGE. **O**: OVERALL COVERAGE. ⊘: VCS DOES NOT REPORT THE COVERAGE VALUE. ⊖: COVERAGE CALCULATION WAS NOT POSSIBLE SINCE SIGSET_1 FAILS TO SELECT SIGNALS (TABLE III). §: HIGHLIGHTING PATH COVERAGE

Module Name	SigSeT_1							PRoN						
	O	L	C	F	B	T	P	O	L	C	F	B	T	P
M1	17.01	32.44	⊘	⊘	⊘	1.58	⊘	18.93	33.45	⊘	⊘	⊘	1.42	⊘
M2	37.23	65.49	37.50	⊘	38.36	2.59	⊘	37.69	67.21	38.64	⊘	39.36	2.57	⊘
M3	28.29	64.15	31.11	0.0	41.85	8.42	20.20§	28.73	60.95	32.89	0.0	42.49	5.07	23.00§
M4	⊖	⊖	⊖	⊖	⊖	⊖	⊖	30.12	61.24	50.94	0.0	42.72	1.98	23.86§
M5	20.32	31.76	25.69	⊘	23.58	0.23	⊘	16.25	28.91	26.07	⊘	19.58	1.46	⊘
M6	20.38	49.73	11.27	0.0	34.04	1.89	25.38§	25.35	53.86	26.76	0.0	38.65	4.40	28.44§

TABLE VIII
COMPARATIVE ANALYSIS OF TRACE SIGNALS FROM HYBRSEL [22], AND PRON WITH RESPECT TO SIMULATION-BASED COVERAGE METRICS FOR DIFFERENT OPENSPARC T2 DESIGN MODULES. **M1**: PMU. **M2**: MCU_RDPCTL_CTL. **M3**: DMU_DSN. **M4**: DMU_ILU. **M5**: NCU_FCD_CTL. **M6**: DMU_CLU. **L**: LINE COVERAGE. **C**: CONDITION COVERAGE. **F**: FSM COVERAGE. **B**: BRANCH COVERAGE. **T**: TOGGLE COVERAGE. **P**: PATH COVERAGE. **O**: OVERALL COVERAGE. ⊘: VCS DOES NOT REPORT THE COVERAGE VALUE. ¶: HIGHLIGHTING PATH COVERAGE

Module Name	HybrSel							PRoN						
	O	L	C	F	B	T	P	O	L	C	F	B	T	P
M1	18.24	42.18	⊘	⊘	⊘	0.30	⊘	18.93	33.45	⊘	⊘	⊘	1.42	⊘
M2	29.48	59.34	28.41	⊘	27.40	2.77	⊘	37.69	67.21	38.64	⊘	39.36	2.57	⊘
M3	25.76	58.97	22.78	6.67	39.44	9.89	16.79¶	28.73	60.95	32.89	0.0	42.49	5.07	23.00¶
M4	26.59	55.45	42.95	0.0	39.44	5.07	16.89¶	30.12	61.24	50.94	0.0	42.72	1.98	23.86¶
M5	3.13	8.60	1.45	⊘	2.12	0.33	⊘	16.25	28.91	26.07	⊘	19.58	1.46	⊘
M6	18.93	47.99	8.45	0.0	32.50	0.80	23.85¶	25.35	53.86	26.76	0.0	38.65	4.40	28.44¶

netlist in each iteration for the specified number of cycles during signal selection to find out the best signals to trace. For these big designs, with a larger number of specified cycles, AASR and HybrSel take much longer time to complete signal selection and often times out. Hence, in order to have a fair comparison of runtime and peak memory usage for signal selection of all the methods on a reasonable number of designs, we used a trace buffer depth of 512 cycles. We found that for any trace buffer depth value of greater than 512 cycles, even HybrSel can only complete signal selection for no more than four designs, thereby reducing the value of this experiment.

This section concerns the signal restoration post tracing using traced signal values. We restore signals with traced signal values using a combination of forward propagation and backward justification. We can afford to restore up to 5000 cycles in this phase, since there is no iterative calculation, unlike in the signal selection phase.

This experiment demonstrates that signals selected by PRoN although not optimized for SRR, often achieve higher restorability.

C. Comparing Behavioral Coverage of Selected Signals

In this experiment we study the behavioral coverage achieved by the selected signals using different tools. In pre-silicon simulation, behavioral coverage metrics are intended to check for important high-level behavioral and functional coverage of the design.

In this experiment, we use USB 2.0 and OpenSPARC T2 design modules. For USB design, we trace values of 355 flip-flops for a simulation duration of 175 ms. Such a long trace is needed since at least 100 ms of simulation is required to activate different important states (such as the high-speed state mode of USB) of the USB line control module. We use the traced value of the selected signals along with five important input control signals as the stimulus in RTL and measure the behavioral coverage by using Synopsys VCS. The behavioral coverage consists of four components, namely *branch coverage*, *line coverage*, *condition coverage*, and *FSM coverage*. Table VI shows the behavioral coverage values reported by VCS. For each of the methods, we do not report the FSM coverage for `u4 (usbf_rf)`, since it did not contain

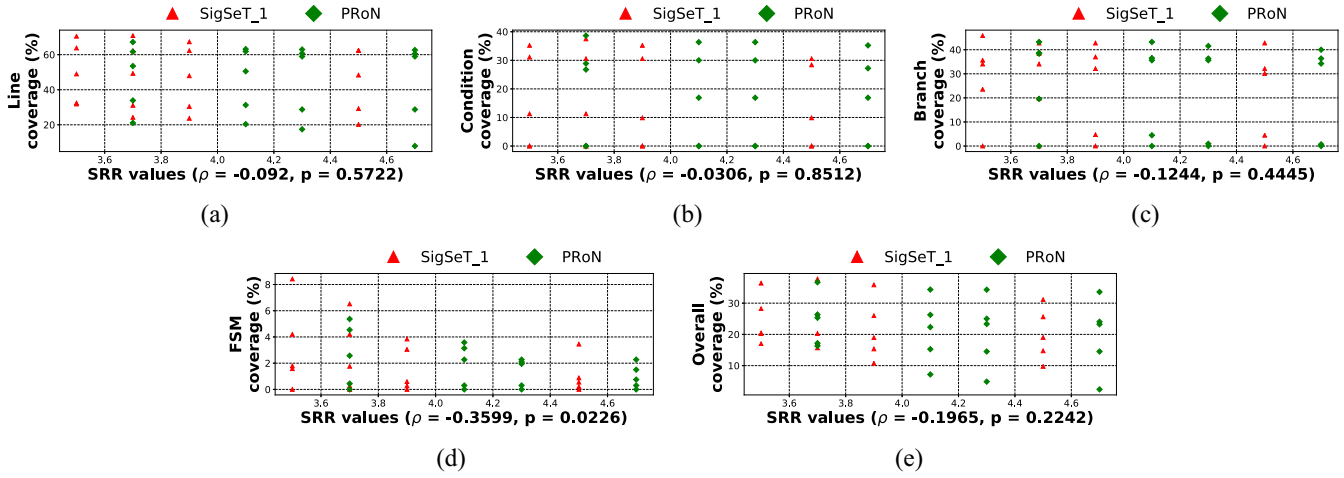

Fig. 8. Graphs showing lack of correlation between SRR and for (a) line coverage, (b) condition coverage, (c) branch coverage, (d) FSM coverage, and (e) overall coverage on different USB modules $u_0, \ldots, u_5$ for the signals selected by SigSeT_1 [10], and PRoN. ρ: correlation co-efficient between SRR and the coverage metric. **p**: p-value indicating rejection probability for the null hypothesis of an uncorrelated system producing datasets that have ρ as extreme as the one computed from observed datasets.

any state machines. Also, we do not report the FSM and conditional coverage for `u2 (usbf_mem_arb)`, as it is a combinational design module. For OpenSPARC T2 design modules, we traced values of 256 flip-flops for 512 cycles. We used the traced values to measure the behavioral coverage by using Synopsys VCS. The behavioral coverage consists of six components namely *line coverage*, *condition coverage*, *branch coverage*, *FSM coverage*, *toggle coverage*, and *path coverage*.[2] Tables VII and VIII show the behavioral coverage values reported by VCS. For SigSeT_1, VCS was able to calculate *path coverage* for two different design modules whereas for HybrSel and PRoN, VCS was able to calculate *path coverage* for three different design modules by using traced signal values. The path coverage values are highlighted with § in Table VII and with ¶ in Table VIII.

For the USB design, the behavioral coverage of signals selected by PRoN is up to 42% (with an average of 19.6%) greater than that of the signals selected by SigSeT_1. This experiment shows that compared to SigSeT_1, PRoN selected more functionally relevant signals from the USB design.

For the OpenSPARC T2 design modules, the overall behavioral coverage of signals selected by PRoN is up to 30.12% (with an average of 5.64%) greater than that of the signals selected by SigSeT_1 and up to 13.12% (with an average of 5.83%) greater than that of the signals selected by HybrSel. For OpenSPARC T2 *we do not report FSM coverage for several design modules, since those modules do not contain any explicit state machines*.

1) *Line Coverage:* Line coverage (Section II-D) of signals selected by PRoN is up to 61.25% (with an average of 10.34%) greater than that of the signals selected by SigSeT_1 and up to 20.31% (with an average of 5.52%) greater than that of the signals selected by HybrSel.
2) *Branch Coverage:* Branch coverage (Section II-D) of signals selected by PRoN is up to 17.46% (with an average of 8.38%) greater than that of the signals selected by SigSeT_1 and up to 42.72% (with an average of 7.4%) greater than that of the signals selected by HybrSel.
3) *Condition Coverage:* Condition coverage (Section II-D) of signals selected by PRoN is up to 24.62% (with an average of 14.25%) greater than that of the signals selected by SigSeT_1 and up to 50.94% (with an average of 11.62%) greater than that of the signals selected by HybrSel.
4) *Path Coverage:* For SigSeT_1, the path coverage (Section II-D) was up to 25.38% (with an average of 22.79%); for HybrSel the path coverage was up to 23.85% (with an average of 20.32%); and for PRoN, the path coverage was up to 28.44% (with an average of 25.1%). For large designs, like OpenSPARC T2 design modules, even a small increment in path coverage manifests in the execution of a large number of additional design paths. In our analysis, signals selected by PRoN achieved up to 4.59% (with an average of 3.33%) more path coverage than SigSeT_1 and HybrSel, implying that signals selected by PRoN executed a larger number of additional design paths compared to the signals selected by SigSeT_1 and HybrSel. This experiment shows that compared to SigSeT_1 and HybrSel, PRoN selects functionally superior signals for tracing from OpenSPARC T2 design modules. This result supports our modification to the PageRank metric to select important internal signals as demonstrated in Section IV-A.
5) *Toggle Coverage:* Toggle coverage (Section II-D) of signals selected by SigSeT_1 is up to 3.35% greater than that of PRoN but on average, toggle coverage of the signals selected by PRoN is 0.37% greater than the signals selected by SigSeT_1. Toggle coverage of signals selected by HybrSel is up to 4.82% (with an average of 2.26%) greater than that of the signals selected by PRoN.

This experiment shows that signals selected by PRoN achieve higher behavioral coverage on industry

[2]We enabled path coverage in VCS by using the -lca option.

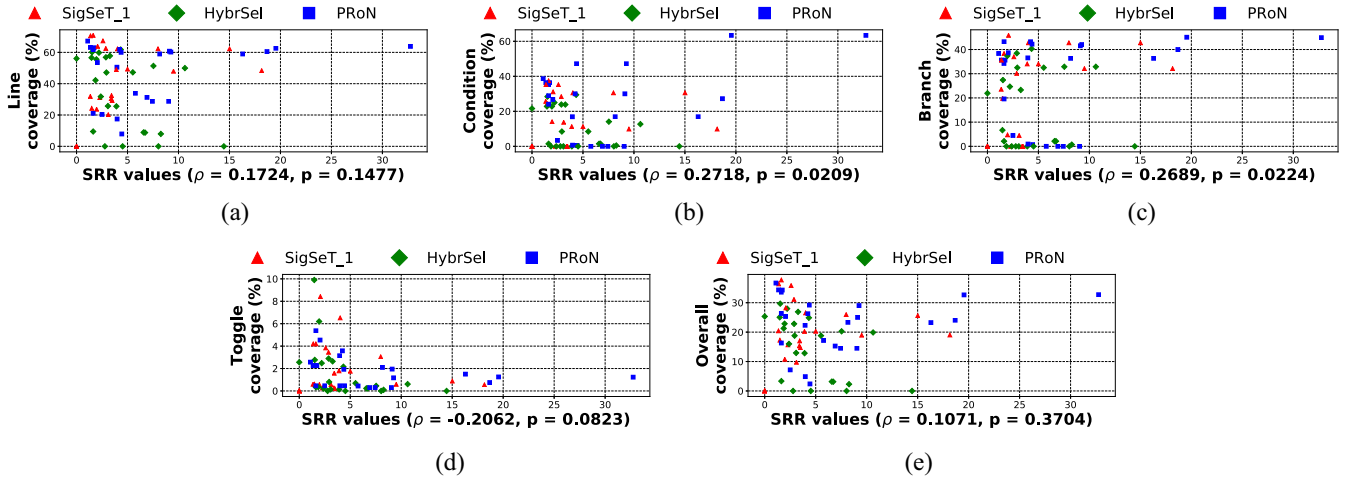

Fig. 9. Graphs showing lack of correlation between SRR and for (a) line coverage, (b) condition coverage, (c) branch coverage, (d) toggle coverage, and (e) overall coverage on different OpenSPARC T2 modules $M_1, \ldots, M_6$ for the signals selected by SigSeT_1 [10], HybrSel [22], and PRoN. **M1**: pmu. **M2**: mcu_rdpctl_ctl. **M3**: dmu_dsn. **M4**: dmu_ilu. **M5**: ncu_fcd_ctl. **M6**: dmu_clu. ρ: correlation co-efficient between SRR and the coverage metric. **p**: p-value indicating rejection probability for the null hypothesis of an uncorrelated system producing datasets that have ρ as extreme as the one computed from observed datasets.

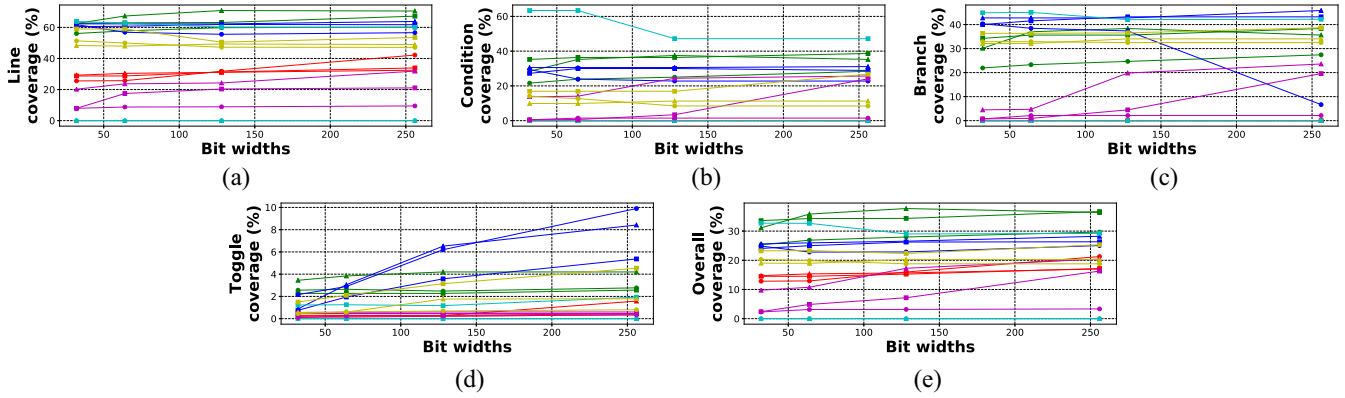

Fig. 10. Graphs showing change in (a) line coverage, (b) condition coverage, (c) branch coverage, (d) toggle coverage, and (e) overall coverage and with different configuration of trace buffer width on different OpenSPARC T2 modules $M_1, \ldots, M_6$ for the signals selected by SigSeT_1 [10], HybrSel [22], and PRoN. **M1**: pmu (■). **M2**: mcu_rdpctl_ctl (■). **M3**: dmu_dsn (■). **M4**: dmu_ilu (■). **M5**: ncu_fcd_ctl (■). **M6**: dmu_clu (■). **SigSeT_1**: ▲, **HybrSel**: ●, **PRoN**: ■.

TABLE IX
COMPARATIVE ANALYSIS OF TRACE SIGNALS FROM TRADITIONAL PAGERANK ALGORITHM [26], AND PRON WITH RESPECT TO SIMULATION-BASED COVERAGE METRICS FOR DIFFERENT OPENSPARC T2 DESIGN MODULES. **M1**: PMU. **M2**: MCU_RDPCTL_CTL. **M3**: DMU_DSN. **M4**: DMU_ILU. **M5**: NCU_FCD_CTL. **M6**: DMU_CLU. **L**: LINE COVERAGE. **C**: CONDITION COVERAGE. **F**: FSM COVERAGE. **B**: BRANCH COVERAGE. **T**: TOGGLE COVERAGE. **P**: PATH COVERAGE. **O**: OVERALL COVERAGE. ⊘: VCS DOES NOT REPORT THE COVERAGE VALUE. ¶: HIGHLIGHTING PATH COVERAGE

Module Name	Traditional PageRank algorithm							PRoN						
	O	L	C	F	B	T	P	O	L	C	F	B	T	P
M1	12.59	24.87	⊘	⊘	⊘	0.30	⊘	18.93	33.45	⊘	⊘	⊘	1.42	⊘
M2	28.73	59.34	25.00	⊘	28.77	1.81	⊘	37.69	67.21	38.64	⊘	39.36	2.57	⊘
M3	23.34	60.31	25.00	0.0	38.77	0.47	15.49¶	28.73	60.95	32.89	0.0	42.49	5.07	23.00¶
M4	25.94	55.47	44.46	0.0	32.11	1.17	18.45¶	30.12	61.24	50.94	0.0	42.72	1.98	23.86¶
M5	2.94	9.42	0.81	⊘	0.94	0.61	⊘	16.25	28.91	26.07	⊘	19.58	1.46	⊘
M6	21.66	59.27	11.27	0.0	34.42	0.84	24.16¶	25.35	53.86	26.76	0.0	38.65	4.40	28.44¶

standard large-scale designs outperforming the signals selected by the state-of-the-art SRR-based techniques.

D. Correlation Analysis Between SRR and High-Level Behavioral Coverage Metrics

This experiment finds if there is a correlation between high SRR values and the behavioral coverage metrics from pre-silicon, in order to determine the extent of high-level functional coverage of SRR. For each of the USB design modules, we traced top 5%, 10%, 15%, and 20% flip-flops per tool and for each of the OpenSPARC T2 design modules, we traced top 32, 64, 128, and 256 flip-flops per tool. We used the trace signal values and the design netlist to calculate the SRR value via backward justification and forward propagation [11]. Also, we used the traced signal values and the instrumented Verilog

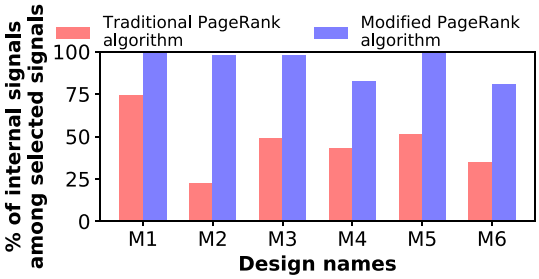

Fig. 11. Comparison between signals selected by the traditional PageRank algorithm [26] and the modified PageRank algorithm of current work for various OpenSPARC T2 design modules. **M1**: pmu. **M2**: mcu_rdpctl_ctl. **M3**: dmu_dsn. **M4**: dmu_ilu. **M5**: ncu_fcd_ctl. **M6**: dmu_clu.

code of each of the design module and calculated different coverage metrics using Synopsys VCS. We use scatter plots to analyze correlation between SRR and the coverage metric values for each group of traced flip-flops for each of the design modules.

In Fig. 8(a)–(e), we analyze the correlation between SRR and the different components of behavioral coverage for USB design modules. In Fig. 9(a)–(e), we analyze the correlation between SRR and the different components of behavioral coverage for OpenSPARC T2 design modules. For each such scatter plot, we have calculated the Pearson rank correlation coefficient ρ and have shown it below the scatter plot.

This experiment shows that there is no correlation between the SRR value and behavioral coverage. This underscores the point that a high SRR has low to no correlation with functional behavior.

E. Sensitivity Analysis Between Behavioral Coverage and Trace Buffer Width and Depth

This experiment finds the sensitivity of the behavioral coverage metrics from pre-silicon with the different configurations of the trace buffer width with a fixed trace buffer depth. For each of the OpenSPARC T2 design modules, we traced top 32, 64, 128, and 256 flip-flops per tool for 512 cycles. We use line plots to analyze the sensitivity between each of the coverage metrics and the different trace buffer width.

In Fig. 10(a)–(e), we analyze the sensitivity between different components of the behavioral coverage and different trace buffer width for OpenSPARC T2 design modules.

This experiment shows that the behavioral coverage increases with the increasing width of the trace buffer. This underscores the point that post-silicon observability is positively sensitive to the trace buffer width.

F. Comparison of the Signals Selected by Traditional PageRank Algorithm [26] and PRoN (Modified PageRank Algorithm)

This experiment demonstrates the improvement in signal selection of PRoN, the modified version of the PageRank algorithm over the traditional PageRank algorithm applied in [26]. We compare the improvement in terms of: 1) the percentage of the selected internal design signals and 2) the improvement in the behavioral coverage metrics of the selected signals.

TABLE X
HIGH-LEVEL FUNCTIONALITY COVERED BY PRON AND SIGSET_1 SELECTED SIGNALS ON USB NETLIST. **P:** PARTIAL BIT SELECTED

Signal Name	Module Name	Signal Functionality	Sig SeT_1	PRoN
no_bufs0	usbf_pe	**A.** Indicates available buffer size is less than payload size to switch to other buffer, **B.** BUF0 is full in DMA mode (Only BUF0 is used in DMA mode), **C.** Indicates if the BUF1 needs to be selected for next operation by the functional controller	✗	✓
token_pid _sel	usbf_pe	Handshaking signals indicating the packet accepting capacity of the buffer	✗	✓
dma_out _buf_avail	usbf_ep_rf	Indicates that there is a space for at least one MAX_PL_SZ packet in the buffer	✗	✓
inta, intb	usbf_rf	A fully programmable interrupt to provide full flexibility to software, the interrupts may be endpoint dependent or independent, indicating an error condition or overall events that have global meaning	✗	✓
state	usbf_pe	Indicates the states of operation of the USB protocol engine	✓	✓
state	usbf_utmi _ls	Indicates the states of operation of the USB protocol engine	P	✓
abort	usbf_pe	Indicates to abort an ongoing data transfer if the following conditions happen **A.** Buffer overflows (Received data packet size is too big and Rx_Data_Valid is asserted), **B.** Register end points matched and protocol engine is not in `IDLE` mode, **C.** Received packet size is more than MAX_PL_SZ	✗	✓
chirp _count	usbf_utmi _ls	A counter to initiate USB high speed mode	✗	✓
pid_seq _err	usbf_pe	An interrupt notifying USB function controller a loss of sync due to bad packets resulting in CRCs	✗	✓

For this experiment, we choose a trace buffer width of 256 bits for each of the OpenSPARC T2 design modules per method.

1) Comparison in Terms of Selected Internal Design Signals: In Fig. 11, we analyze the percentage of internal design signals among the signals that are selected for tracing by two methods. In traditional PageRank algorithm, up to 77.34% of selected signals are design output signals (with an average of 54.10%) whereas for the PRoN no more than 19.14% of selected signals are design output signals (with an average of 6.77%). While for traditional PageRank algorithm up to 74.60% of selected signals are internal design signals (with an average of 45.89%), for PRoN up to 100% of selected signals are internal design signals (with an average of 93.23%). Further analysis shows that the output signals that are selected by PRoN are connected to highly important internal design signals in a feedback loop making those signals relevant for tracing. PRoN selects up to 77.35% more internal design signals (with an average of 41.80%) for tracing compared to the traditional PageRank algorithm of [26].

This experiment shows that for tracing, the modified PageRank algorithm selects significantly more internal design signals compared to the traditional PageRank algorithm method of [26], thereby increasing post-silicon observability.

2) Comparison in Terms of Behavioral Coverage: In this experiment, we study the behavioral coverage achieved by the selected signals using traditional PageRank algorithm [26] and PRoN. Our experimental setup and the behavioral coverage metrics that are used for this comparison, are similar to that of Section VI-C.

In Table IX, we compare the behavioral coverage of the signals that are selected by the two methods. Our analysis shows that the signals selected by PRoN achieves up to 13.31% (average 6.97%) more overall behavioral coverage compared to the signals selected by the traditional PageRank algorithm. For large designs like OpenSPARC T2 design modules, even a small increment in path coverage manifests in the execution of a large number of additional design paths. In our experiment, signals selected by the PRoN achieves *up to* 7.51% *(average* 5.73%*) path coverage* compared to the traditional PageRank algorithm implying that signals selected by the PRoN executed a large number of additional design paths compared to the signals selected by the traditional PageRank algorithm.

This experiment shows that signals selected by PRoN achieve superior behavioral coverage on industry standard large-scale designs outperforming the signals selected by the traditional PageRank algorithm.

G. High-Level Functionality Selected by PRoN on USB Netlist

To give a flavor of the type of high-level functionality captured by the signals selected, we provide a qualitative analysis of two algorithms, PRoN and SigSet_1. In Table X, for each signal, we list the corresponding RTL module and its high-level functionality. PRoN selects all the FSM state registers of the USB protocol engine (`usbf_pe`) and the USB line state module (`usbf_utmi_ls`) and other important signals. On the other hand, SigSet_1 selects only one signal completely and the other partially.

VII. Related Work

Automatic selection of trace buffer signals to maximize the SRR has been studied and many methods have been proposed such as partial-restoration-based [20], [25], complete-restoration-based [10], [12], simulation-based [16], hybrid analysis-based [22], and machine-learning-based [31], [33]. Hardware tracing combining scan chain and trace buffers were studied in [11], [21], and [32]. Hardware tracing using graph centrality measure was studied in [18]. The proposed work improves over above mentioned works in terms of scalability, functional relevance of the selected signals to comprehend post-silicon execution, and behavioral coverage of the selected signals.

Many existing post-silicon debug technique can benefit from PRoN and its improved functionally relevant signals. These include trace analysis-based SoC protocol debug [14], [38], post-silicon validation and trace signal selection via bit-flip detection [34], [35], post-silicon bug localization, and validation for processors [24], [28], [29], [36].

VIII. Conclusion

In light of our experimental findings, the use of SRR as a signal selection metric is not advisable. Instead, an alternate metric needs to be proposed for hardware signal tracing. In [26], we proposed *assertion coverage* as one such metric. This comprises the number of assertions that can be evaluated using the traced signal values. This metric certainly captures high-level behavioral intent since it uses assertions. It may be noted that it depends heavily on the quality of assertions, increasing the subjectivity of the approach. There is a need to define and characterize a metric for signal selection that reflects high-level functionality better than SRR. In future work, we will evaluate assertion coverage and develop such metrics.

In conclusion, we have shown that the SRR as a metric does not reflect the behavioral coverage of the design relevant to practical post-silicon debug. Unsurprisingly, we found no study reporting on the usage of SRR-based methods on industry scale design; all reported applications have been on small benchmarks (e.g., ISCAS89) that are not representative of the complexities of an industrial integrated circuit. The current and future needs of industry are better served if more representative metrics are used for signal selection. We present a signal selection method based on analyzing structural connectivity of the circuit netlist which in turn selects functionally relevant signals by computing variable importance. We demonstrate experiments at a scale and complexity that has hitherto never been used in hardware signal tracing literature. Our algorithm can scale to very large designs with moderate usage of computing resources, and selects high quality signals that closely reflect high-level behavioral functionality.

Acknowledgment

The authors would like to thank Prof. S. Ray of the University of Florida, Gainesville, for many insightful discussions. They would also like to thank R. Jiang for implementation.

References

[1] *Intel Core I7 Processors*. Accessed: Mar. 16, 2018. [Online]. Available: https://www.intel.com/content/www/us/en/processors/core/8th-gen-processor-family-s-platform-datasheet-vol-1.html

[2] *LC3B Processor*. Accessed: Oct. 31, 2015. [Online]. Available: https://courses.engr.illinois.edu/ece411/mp/LC3b_ISA.pdf

[3] *Nangate FreePDK*. Accessed: Oct. 31, 2015. [Online]. Available: http://www.nangate.com/?page_id=2325

[4] *OpenSPARC T2*. Accessed: Mar. 16, 2018. [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/opensparc-t2-page-1446157.html

[5] *PageRank Algorithm*. Accessed: Mar. 16, 2018. [Online]. Available: http://www.ccs.northeastern.edu/home/daikeshi/notes/PageRank.pdf

[6] *PRoN Framework*. Accessed: Mar. 16, 2018. [Online]. Available: https://sites.google.com/view/dpal2/tools

[7] *System Verilog LRM*. Accessed: Mar. 16, 2018. [Online]. Available: http://standards.ieee.org/getieee/1800/download/1800-2012.pdf

[8] *USB 2.0*. Accessed: Oct. 31, 2015. [Online]. Available: http://opencores.org/project,usb

[9] *Valgrind Massif Tool*. Accessed: Oct. 31, 2015. [Online]. Available: http://valgrind.org/docs/manual/ms-manual.html

[10] K. Basu and P. Mishra, "Efficient trace signal selection for post silicon validation and debug," in *Proc. 24th Int. Conf. VLSI Design (VLSI Design)*, 2011, pp. 352–357.

[11] K. Basu, P. Mishra, and P. Patra, "Efficient combination of trace and scan signals for post silicon validation and debug," in *Proc. IEEE Int. Test Conf. (ITC)*, Anaheim, CA, USA, Sep. 2011, pp. 1–8.
[12] K. Basu, P. Mishra, P. Patra, A. Nahir, and A. Adir, "Dynamic selection of trace signals for post-silicon debug," in *Proc. 14th Int. Workshop Microprocessor Test Verification (MTV)*, Dec. 2013, pp. 62–67.
[13] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Comput. Netw.*, vol. 30, nos. 1–7, pp. 107–117, 1998.
[14] Y. Cao, H. Zheng, H. M. Palombo, S. Ray, and J. Yang, "A post-silicon trace analysis approach for system-on-chip protocol debug," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)* Boston, MA, USA, Nov. 2017, pp. 177–184.
[15] A. Chatterjee, S. Deyati, B. Muldrey, S. Devarakond, and A. Banerjee, "Validation signature testing: A methodology for post-silicon validation of analog/mixed-signal circuits," in *Proc. Int. Conf. Comput.-Aided Design*, 2012, pp. 553–556.
[16] D. Chatterjee, C. McCarter, and V. Bertacco, "Simulation-based signal selection for state restoration in silicon debug," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2011, pp. 595–601.
[17] K. Han, J.-S. Yang, and J. A. Abraham, "Enhanced algorithm of combining trace and scan signals in post-silicon validation," in *Proc. IEEE 31st VLSI Test Symp. (VTS)*, 2013, pp. 1–6.
[18] E. Hung and S. J. E. Wilton, "Scalable signal selection for post-silicon debug," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1103–1115, Jun. 2013.
[19] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 2, pp. 285–297, Feb. 2009.
[20] H. F. Ko and N. Nicolici, "Automated trace signals selection using the RTL descriptions," in *Proc. IEEE Int. Test Conf. (ITC)*, 2010, pp. 1–10.
[21] H. F. Ko and N. Nicolici, "Combining scan and trace buffers for enhancing real-time observability in post-silicon debugging," in *Proc. Eur. Test Symp.*, 2010, pp. 62–67.
[22] M. Li and A. Davoodi, "A hybrid approach for fast and accurate trace signal selection for post-silicon debug," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 7, pp. 1081–1094, Jul. 2014.
[23] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Chicago, IL, USA, Jun. 2005, pp. 15–26.
[24] D. Lin *et al.*, "Effective post-silicon validation of system-on-chips using quick error detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 10, pp. 1573–1590, Oct. 2014.
[25] X. Liu and Q. Xu, "On signal selection for visibility enhancement in trace-based post-silicon validation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 8, pp. 1263–1274, Aug. 2012.
[26] S. Ma, D. Pal, R. Jiang, S. Ray, and S. Vasudevan, "Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design, (ICCAD)*, Austin, TX, USA, Nov. 2015, pp. 1–8.
[27] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the Web," Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Rep., 1998. [Online]. Available: http://ilpubs.stanford.edu:8090/422/
[28] S.-B. Park, A. Bracy, H. Wang, and S. Mitra, "BLoG: Post-silicon bug localization in processors using bug localization graphs," in *Proc. ACM 47th Design Autom. Conf.*, 2010, pp. 368–373.
[29] S.-B. Park, T. Hong, and S. Mitra, "Post-silicon bug localization in processors using instruction footprint recording and analysis (IFRA)," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1545–1558, Oct. 2009.
[30] P. Patra, "On the cusp of a validation wall," *IEEE Design Test Comput.*, vol. 24, no. 2, pp. 193–196, Mar./Apr. 2007.
[31] K. Rahmani, P. Mishra, and S. Ray, "Efficient trace signal selection using augmentation and ILP techniques," in *Proc. 15th Int. Symp. Qual. Electron. Design (ISQED)*, Santa Clara, CA, USA, Mar. 2014, pp. 148–155.
[32] K. Rahmani, S. Proch, and P. Mishra, "Efficient selection of trace and scan signals for post-silicon debug," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 313–323, Jan. 2016.
[33] K. Rahmani, S. Ray, and P. Mishra, "Postsilicon trace signal selection using machine learning techniques," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 570–580, Feb. 2017.
[34] P. Taatizadeh and N. Nicolici, "Automated selection of assertions for bit-flip detection during post-silicon validation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 2118–2130, Mar. 2016.
[35] A. Vali and N. Nicolici, "Bit-flip detection-driven selection of trace signals," in *Proc. 21st IEEE Eur. Test Symp. (ETS)*, Amsterdam, The Netherlands, May 2016, pp. 1–6.
[36] I. Wagner and V. Bertacco, "Reversi: Post-silicon validation system for modern microprocessors," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2008, pp. 307–314.
[37] S. Yerramilli, "Addressing post-silicon validation challenge: Leverage validation and test synergy," in *Proc. Int. Test Conf. Keynote*, 2006, pp. 12–17.
[38] H. Zheng, Y. Cao, S. Ray, and J. Yang, "Protocol-guided analysis of post-silicon traces under limited observability," in *Proc. 17th Int. Symp. Qual. Electron. Design (ISQED)*, Santa Clara, CA, USA, Mar. 2016, pp. 301–306.

Debjit Pal (S'18) received the B.Tech. degree from Jadavpur University, Kolkata, India, and the M.S. degree from the Indian Institute of Technology Kharagpur, Kharagpur, India. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, University of Illinois at Urbana–Champaign, Urbana, IL, USA.

His current research interests include post-silicon validation and debug, pre-silicon validation and debug, system verification, and assertion-based verification.

Mr. Pal was a recipient of the best paper nomination in ICCAD 2015 and DAC 2018, and the IEEE CEDA System Validation and Debug Technology Committee Student Research Award in 2016.

Sai Ma received the B.S. and M.S. degrees from the Electrical and Computer Engineering Department, University of Illinois at Urbana–Champaign, Urbana, IL, USA, with a focus on trace buffer signal selection for post-silicon debugging.

Ms. Ma was a recipient of the Best Paper Award of DAC 2014 and was nominated for Best Paper Award of ICCAD 2015.

Shobha Vasudevan received the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, TX, USA.

She is currently an Associate Professor with the Electrical and Computer Engineering Department, University of Illinois at Urbana–Champaign, Urbana, IL, USA. Her current research interests include system verification, analog and digital validation, formal and static analysis, machine learning, and causal inferencing in big data.

Dr. Vasudevan was a recipient of the Best Paper Award of VLSI 2014 and DAC 2014, the NSF Career Award, the ACM SIGDA Outstanding New Faculty Award, the IEEE CEDA Early Career Award, the IBM Faculty Partnership Award in 2017, and several best paper nominations.