Trace Signal Selection to Enhance Timing and Logic Visibility in Post-Silicon Validation

Hamid Shojaei, and Azadeh Davoodi
University of Wisconsin
1415 Engineering Drive, Madison WI 53706
Email: {shojaei, adavoodi}@wisc.edu

***Abstract*—Trace buffer technology allows tracking the values of a few number of state elements inside a chip within a desired time window, which is used to analyze logic errors during post-silicon validation. Due to limitation in the bandwidth of trace buffers, only few state elements can be selected for tracing. In this work we first propose two improvements to existing "signal selection" algorithms to further increase the logic restorability inside the chip. In addition, we observe that different selections of trace signals can result in the same quality, measured as a logic visibility metric. Based on this observation, we propose a procedure which biases the selection to increase the restorability of a desired set of *critical* state elements, without sacrificing the (overall) logic visibility. We propose to select the critical state elements to increase the "timing visibility" inside the chip to facilitate the debugging of timing errors which are perhaps the most challenging type of error to debug at the post-silicon stage. Specifically, we introduce a case when the critical state elements are selected to track the transient fluctuations in the power delivery network which can cause significant variations in the delays of the speedpaths in the circuit in nanometer technologies. This paper proposes to use the trace buffer technology to increase the timing visibility inside the chip, without sacrificing the logic visibility.**

I. Introduction

Post-silicon validation of VLSI chips has become significantly time-consuming in nanometer technologies and impacting the product time-to-market. Due to the high complexity of modern day electronic systems, logic bugs may escape the pre-silicon validation stage. However, at this stage, the lack of visibility to the signals inside the chip makes the validation a cumbersome task.

Trace buffer technology has been recently used in order to track few internal state elements (i.e., signals) during the operation of a chip. These signals are selected for tracing at the design stage and the traces are analyzed at the post-silicon stage to debug logic errors. Many recent works have focused on the trace selection problem in order to maximize the chip *logic* visibility [4], [5], [6], where visibility is the metric used to reflect the degree of restoring the remaining state elements in the chip using the selected trace signals.

In this work, we first present two enhancements to the existing trace selection algorithms:

1. During computation of the visibility corresponding to a set of candidate trace signals, we show the ordering of state elements is very important. We discuss an ordering which results in more accurate computation of visibility during trace selection.
2. We propose a Pareto-algebriac procedure which in effect defers the selection of multiple trace signals, compared to existing greedy techniques which select one trace at each step. As a result, we can obtain a solution of higher visibility.

Furthermore, we observe that due to the limited bandwidth of the trace buffer, many alternatives for selecting (the same number of) trace signals can result in the same degree of visibility. We use this observation to extend the definition of trace selection problem such that a desired set of *critical* state elements are more likely to be restored, while maximizing the logic visibility. We propose a procedure by extending the visibility computation algorithm in [5].

We then consider the case when the critical state elements are selected in order to increase the "timing visibility" inside the chip which can be useful to debug timing errors. Timing errors refer to those malfunctions which are due to failure on the speedpaths to meet a target operation frequency of the chip. Although small in number, debug of timing errors can take the majority of the post-silicon validation cycle.

Timing errors may manifest at a specific operating and workload condition. Recent case studies on industrial microprocessors have shown timing errors can be highly sensitive to internal signal transitions. Specifically those transitions which cause fluctuations in the power delivery network (i.e., power-droop) can significantly impact the path delays in nanometer technologies and thus cause timing errors [3]. Upon detecting a timing error, determining if the root-cause is power-droop is quite challenging because of the combination of the following factors: 1) the actual input patterns (e.g., instructions in a microprocessor) which result in excitation of an internal speedpath and subsequently in a timing error may not be known easily; 2) transient simulation of power delivery network is time-consuming; 3) low frequency droops may cause timing errors which are detected many cycles after the switching has occurred.

In this paper, we introduce a method for determining the *critical* state elements (fed to the trace selection procedure) in order to build and update a spatial map of the chip's power-droop for a window of time corresponding to the trace buffer size. Using simulation, we demonstrate that by restoring these critical state elements, a more accurate map of power-droop can be obtained, compared to traditional trace selection procedures which do not pay attention to critical state elements. This is without any sacrifice in the chip logic visibility.

In the remainder of the presentation we first discuss necessary background in Section II. In Section III, we discuss our trace selection algorithm. In Section IV we discuss our procedure to track power-droop. Simulation results are presented in Section V.

II. Preliminaries

In this section, we first review how a visibility metric is defined and computed for an arbitrary gate inside the chip, for a given set of trace signals. We then review the trace selection algorithm of [5].

We consider a graph model of the circuit in which each gate and state element is represented by a node and each interconnect is represented by a directed edge from a node to (one of) its fanout(s). Given a node n_i, we define its "1-visibility" (denoted by v_{i1}) to be the probability that logic value 1 will be (correctly) restored at the output of the corresponding gate or state element, from the few trace signals. The "0-visibility" of node n_i (denoted by v_{i0}) is similarly defined. The expression of v_{i1} is given using the following equation (generalized from [5]).

$$v_{i1} = \begin{cases} P_{i1}; & \text{if } n_i \text{ is a trace signal.} \\ P_{i1} \cdot f(v_{j1}, v_{j0}); & \text{if } \forall n_j \in \text{neighborhood } n_i. \end{cases}$$

Algorithm 1: Computing visibility V^o for a given set of trace signals.

```
1: Set $v_{j0}$=$v_{j1}$=0 ($\forall g_j$ and $s_j$ not traced), $v_{k0}$=$P_{k0}$,$v_{k1}$=$P_{k1}$ ($\forall s_k$ traced).
2: repeat
3:   repeat
4:     for all $s_k$ with $v_k \neq 0$ do
5:       for next node $g_j \in \mathcal{G}_{out}(k)$ do
6:         if $d_j = B$ then
7:           Continue;
8:         end if
9:         $d_j = F$; Update $v_{j0}$ and $v_{j1}$;
10:        Update $v_{l0}$ and $v_{l1}$ if $s_l$ connected to $g_j$;
11:      end for
12:    end for
13:  until $V^o = \sum_{\forall s_j} v_j$ converges
14:  repeat
15:    for all $s_k$ with $v_k \neq 0$ do
16:      for next node $g_j \in \mathcal{G}_{in}(k)$ do
17:        if $d_j = F$ then
18:          Continue;
19:        end if
20:        $d_j = B$; update $v_{j0}$ and $v_{j1}$;
21:        Update $v_{l0}$ and $v_{l1}$ if $s_l$ connected to $g_j$;
22:      end for
23:    end for
24:  until $V^o = \sum_{\forall s_j} v_j$ converges
25: until $V^o = \sum_{\forall s_j} v_j$ converges
```

where P_{i1} is the probability that n_i takes logic value 1. If n_i is a state element which is selected as a trace signal, then v_{i1} is equal to P_{i1}. The value P_{i1} can be computed, for example via simulation for a given set of input patterns. Otherwise, if n_i is not traced, then v_{i1} is a function of v_{j1} and v_{j0} of any node n_j in the *neighborhood* of n_i. These neighborhood nodes in the general case may be the fanin and fanout nodes of n_i, or they may be the other fanins of *a* fanout node of n_i. If n_i is a logic gate, then the function $f()$ in the equation and the neighborhood nodes are specified based on the logic function corresponding to n_i. For example, the authors in [5] provide expressions for $f()$ for few types of logic gates. If n_i is a state element, then $f()$ is equal to v_{j1} for its input n_j.

The expression for v_{i0} is defined using a similar equation. We further define the visibility of node n_i as the summation of its 1-visibility and its 0-visibility: $v_i = v_{i1} + v_{i0}$.

As can be inferred from the above discussion, the 1(or 0)-visibilities of n_i and of a (fanin or fanout) neighbor n_j can be inter-dependent. They should be computed only from the few selected trace signals. The authors in [5] propose an iterative procedure to continuously compute and update the node visibilities until convergence is achieved on a defined metric which reflects the overall visibility.

The procedure is described using Algorithm 1. Here, if n_i is a logic gate, we denote it by g_i, and otherwise it is denoted by s_i for a state element. Starting from any state element s_k with $v_k \neq 0$ (i.e., initially a traced node), the algorithm starts with a "forward pass", where the nodes (gates) in the fanout cone of s_k are visited in a breadth-first manner. We denote the gates in the fanout cone of s_k which are ordered topologically, by the ordered set $\mathcal{G}_{out}(k)$. When a node $g_j \in \mathcal{G}_{out}(k)$ is visited for the first time, a "direction flag" (d_j) will be set to "forward" (F), indicating that g_j is initially reached through a forward pass. For each node $g_j \in \mathcal{G}_{out}(k)$, if $d_j = F$, then the quantities v_{j0} and v_{j1} will get computed (using the current visibilities of the nodes in its neighborhood).

During a forward pass, it may be possible to restore a new state element. (See line 10). At each step of forward pass, the "overall visibility" given by $V^o = \sum_{\forall s_j} v_j$ is computed. The forward pass repeats until the value of V^o converges.

Next, a set of iterative "backward passes" are done similar to the forward pass. Here for a state element s_k with $v_k \neq 0$, a backward pass reaches the gates in its fanin cone. The gates in the fanin cone of s_k which are ordered topologically are denoted by the ordered set $\mathcal{G}_{in}(k)$ in Algorithm 1. Once a node is reached during a backward pass, its direction flag will be checked and the node visibility will be updated only if the node was not reached before through a forward pass. This restriction ensures that a node is restored either through its fanin or its fanout nodes *throughout the entire procedure*. It is one way to ensure redundant information is not used which may result in incorrect computation of the visibilities.

Once a set of iterative forward passes followed by a set of iterative backward passes finish, the entire procedure repeats itself to update V^o. This process continues until the value of V^o converges.

Algorithm 1 is used to drive a trace selection procedure. Specifically, for a trace buffer of bandwidth b, we should select b signals such that the overall visibility V^o is maximized. Typically the following greedy procedure is used for trace selection. To select b trace signals, Algorithm 1 is evoked b times. Each time, one trace signal is selected and added to the set of already-selected trace signals.

III. Trace Selection for Enhancing Timing Visibility

In this work, we would like to extend the trace selection problem so that during maximization of V^o, we further increase the visibility of a set of specified *critical* state elements. We later discuss how the critical state elements are specified to track power-droop.

A. Problem Definition

For a given set of critical state elements denoted by $\mathcal{S}_c$, we define "timing visibility" as

$$V^t = \sum_{\forall s_i \in \mathcal{S}_c} v_i \tag{1}$$

The visibility on the remaining state elements is then given by

$$V^r = \sum_{\forall s_i \notin \mathcal{S}_c} v_i \tag{2}$$

Given a trace buffer bandwidth of b, our goal is to provide a set of possibilities for selecting b trace signals which represent a tradeoff curve between the overall visibility ($V^o = V^r + V^t$) and the timing visibility (V^t). This tradeoff information may be used in different ways; for example, the b traces with the highest V^t within a target V^o can be selected. We discuss more details later in Section IV.

B. Our Procedure for Visibility Computation

Consider Algorithm 1. In computing visibility of each node, either forward or backward direction encoding is used for the node which remains the same throughout the algorithm. This procedure ensures redundant information is not used. In addition during a forward or a backward pass, state elements with non-zero visibility may be selected in any order (in lines 4 and 15). Both of these two factors can significantly impact the computed visibility of the nodes and are important when considering the timing visibility for the critical state elements. We propose a revised version of Algorithm 1. Our method returns both V^t and V^r (instead of just V^o in Algorithm 1). Our goal is to improve Algorithm 1 by two means: 1) identifying a more effective ordering of the state elements with non-zero visibility; 2) appropriately imposing F or B direction during each pass.

Algorithm 2 gives our visibility computation procedure. First, for each state element s_k we compute a forward *reachability* weight W_k^F. This weight reflects the reachability of s_k to critical state elements in $\mathcal{S}_c$, if we decide to forward propagate from s_k. Similarly we define a backward reachability weight W_k^B if we decide to backward propagate from s_k. These are given in lines 2-5 of the algorithm.After computing the weights, we sort them in descending order (line 6).

Algorithm 2: Computing V^t and V^r for a given set of trace signals.

```
1: Set $v_{j0}$=$v_{j1}$=0 ($\forall g_j$ and $s_j$ not traced), $v_{k0}$=$P_{k0}$,$v_{k1}$=$P_{k1}$ ($\forall s_k$ traced).
2: for all $s_k$ do
3:    $W_k^F$ = compute reachability weight ($s_k$, $F$); //use Algorithm 3
4:    $W_k^B$ = compute reachability weight($s_k$, $B$); //use Algorithm 3
5: end for
6: Sort all $W_k^F$ and $W_k^B$ values in descending order ($\forall s_k$).
7: repeat
8:    Select $W_k$, the weight with the next largest value in the ordered list
9:    if $v_k \neq 0$ then
10:       $d_k = (W_k == W_k^F)?F : B$.
11:       Set $\mathcal{G}(k) = (d_k == F)?\mathcal{G}_{out}(k) : \mathcal{G}in(k)$;
12:       for next node $g_j \in \mathcal{G}(k)$ do
13:          if $d_j \neq d_k$ then
14:             Continue;
15:          end if
16:          $d_j = d_k$; update $v_{j0}$ and $v_{j1}$;
17:          Update $v_{l0}$ and $v_{l1}$ for $s_l$ connected to $g_j$;
18:       end for
19:    end if
20: until updated $V^t$ and $V^r$ converge
```

Next an iterative procedure begins. At each iteration, a state element s_k with non-zero visibility is identified together with a propagation direction (lines 7-10). We first identify the highest forward or backward weight value in the sorted list (and restart from beginning of the list, when we reach the end). Then the state element which has the highest weight is identified. Next, if the state element has a non-zero visibility, then we select it for propagation. The propagation direction corresponds to its weight type in line 10 (e.g., forward direction for forward weight).

Next, based on direction d_k, we identify an ordered set of gates $\mathcal{G}(k)$ which are either in the fanin or the fanout cone of s_k (see line 11). If direction is forward propagation, then the set is $\mathcal{G}_{out}(k)$ (i.e., the fanout cone of k as defined in the previous section). Similarly, if the direction is backward propagation, the set is $\mathcal{G}_{in}(k)$. For each gate $g_j \in \mathcal{G}(k)$, we check if its direction d_j is same as d_k. We update the node visibilities only if the directions are the same. Otherwise, we move to the next node in $\mathcal{G}(k)$. These steps are shown in lines 12-18. We update both V^t and V^r values after one forward or backward propagation ends for the considered state element s_k.

In our algorithm, the direction of each node is marked only once and will never change again. This ensures the visibility of the node will be computed (correctly) only through forward or backward propagation. Otherwise, the use of redundant information may result in over-estimation of a node visibility. Our procedure allows more flexibility in reaching the nodes. It furthermore will try to reach the critical state elements through the state elements (of non-zero visibility) which have the highest reachability weights. As we shortly discuss, these weights are reflective of "reachability" to critical state elements. The algorithm terminates when both values of V^t and V^r have converged.

C. Computing the Reachability Weights

We are given s_k and a direction d_k of either F or B. The objective is to find a corresponding weight (W_k^F or W_k^B) reflecting the reachability to the critical state elements. Algorithm 3 describes our procedure. First, define a "state fanin cone" for s_k which is only composed of state elements and is denoted by $\mathcal{S}_{in}(k)$. We form this set by recursively traversing the fanin cones of each state element as they are visited, starting from s_k until reaching the primary inputs. All the state elements that are visited in this traversal are added to the set $\mathcal{S}_{in}(k)$ as shown in Figure 1. We similarly define a "state fanout cone" for s_k and denote it by $\mathcal{S}_{out}(k)$.

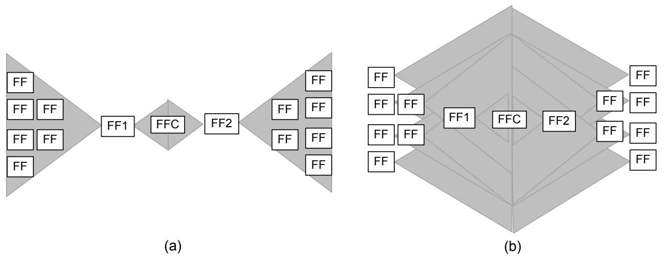

Fig. 1. Estimated visibility of critical state element is higher using the ordering in (b) compared to (a).

Algorithm 3: Reachability weight computation given s_k and d_k.

```
1: Set $\mathcal{S}(k) = (d_k == F)?\mathcal{S}_{out}(k) : \mathcal{S}_{in}(k)$;
2: for all $s_j \in \mathcal{S}(k)$ do
3:    if $s_j \in \mathcal{S}_c$ then
4:       $D_{jk}$=$L_{max}$-$|L_i - L_j|$.
5:       $W$+=$D_{jk}$.
6:    end if
7: end for
8: Return $W$
```

Given s_k and direction d_k, we first identify a set of state elements denoted by $\mathcal{S}(k)$, corresponding to either $\mathcal{S}_{in}(k)$ or $\mathcal{S}_{out}(k)$ (line 1). We then go through each $s_j \in \mathcal{S}(k)$ which can be in arbitrary order. If s_j is a critical state element, we update the weight of s_k.

The weight update is by summing a factor which we found to be very helpful to increase timing visibility. This factor is a distance D_{jk} between s_j and s_k given by line 4. It is computed using the levels of state elements s_j and s_k (denoted by L_j and L_k), and a maximum level $L_{max} = \max_{\forall s_i}(L_i)$. The level of a state element is the maximum number of state elements required to reach from any primary input to it. As can be seen from line 4, if s_k is closer to critical state element s_j, then D_{jk} is a larger value implying a higher reachability of s_k to the critical s_j. We update the weight of s_k, every time a critical state element is reached in $\mathcal{S}(k)$.

Figure 1 illustrates the benefits of our visibility computation algorithm which impacts the trace selection, compared to the one explained in Section II. Consider a critical state element FFC$\in \mathcal{S}_c$. Figure 1(a) represents the case when the visibility of FFC is computed using Algorithm 1. In this case, the visibility of FFC is computed from the nodes in the fanout cone of FF1 and fanin cone of FF2. However, using Algorithm 2 and by incorporating the reachability weights to order the state elements, we can estimate a much higher visibility for FFC by using all the state elements (see Figure 1(b)).

D. Trace Selection Algorithm

Algorithm 4 for trace signal selection replaces the greedy heuristic of [5]. The inputs of the algorithm are the number of trace signals (b), and the number of trace signal selection alternatives (N). The algorithm generates N trace selection possibilities such that they provide a tradeoff between V^t and V^o using concepts from Pareto algebra [2]. We denote $\mathcal{C}$ as a set of configurations where the k^{th} configuration, c_k, is represented as a triple given by $(\mathcal{T}_k, V_k^o, V_k^t)$. Here $\mathcal{T}_k$ is a set of selected trace signals for configuration c_k with corresponding V_k^t and V_k^o.

In Algorithm 4, we initially generate $|FF|$ number of configurations, where $|FF|$ is the number of state elements. Each configuration only has one of the state elements as a trace signal (lines 1-5).

Algorithm 4 then follows b steps. At the beginning of step i, each configuration c_k in $\mathcal{C}$ has i trace signals in its $\mathcal{T}_k$ field. Then a new trace signal is added to each of the configurations. The result is a new configuration set $\mathcal{C}_{tmp}$ in which each configuration has $i+1$ trace signals (lines 7-13).

Algorithm 4: Generating N alternatives of selecting b trace signals.

```
1: $\mathcal{C} = \emptyset$;
2: for $i = 0$ to $|FF|$ do
3:     $(V_i^o, V_i^t)$= compute visibility if $s_i$ is trace signal; //use Algorithm 2
4:     $\mathcal{C} = \mathcal{C} \bigcup (\mathcal{T}_i = \{s_i\}, V_i^o, V_i^t)$;
5: end for
6: for $i = 0$ to $b$ do
7:     for all $c_k \in \mathcal{C}$ do
8:         for all $s_j \in \{FF - \mathcal{T}_k\}$ do
9:             add $s_j$ to $\mathcal{T}_k$
10:            $(V_j^o, V_j^t)$ = compute visibility, given trace signals $\mathcal{T}_k \bigcup s_j$;
11:            $\mathcal{C}_{tmp} = \mathcal{C} \bigcup (\mathcal{T}_k \bigcup \{s_j\}, V_j^o, V_j^t)$;
12:        end for
13:    end for
14:    $\mathcal{C}_{tmp}$.min();    //min operation extracts Pareto points
15:    $\mathcal{C}_{tmp}$.reduce($N$);    //reduce operation samples N Pareto points
16:    $\mathcal{C} = \mathcal{C}_{tmp}$;
17: end for
```

The set $\mathcal{C}_{tmp}$ is then reduced in size to store only N configurations which provide a tradeoff between their V^t and V^o fields. This is done by examining the set in the two dimensional space of V^t and V^o and first extracting the *Pareto points* using the `min` operation of Pareto algebra [2] (line 14). We then apply a `reduce` procedure to sample N Pareto points (line 15). This is done by uniform sampling of the 2-dimensional V^o—V^t space. In the end of step i, configuration $\mathcal{C}$ is updated to $\mathcal{C}_{tmp}$. In the end, $\mathcal{C}$ will include N configurations, each with b trace signals which provide a tradeoff between V^o and V^t.

Consider the run-time complexity of Algorithm 4. In each step, N configurations are combined with the remaining state elements of the design ($O(|FF|)$). Consequently, the complexity of each step is O($N \times |FF|$). The overall complexity for b steps is O($b \times N \times |FF|$).

IV. Critical State Element Selection

Power droop which is the transient fluctuations in the power delivery network is an important cause of post-silicon timing errors [3]. Each logic cell or state element draws current from the power delivery network proportional to its size when it is switching. The power droop at each instance of time is highly proportional to the switching map of the logic cells and state elements at that time. In addition to the temporal dependence, the amount of droop at a specific location also depends on the localized switching activity for the logic cells and state elements at that location [1]. Upon observing a timing error, it is usually very difficult to understand if the root-cause of failure is due to power droop.

Selection Process: Our procedure to select critical state elements helps build a spatial map of switching activity of the chip at each instance of time and for a fairly large timing window corresponding to the trace buffer size. We divide the chip uniformly based on a coarse grid into M bins and select one critical state element per bin which can best restore the states of the sequential elements of that region. Our procedure is enumerated below.

1. For each grid bin, identify all state elements falling in the bin from the placement information. We denote the set of state elements in bin i by $\mathcal{S}_i$.
2. Apply Algorithm 1 for the state elements in $\mathcal{S}_i$ ($\forall i = 1, .., M$). For each bin select *only* one critical state element.

The M state elements selected in the M bins compose $\mathcal{S}_c$. The set $\mathcal{S}_c$ is provided to Algorithm 4 which generates multiple trace selection scenarios and provides a tradeoff between V^t and V^o. To use the tradeoff curve, for a target overall visibility V^o_{tar}, e.g., based on estimate using Algorithm 1, we select the solution with V^o which is closest to V^o_{tar} to get the highest corresponding timing visibility V^t. Note after applying Algorithm 2, the critical state elements may or may not be selected as a trace signal.

Debug Process: During post-silicon validation, upon observing a bug, one can detect the corresponding input patterns (e.g., instruction in a microprocessor) at that instance. Appropriate events should then be triggered to allow the system to run back within a timing window of T instructions, corresponding to the trace buffer size, before the bug was observed. This time the trace signals will be stored in the trace buffer and extracted for analysis to determine if the cause of timing failure is power droop. Specifically, the states of many sequential elements and logic gates will be restored for each cycle. Then a spatial map of power droop is computed for each cycle where the droop at each location is proportional to the switching activity of the cells and the sizes of the switching cells at that location.

V. Simulation Results

A. Evaluation of Restorability

We implement the procedure of [5] which uses Algorithm 1 for visibility computation and a greedy procedure for trace selection. We also implement Algorithm 2 which is provided a set of critical state elements and Algorithm 4 for trace selection. All codes are implemented using C++. In our experiments we consider 6 large cases from ISCAS'89 benchmarks. The benchmarks are synthesized using Synopsys Design Compiler and a 90nm TSMC library. They were placed using Cadence Encounter for the power droop experiment.

To compute the functional probability of each signal (P_{i1} and P_{i0} defined in Section II), we simulate each benchmark for 10000 cycles with random values for primary inputs using an event-driven simulator. We also use our event-driven simulator to generate logic values at all the nodes to be used as golden reference—both for trace generation, and for evaluation of restorability in the remaining nodes. This is done by generating 1000 randomly-generated primary input patterns and computing logic values for all the nodes at each cycle. We select 16 critical state elements to monitor the power droop using a 4×4 grid based on the procedure given in Section IV.

After selecting the trace signals (using either of the algorithms), their logic values are taken from the golden reference at the corresponding nodes. We then implement a forward and a backward X-simulator to restore as many untraced state elements as possible using the trace values. We start by assuming all the untraced state elements have a value of X. Then we iteratively apply the forward and backward X-simulators. At each iteration we propagate forward (and then backward) using the trace values from the golden simulator only from the traced state elements, and try to restore the logic values of the remaining state elements. We repeat the process iteratively until no further restoration is possible. We compute two restorability factors: 1) R^o: number of state elements (critical or non-critical) which could be fully or partially restored; 2) R^t: number of critical state elements which could be partially or fully restored. We consider trace buffer bandwidths of 16 and 32 and compare three cases.

- `Greedy-V`^{o}: The greedy trace selection approach of [5]. It uses Algorithm 1 to compute (only) V^o;
- `Greedy-V`^{t}: Algorithm 4 with N=1 is used for trace selection and Algorithm 2 is used to compute V^t and V^o;
- `Pareto-V`^{t}: Same as `Greedy-V`^{t} but we use N=4.

In the second case by setting N=1 we force selection of only one Pareto-point at each step of trace selection which was the state element providing the maximum overall visibility V^o. This strategy was equivalent to a greedy selection of trace signals similar to [5]. But unlike [5] we maximized V^t within the visibility computation. In `Pareto-V`^{t}, for N=4 we generate four different trace selection possibilities. We then report two scenarios: (a) We select the solution which has the closest visibility V^o to the one in `Greedy-V`^{o}; (b) We select the solution which has the maximum timing visibility V^t.

TABLE I

COMPARISON OF R^o AND R^t FOR TWO CASES OF TRACE SIGNAL SELECTION.

b=16

Testbench	$\|FF\|$	`Greedy-V`^{o} [5]			`Greedy-V`^{t}			`Pareto-V`^{t} (a)		(b)		runtime (sec)
		R^o	R^t	runtime (sec)	R^o	R^t	runtime (sec)	R^o	R^t	R^o	R^t	
S15850	137	132	13	16	137	16	18	137	16	137	16	97
S35932	1728	302	14	242	313	16	239	315	16	313	16	1561
S38417	1564	202	10	3214	213	15	3352	210	14	210	15	9253
S38584	1166	209	8	1472	199	13	887	213	15	211	16	2997
S5378	163	130	12	10	129	10	15	129	13	128	16	52
S9234	145	65	10	12	69	10	10	69	10	64	11	39

b=32

Testbench	$\|FF\|$	`Greedy-V`^{o} [5]			`Greedy-V`^{t}			`Pareto-V`^{t} (a)		(b)		runtime (sec)
		R^o	R^t	runtime (sec)	R^o	R^t	runtime (sec)	R^o	R^t	R^o	R^t	
S15850	137	137	16	51	137	16	59	137	16	137	16	141
S35932	1728	350	16	1400	380	16	1423	380	16	377	16	6354
S38417	1564	222	8	5124	221	16	5021	320	16	312	16	15421
S38584	1166	210	9	4510	214	15	2313	225	14	210	16	3215
S5378	163	145	13	31	147	16	40	146	15	146	16	168
S9234	145	90	10	35	90	10	41	90	10	89	15	110

Table 1 shows the results. Column 2 shows the number of state elements for each benchmark ($|FF|$). First, we observe a higher value of R^o in `Greedy-V`^{t} compared to `Greedy-V`^{o} (columns 3 and 6). Note, both techniques use the same greedy trace selection mechanism while ours is only different in visibility computation. We obtain a better solution only by imposing ordering of state elements and appropriate forward or backward direction during the visibility computation. Considering `Pareto-V`^{t}(a) in column 9 we show further improvement in R^o by deferring the trace selection decision.

Second, comparing R^t for the 16 critical state elements, we observe `Pareto-V`^{t}(a) is better than `Greedy-V`^{t} which is in turn better than `Greedy-V`^{o} (columns 4, 7, 10). When considering picking the solution with maximum V^t in `Pareto-V`^{t}(b), we observe more improvement in R^t compared to `Pareto-V`^{t}(a) and `Greedy-V`^{t}. Notice, R^o is also comparable in `Greedy-V`^{t} and `Pareto-V`^{t}.

Comparing runtimes, `Greedy-V`^{t} does not result in any additional runtime overhead over `Greedy-V`^{o}. The runtimes of `Pareto-V`^{t}(a) and (b) are similar so one runtime is reported. It is larger than `Greedy-V`^{t} since now four alternatives are generated.

B. Enhancing the Power Droop Visibility

In this experiment we consider the following three cases.

- `Golden`: In this case all the primary inputs patterns are known. We simulate the design for 1000 cycles to compute the switching activity of all the gates and state elements.
- `Greedy-V`^{o}: We take the solution of `Greedy-V`^{o} for $b = 32$ from the previous experiment. We use the values *only* at the selected state elements using the same patterns of the `Golden` case. At each cycle we use forward and backward simulations to restore the values for all gates and state elements. For the unrestored nodes at a clock cycle, we (pessimistically) assume switching.
- `Greedy-V`^{t}: We take the solution of `Greedy-V`^{t} from the previous experiment (i.e., for the same 16 critical state elements to uniformly cover a grid of 4x4). Again, at each clock cycle we apply forward and back simulations to restore all possible states of combinational and sequential nodes and for the unrestored nodes, we assume the pessimistic case of switching.

Spatial Comparison: To build the spatial map at a given clock cycle, we consider a resolution of 8×8. For each of the 64 bins, we estimate power droop as follows. We add the weighted switching activities of all gates and state elements in a region where a weight corresponds to the area of that component. We denote this estimate by SW. We compute the error in SW in `Greedy-V`^{o} and `Greedy-V`^{t} against `Golden` for each bin. We report the average of the errors over all the bins at clock cycle 500 in columns 2 and 3 of Table 2.

TABLE II

COMPARISON OF ERROR OF WEIGHTED SWITCHING ACTIVITY

Bench	Spatial		Temporal	
	`Greedy-V`^{o}	`Greedy-V`^{t}	`Greedy-V`^{o}	`Greedy-V`^{t}
S15850	13%	8%	25%	11%
S35932	47%	41%	48%	42%
S38417	33%	23%	32%	19%
S38584	31%	30%	34%	21%
S5378	17%	10%	28%	19%
S9234	21%	20%	17%	11%
Ave	27.0%	22.0%	30.7%	20.5%

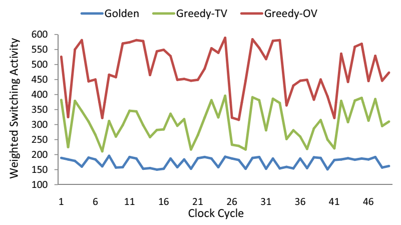

Fig. 2. Temporal profiling of the weighted switching activity.

The error in `Greedy-V`^{t} is smaller than `Greedy-V`^{o}.

Temporal Comparison: Here, we consider the summation of SW for all the 64 bins at each clock cycle. We compute the error at each clock cycle for `Greedy-V`^{o} and `Greedy-V`^{t} against `Golden`, and report the average error over the 1000 clock cycles in columns 4 and 5 of Table 2. The error in `Greedy-V`^{t} is smaller than `Greedy-V`^{o}. Figure 2 plots this quantity for 50 clock cycles in benchmark S9234.

VI. CONCLUSIONS

We introduced a new variation of trace signal selection problem to also enhance visibility on a set of pre-specified critical state elements. We discussed a case when critical state elements are selected to track transient fluctuations in the power delivery network which can be used to analyze timing errors. We also proposed two enhancements to the conventional trace selection algorithms to improve logic visibility.

REFERENCES

[1] E. Alpaslan, et al. NIM - A noise index model to estimate delay discrepancies between silicon and simulation. In *DATE*, 2010.
[2] M. Geilen, T. Basten, B. Theelen, and R. Otten. An algebra of Pareto points. *Fundam. Inf.*, 78(1):35–74, 2007.
[3] K. Killpack, et. al, Case study on speed failure causes in a microprocessor. *IEEE Design & Test of Computers*, 25:224–230, 2008.
[4] H. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *TCAD*, 28:285–297, 2009.
[5] X. Liu and Q. Xu. Trace signal selection for visibility enhancement in post-silicon validation. In *DATE*, 2009.
[6] J.-S. Yang and N. Touba. Automated selection of signals to observe for efficient silicon debug. In *VTS*, IEEE, 2009.