A Hybrid Approach for Fast and Accurate Trace Signal Selection for Post-Silicon Debug

Min Li and Azadeh Davoodi
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
Email: {mli46, adavoodi}@wisc.edu

Abstract— **The main challenge in post-silicon debug is the lack of observability to the internal signals of a chip. Trace buffer technology provides one venue to address this challenge by online tracing of a few selected state elements. Due to the limited bandwidth of the trace buffer, only a few state elements can be selected for tracing. Recent research has focused on automated trace signal selection problem in order to maximize restoration of the untraced state elements using the few traced signals. Existing techniques can be categorized into high quality but slow "simulation-based", and lower quality but much faster "metric-based" techniques. This work presents a new trace signal selection technique which has comparable or better quality than simulation-based while it has a fast runtime, comparable to the metric-based techniques.**

I. Introduction

Post-silicon validation of VLSI chips has become significantly time-consuming in nanometer technologies. It thus impacts the time-to-market of electronic products. Due to the high complexity in modern designs, logic bugs may escape the pre-silicon validation stage. Later on, at the post-silicon stage, the lack of visibility to the signals inside the chip makes the validation a cumbersome task.

Trace buffer technology has been proposed in order to track a few internal state elements within the observation window when the chip is operating [1]. These signals are selected for tracing at the design stage and the traces are analyzed at the post-silicon stage to debug logic errors. Specifically, the collected traces are first used to restore as many other state elements within the observation window. State Restoration Ratio (SRR) has been used to measure the quality of a set of selected trace signals. SRR is computed using simulation of the circuit. A larger SRR means that a higher number of untraced state elements can be restored using the traced signals within the observation window. It thus allows a more effective analysis to localize the bug in time and space.

Due to the complexity of modern IC designs, the selection of trace signals can no longer be done manually. Recently, many automated trace selection algorithms are proposed in order to increase the SRR of the generated solution. These algorithms can be categorized into two types, namely metric-based and simulation-based algorithms.

Metric-based algorithms utilize metrics which allow approximating the ability of a candidate trace signal to restore the untraced state elements while taking into account the restoration that can be made from a subset of already-selected trace signals. For example the work in [4] defines "forward restorability" and "backward restorability" to help select the trace signals. However, these metrics were shown to result in a high approximation error of the SRR in a number of circumstances [3]. Later on, improved metrics namely "restorability" and "visibility" were proposed in [5] and were shown to result in a higher solution quality in terms of SRR, compared to [4]. The work [2] further improves the solution quality by introducing a new set of metrics which are based on signal dependency, and by iteratively updating and expanding a "traced region" until the required number of state elements are selected. Besides the above-mentioned algorithms, the work [6] introduces a variation of the problem which aims to increase the likelihood of restoration on a set of *critical* state elements as a secondary objective besides maximizing the SRR.

This research was supported by NSF under Grant 1053496.

The main advantage of metric-based algorithms is their fast execution runtime due to quick evaluation of the related metrics. However, their solution quality, measured in terms of SRR, are drastically lower than the simulation-based algorithms. Specifically, the work [3] elaborates that simulation is a better way to emulate the restoration process and naturally captures different sources of inaccuracy such as signal correlation. By using simulation and utilizing a pruning-based strategy, the work [3] achieves the best solution quality among the existing works. However, the runtime of [3] is significantly higher than metric-based algorithms despite a GPU-based implementation.

The work [7] also proposes a trace selection algorithm by solving an ILP-based formulation based on error propagation. The use of circuit satisfiability as a more accurate alterative to simulation for trace signal selection is also discussed in [8].

A. Contributions

In this work, we present a new trace selection algorithm which is hybrid and utilizes the right blend of simulation with quickly-measured metrics. The contributions of this work are as follows.

- A new set of metrics are proposed to quickly identify the top candidates for tracing at each step of our algorithm.
- These metrics include an Impact Weight of each state element for relative comparison which further depends on a proposed *demand* metric. The demand metric reflects the remaining restoration needed by a state element to be restored from a candidate state element for tracing. It takes into account a partial restoration from the already-selected trace signals.
- Updating the Impact Weight and demand metrics at each step is based on computing a restorability rate for each untraced state element. The restorability rates of *all* the untraced state elements are computed with an overall small number of simulations.
- At each step of our algorithm, after a few top candidates for tracing are identified using the Impact Weights, simulation is used to accurately evaluate the SRR for each candidate.

In our experimental results, we use a setup similar to the previous works and experiment with trace buffers of different bandwidths. We also take into account the impact of control signals in our algorithm when conducting our experiments. We demonstrate in our simulations that the solution quality of our algorithm in terms of the measured SRR is comparable or better than a simulation-based approach which is the best reported solution quality among the existing algorithms. At the same time, our algorithm is significantly faster than simulation-based and has a runtime comparable to metric-based techniques which have the fastest runtime among the existing algorithms.

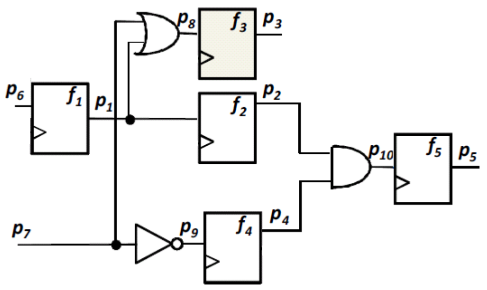

Fig. 1. Example for illustration of the notations and metrics

II. Preliminaries

A. Terminology and Problem Definition

Given a sequential circuit, we denote a signal $s := (p, v, n)$ when pin p takes value v at clock cycle n. Here the value v could be 0, 1, or x if the value is unknown. Let pin p designate the index of an output pin of either a combinational gate or a state element. It may also designate a primary input which may further be a control signal, typically to select an operation mode of the circuit. We denote the subset of pins for state elements, gates, and control by $\mathcal{P}_F$, $\mathcal{P}_G$, and $\mathcal{P}_C$, respectively. For signal $s = (p, v, n)$ in an observation window of N clock cycle, we assume $n = 1, 2, \ldots, N$. For a control signal (p, v, n), we have $p \in \mathcal{P}_C$ and its value is known ($v \neq x$).

For pin $p \in \mathcal{P}_G$, we denote FO_p as the set of its "fanout pins" which are outputs of a combinational gate for which p is an input. Similarly, we denote FI_p to be the set of "fanin pins", if they are inputs of a combinational gate for which p is an output.

We define a *trace signal* (p, v, n) if $p \in \mathcal{P}_F$, corresponding to a state element. The trace signal is captured at run-time for an observation window of $1 \leq n \leq N$. Also since the signal is captured in an on-chip trace buffer, its values are known within the observation window and we have $v \neq x$. Let us denote the set of the trace signals with $\mathcal{S}_T$. The size of $\mathcal{S}_T$ is $B \times N$ for a trace buffer of bandwidth B allowing simultaneous tracing of B signals in N cycles. As an example, in Figure 1, we have $\mathcal{P}_F = \{p_1, p_2, p_3, p_4, p_5\}$, $\mathcal{P}_G = \{p_8, p_9, p_{10}\}$. For p_8 we have $FI_{p_8} = \{p_1, p_7\}$ and $FO_{p_8} = \{p_3\}$. The highlighted flipflop f_3 is traced, so we have $\mathcal{S}_T = \{(p_3, v, n)\}$.

A signal (p, v, n) is defined to be *restored* in cycle n if pin p does not correspond to a pin of a trace signal or of a control input signal, and the value v can be restored to 0 or 1 by using the values of the trace and control signals. The algorithmic procedure for determining if a signal can be restored will be explained shortly. The trace selection problem aims to find B trace signals such that the total number of restored signals over the N cycles are maximized.

B. Restoration Using An XSimulator

The set of signals which can be restored using the trace and control signals is determined using an XSimulator. Algorithm 1 describes our variation of an XSimulator given in [4] which we refer to as the `XSim-core` procedure. The inputs to the algorithm are a set of signals at a single cycle n denoted by $\mathcal{S}_I^n$ which are assumed to have a known value of 0 or 1. For example $\mathcal{S}_I^n$ could be a combination of the trace and control signals at cycle n.

In our variation of XSimulator, we also introduce a binary "`restore-once`" flag as input. It controls if restoration of an (unknown) signal should stop as soon as the signal takes a known value ($\neq x$). Otherwise, it is possible for a signal to be restored multiple times. More details about the use of this flag in our algorithm will be discussed in Section III. The output is the set of signals which can be restored *using* the input signals and is denoted by $\mathcal{S}_R$. Note, these signals may be restored at any clock cycle which could be same as, before or after n. However the XSimulator does not record this cycle and instead uses a '0' for the cycle field of a restored signal.

Algorithm 1 `XSim-core`($\mathcal{S}_I^n$, &$\mathcal{S}_R$, `restore-once`)

```
1: $\mathcal{S}_R^n = \emptyset$;  $Q = \emptyset$;  visited[$p$]=false;  $\forall p$
2: **for** each $s := (p, v, n) \in \mathcal{S}_I^n$ **do**
3:     Enqueue($Q$, $s$);  visited[$p$] = restore-once;
4: **end for**
5: **while** $Q \neq \emptyset$ **do**
6:     $s_i := (p_i, v_i, 0)$
7:     $s_i \leftarrow$ Dequeue($Q$)
8:     **for** each $p \in \{FI_{p_i} \cup FO_{p_i}\}$ and !visited[$p$] **do**
9:         $s := (p, v = x, 0)$
10:        evaluate if $s$ can be restored using $\{\mathcal{S}_I^n \cup \mathcal{S}_R\}$
11:        **if** $v \neq x$ **then**
12:            $\mathcal{S}_R \leftarrow \mathcal{S}_R \cup \{s\}$
13:            Enqueue($Q$, $s$);  visited[$p$] = restore-once;
14:        **end if**
15:    **end for**
16:    if the values of all the signals remain unchanged set $Q = \emptyset$
17: **end while**
```

The procedure starts by marking each pin as not visited, except for the ones corresponding to $\mathcal{S}_I^n$ which are set to the `restore-once` flag. The signals in $\mathcal{S}_I^n$ are also added to a queue Q. (See lines 1-3.)

At each step the signal s_i is dequeued from the head of the queue. Then a signal s defined as $s := (p, v = x, 0)$ which corresponds to a fanin or fanout pin of s_i is considered for restoration. If p has not been visited before, its value is evaluated given the values of the other fanins and fanouts which are known ($\neq x$). (See lines 6-10.) Such fanins and fanouts belong to either $\mathcal{S}_I^n$ or have been restored in the previous steps of the algorithm so they belong to the current set of restored signals $\mathcal{S}_R$. If s is restored, then $v \neq x$ after the evaluation and it is added to $\mathcal{S}_R$. Next, pin p takes the value of `restore-once` flag and s is enqueued. The process terminates when the queue is empty which may happen if all the signals are dequeued or if there is no change in the values of the signals compared to the previous iteration of the while loop. (See line 16.) The algorithm outputs the latest $\mathcal{S}_R$ as the set of restored signals using $\mathcal{S}_I^n$.

If the `restore-once` flag is true, as soon as a signal is restored, the algorithm stops considering it for further restoration. As a result, fewer signals may be restored but the algorithm terminates much faster. For example in Figure 1, assume p_4 is restored to 1, p_2 is not restored, and the `restore-once` flag is true. As a result, p_{10} will not be restored. However if `restore-once` is false, it is possible for p_4 to be later restored to 0 (thus enqueued more than once) which allows further restoration of p_{10}. Our trace selection procedure utilizes both modes, for quickly finding a subset of restorable signals as well as finding all the restorable signals.

A common measure for the quality of trace selection is the **State Restoration Ratio (SRR)**, computed within an observation window of M clock cycles. SRR is computed using Algorithm 1 as follows. We are given the input set S_I^n as the trace signals observed in a window of M clock cycles and the control signals. The `restore-once` flag is also set to false. Next, the `XSim-core` procedure is evoked M times, for $n = 1, \ldots, M$. After finding S_R for each cycle we have $SRR = (B \times M + \sum_{n=1}^{M} k_n)/B \times M$ where B is the trace buffer bandwidth, and k_n indicates the number of restored signals which correspond to state elements using S_I^n. For an example to compute SRR, please refer to related work [3].

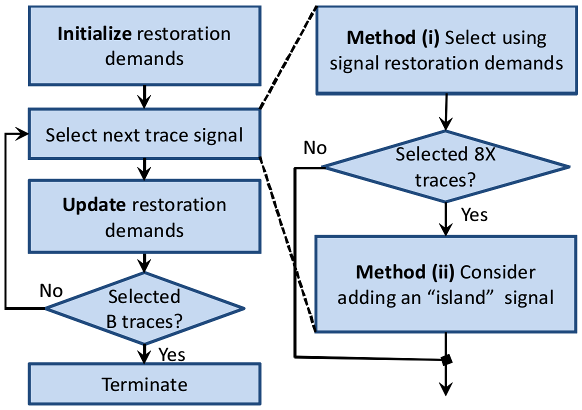
Initialize restoration demands
Select next trace signal
Update restoration demands
Selected B traces?
No
Yes
Terminate
Method (i) Select using signal restoration demands
Selected 8X traces?
No
Yes
Method (ii) Consider adding an "island" signal

Fig. 2. Overview of our trace selection algorithm

Algorithm 2 `Reachability-list`$(f, v, \mathcal{S}_C, \&L_f^v)$

```
1: $s := (p_f, v \neq x, 0)$
2: XSim-core($\{s\} \cup \mathcal{S}_C, \&\mathcal{S}_R$, restore-once=true)
3: XSim-core($\mathcal{S}_C, \&\mathcal{S}_{R_C}$, restore-once=true)
4: $S_R = S_R \setminus \mathcal{S}_{R_C}$
5: Return $L_f^v$ as the set of state elements in $S_R$
```

III. Algorithm

Figure 2 shows an overview of our trace selection algorithm. Our algorithm is driven by computing and continuous updating of an Impact Weight which is defined based on a new metric introduced in this work, namely the "restoration demand". It reflects the remaining demand of a state element i to be restored by another state element f when f takes a known value of 0 or 1, and given that i may be partially restored by an existing set of already-selected trace signals. Computation of this metric is quite fast using the `XSim-core` procedure. After a pre-processing step to compute the initial restoration demands and weights, at each step, one or more new trace signals are selected which is followed by updating the restoration demands and weights for the next step, until all the B trace signals are selected.

The procedure for selecting the next trace signal involves two methods. Method (i) uses the restoration demands and is applied at each step. It does not favor a small class of state elements, referred to as "islands" in this work, which have a poor rate of restoring the other state elements, in the presence of very few or no other trace signals. So after a few steps (e.g., every 8 selected signals), method (ii) is used to consider adding an island signal. (See Figure 2.)

We first define the restoration demand of each state element and the island state elements before discussing the steps of the algorithm.

A. Restoration Demand

We first define other metrics to enable discussing the restoration demand. An example is also given to illustrate the metrics.

1) L_f^v**: Reachability list of state element** f **taking value** v*:* Given state element f, we consider the case *if* it takes a known value of 0 or 1. We denote the reachability list by L_f^v and define it as a set of state elements which can be restored only if state element f takes a known value of $v \neq x$. Note this definition is not associated with any particular clock cycle. In other words, the reachability list L_f^v allows identifying those state elements whose values can immediately be restored if state element f takes a known value without relying on any other (restored) state elements.

Algorithm 3 `Restorability-FFs`$(F, \mathcal{S}_C, \mathcal{S}_T^1, \ldots, \mathcal{S}_T^M, \&r_f\ \forall f \in F)$

```
1: $r_f = 0 \quad \forall f \in F$
2: XSim-core($\mathcal{S}_C, \&\mathcal{S}_{R_C}$, restore-once=false)
3: for n=1 to $M$ do
4:     XSim-core($\mathcal{S}_{R_C} \cup \mathcal{S}_T^n, \&\mathcal{S}_R$, restore-once=false)
5:     for each $f \in F$ do
6:         $r_f = r_f + 1$ if $f$ is a state element in $\mathcal{S}_R$
7:     end for
8: end for
9: return $r_f = \frac{r_f}{M}\ \forall f \in F$
```

For each state element f, two reachability lists with values $v = 0$ and $v = 1$ are computed. Algorithm 2 shows the procedure. The inputs are the state element f, its considered value $v \neq x$, the set of input control signals denoted by $\mathcal{S}_C$. (The notation does not associate the control signals with a clock cycle because we assume they remain constant within the observation window.) The output is L_f^v.

First, signal s is formed corresponding to state element f. (Since no particular cycle is considered, a special value of '0' is used for the clock cycle field in s.) In line 2, the `XSim-core` procedure is called with the union of control signals and s as its input to identify the signals $\mathcal{S}_R$ which can be restored. Since some of the signals in $\mathcal{S}_R$ may be restorable solely using $\mathcal{S}_C$, they must be removed to allow identifying the signals which can be restored when s is also used. To remove these signals, in line 3, the `XSim-core` procedure is called again, this time using only $\mathcal{S}_C$ as input. The restored signals denoted by $\mathcal{S}_{R_C}$ are then removed from S_R (line 4). The output L_f^v is the set of state elements corresponding to the signals in S_R.

Note, when calling the `XSim-core` procedure, the `restore-once` flag is set to true. This ensures quick computation of the reachability list within our trace selection algorithm.

For example, in Figure 1 we have $L_1^0 = \{f_2, f_5\}$, $L_1^1 = \{f_2, f_3\}$, $L_2^0 = \{f_1, f_5\}$, $L_2^1 = \{f_1, f_3\}$, $L_3^0 = \{f_1, f_2, f_4, f_5\}$, $L_3^1 = \emptyset$, $L_4^0 = \{f_5, f_3\}$, $L_4^1 = \emptyset$, $L_5^0 = \emptyset$, $L_5^1 = \{f_1, f_2, f_3, f_4\}$.

It is possible that some state elements have an empty reachability list for both values of v which we refer to as "island" state elements. More precisely, f is an island state element if $L_f^0 = L_f^1 = \emptyset$.

2) r_f**: Restorability rate of state element** f*:* This metric reflects the probability that a single state element f can be restored using the trace signals identified so far. The probability is computed within an observation window of M clock cycles by continuously calling the `XSim-core` procedure. While, in practice the observation window corresponding to the trace buffer depth is 1K to 8K cycles, it has been shown in [3] that a much smaller observation window (e.g., M=64 cycles) provides sufficient accuracy for evaluations within the simulation-based procedure. Similarly, we use $M = 64$.

Algorithm 3 shows the details of computing the restorability rate for all the state elements. The inputs are the set of state elements which are not selected so far (denoted by F), input control signals (denoted by $\mathcal{S}_C$), and the trace signals so far within an observation window of M cycles (denoted by $\mathcal{S}_T^1, \ldots, \mathcal{S}_T^M$). Recall $s = (p, v, n) \in \mathcal{S}_T^n$, if $v \neq x$ and $p \in \mathcal{P}_F$ indicating that s corresponds to a state element which is traced in cycle n. The output is the restorability rate of state element f, denoted by r_f. One call to the algorithm is sufficient to compute r_f for *all* $f \in F$.

According to Algorithm 3, initially r_f is set to 0 $\forall f \in F$. Next, the set of control signals $\mathcal{S}_C$ is used to identify a set of restored signals denoted by $\mathcal{S}_{R_C}$ using the `XSim-core` procedure. (See line 2.) At each step, the `XSim-core` procedure is used to identify the signals which can be restored at cycle n for $n = 1, 2, \ldots, M$.

Note in line 4, the input to XSim-core is $\mathcal{S}_T^n \cup \mathcal{S}_{R_C}$ which allows accounting for the impact of the control signals in addition to the trace signals in that cycle. The restored set is denoted by $\mathcal{S}_R$. If a state element $f \in F$ can be restored, then its corresponding signal is in $\mathcal{S}_R$ and r_f is added by 1. In the end, r_f is returned as a probability by dividing it by M for each $f \in F$.

The input trace signals $\mathcal{S}_T^n$ for $n = 1, 2, \ldots, M$, for any call to this function, are computed using a one-time simulation done in the initial pre-processing step of the flow shown in Figure 2. Specifically, the circuit is simulated for 1K cycles and the values of all state elements for each cycle are stored in the initialization step. Next, at each call of Algorithm 3, the values of the corresponding trace signals are looked up from the stored patterns for the M cycles. Specifically, given the small size of the observation window, i.e., $M = 64$, we randomly select 3 non-overlapping observation windows with different starting cycles from the 10K simulated cycles. For each window, Algorithm 3 is called once and the final stored value of r_f is computed as the average of these three values. (These 3 runs of Algorithm 3 are implemented as 3 threads running on a multi-core machine.) This idea of averaging over multiple computations was also used in [3] but for computing the SRR when M is small. In addition, the calls to the XSim-core procedure are made by setting the restore-once flag to false. Recall by setting this flag to false, all the possible restorable signals will be identified. So it allows exact evaluation if f can be restored by the current trace signals.

3) $d_{i,f}^v$**: Demand of state element** i **from state element** f **taking value** v*:* If state element i is not fully restored (i.e., $r_i < 1$) we would like to quantify its demand if it is restored by another state element f. We define $d_{i,f}^v$ as the demand of i to get fully restored by f when f has a known value.

In practice, state element f which allows restoring state element i, only falls within a small subset of the entire set of state elements and considering f to be among the set of all the state elements results in many unnecessary and time-consuming computations. Therefore, we limit f to be a state element which includes i in its reachability list, for either value 0 or 1 (i.e., $i \in L_f^v$, $v \in \{0, 1\}$). We approximate the demand $d_{i,f}^v$ as follows.

$$d_{i,f}^v \approx \min(1 - r_i, a_j^v), \ \forall i \in L_f^0 \quad or \quad i \in L_f^1 \tag{1}$$

where a_f^v is the probability that state element f takes value v. The probability a_f^v is accurately computed in the initialization step using circuit simulation for a suitable number of clock cycles (e.g., 10K in this work with random values for non-control input vectors). In Equation 1, the quantity $1 - r_i$ reflects the remaining restoration demand of i. If it is larger than a_f^v, then the demand is given by a_f^v which is the likelihood that f takes value v. Equation 1 is an upper bound approximation which can be computed fast. Otherwise, accurate computation of the demands requires many time-consuming simulations and would be impractical for realization of a fast and scalable algorithm.

4) w_f**: Impact Weight of state element** f*:* The Impact Weight of a state element f captures the amount of restoration if f is selected as the next trace signal. A key point in computing this Impact Weight is considering the remaining restoration of the unrestored state elements which is given by the restoration demand metric. Specifically, the Impact Weight of state element f is defined as follows.

$$w_f = \sum_{v=0,1} \sum_{\forall i \in L_f^v} d_{i,f}^v \tag{2}$$

In the above equation, the corresponding demands of the state elements in the reachability lists of f for values 0 and 1 are added. The higher Impact Weight for a state element f can be an indication that more state elements can be restored if f is selected as the next trace while accounting for the amount of restoration using the already-selected trace signals.

As an example, in Figure 1, the Impact Weight of flipflop f_2 is given by $w_2 = d_{1,2}^0 + d_{1,2}^1 + d_{3,2}^1 + d_{5,2}^0$. At the beginning when no trace signal is selected, the restoration rates of all the flipflops are 0. Therefore the demand $d_{j,2}^v = a_j^v$ according to Equation 1. Assuming the two primary inputs of this circuit are independent and each has a probability of 0.5 to be 0 or 1, we obtain the probability rates $a_1^0 = a_1^1 = 0.5$, $a_3^1 = 0.75$, $a_5^0 = 0.75$, and $w_2 = 2.5$.

B. Steps of the Algorithm

We now discuss the details of different steps of our algorithm. These are shown in bold in Figure 2.

1) **Initialization***:* In this step, first, the circuit is simulated for 10K cycles using random values for non-control primary input vectors. The simulation results are used to compute the probability a_f^v for each state element. As mentioned before, they are also used to provide the trace signals which are fed as inputs to Algorithm 3 to compute r_f for each state element f. Next, the demands and the Impact Weights are computed, similar to the given example.

2) **Method (i) trace selection using the restoration demands***:* At each step of the algorithm, method (i) is initially used to identify the next trace signal. In general, a state element with a higher Impact Weight is a better candidate for the next trace signal. However, simply selecting the state element with the maximum weight may not be a good choice because based on our observations, there may be other state elements with similar yet slightly smaller weight values which may result in a higher state restoration ratio (SRR). Therefore we evaluate the top $k\%$ of the state elements with the highest restoration demands. To select the next trace signal from the above identified subset, we consider adding each to the current set of selected trace signals and directly measure the SRR for an observation window of $M = 64$ cycles. (See Section II-B for computation of SRR.) The next trace signal is the one which yields to the maximum SRR.

Since computing SRR involves X-Simulation, the parameter k should be set to a small value to ensure the runtime of the algorithm is feasible. In our implementation of method (i), we identify the top 5% of the state elements with the highest weight. We observed that this value is reasonable to identify the state elements with high weight values, and yet is small to ensure a negligible runtime overhead.

3) **Method (ii) trace selection with island consideration***:* Recall from Section III-A1 that an island state element has empty reachability lists, for both values of 0 and 1. It means that stand alone, an island state element is not able to restore any other state elements. Therefore as shown in Figure 2, after selecting every 8 trace signals, we consider adding an island signal. For example, for a typical trace buffer bandwidth of 64 bits, adding an island is considered seven times throughout the course of the algorithm.

Specifically, to select an island signal, we simply add each island signal individually to the current list of selected trace signals and measure the SRR for an observation window of 64 cycles. This is because we observed the number of islands are typically very low and consequently the runtime overhead to compute the SRRs is not significant. Once the SRRs are computed, the island with the maximum SRR is identified and if the SRR is higher than a threshold, then the island is also added to the set of trace signals. In that case, within one step of the algorithm two trace signals will be added to the set. However, if an island is not selected, adding an island will be postponed when eight additional trace signals are identified.

TABLE I
RUNTIME COMPARISON OF DIFFERENT ALGORITHMS

		METR: Metric-based [6]			SIM: Simulation-based [3] (implemented on a quad-core machine)			HYBR: Hybrid		
	#FFs	Trace Size			Trace Size			Trace Size		
Benchmark		8	16	32	8	16	32	8	16	32
S5378	163	8	27	66	00:06:50	00:06:40	00:05:30	5	27	28
S9234	145	6	17	38	00:07:28	00:06:05	00:04:10	26	84	86
S13207	327	48	117	254	00:48:12	00:46:42	00:41:3	68	163	166
S15850	137	7	18	37	00:02:34	00:02:03	00:00:40	83	193	197
S35932	1728	73	167	408	07:13:00	07:12:00	07:11:00	139	208	217
S38417	1564	3690	7620	13428	50:05:00	50:04:00	50:02:00	434	2508	2521
S38584	1166	53	140	354	16:33:00	16:32:00	16:31:00	167	741	752

4) **Updating the Impact Weights***:* Method (i) relies on using the Impact Weights corresponding to the most recent set of trace signals. Therefore, at each iteration of the loop in Figure 2, these weights are updated. Specifically, the core metric to be updated is the restorability rate of each state element which is used to compute the demand in Equation 1 and the weight in Equation 2. In order to update the r_f values, Algorithm 3 is called which now takes the new sets of trace signals as inputs as explained in Section III-A2. Note, the reachability lists do not change after the initialization step.

We further discuss the complexity of some of the steps for computing/updating the weights. First, updating the demands and the weight for one state element can be done in constant time once the restorability rate of state elements (r_f) are updated. This can be observed from Equation 1. For the weight given by Equation 2, in practice we observe a constant computational complexity because each state element is only contained in the reachability list of a few number of state elements, much smaller than the total number of state elements in the circuit. The computational complexity is dominated due to updating the r_f values however this only requires calling `XSim-core` procedure $M = 64$ times in Algorithm 3 for all the untraced state element.

IV. SIMULATION RESULTS

Our trace selection algorithm, which we refer to in short, as HYBR in our experiments, was implemented in C++. It was tested on the ISCAS89 benchmarks which were synthesized using Synopsys Design Compiler with a 90nm TSMC library for trace buffers of various bandwidths. The number of flipflops for each benchmark is reported in column 2 of Table I.

To measure the solution quality, the State Restoration Ratio (SRR) with an observation window of 4096 cycles is used with random values for non-control primary inputs. Note, this observation window is the size that can typically be captured by a trace buffer, and is also assumed in all the previous works. The procedure to calculate SRR was explained in Section II-B.

Furthermore, two of the benchmarks (`S35932` and `S38584`) have control signals as primary inputs to the circuit. The names and values of these control signals are as follows. For `S38584` we identify 'g35' as an active low global reset so it was set to 1 which is also pointed out in [4]. For `S35932`, the active low global reset signal 'RESET' was set to 1. Moreover, two control input signals 'TM0' and 'TM1' define four working modes in this benchmark. Therefore, we ran this benchmark four times for each of the working modes and measure four separate SRR values. We then report the average of these four SRR values for `S35932` in our experiments.

We also make **comparison with other trace selection algorithms**. We note that due to the technology library used for synthesis of the benchmarks in our simulations, direct comparison with other existing algorithms by looking at the reported SRR values from the related publications was inaccurate. Therefore, we also implement the following two trace selection algorithms for comparison: 1) METR: metric-based [6][1], and 2) SIM: simulation-based [3]. METR uses a metric to *approximate* the SRR while SIM directly uses simulation to accurately compute SRR for trace signal selection throughout the course of the algorithm. As a result, METR is typically much faster than SIM. However SIM is shown to result in a much higher solution quality. We use METR mainly as a reference for runtime, and SIM mainly as a reference for the solution quality of our algorithm. We also note that among the metric-based algorithms, we select [6] due to its fast execution runtime which is similar to [5] but faster than [4] and [2]. Both [6] and [3] use the XSimulator in their internal procedures and for fair comparison, we use the same procedure (i.e., `XSim-core` given by Algorithm 1) which provides the most efficient implementation. Our implementation of SIM included the pruning phase given in [3] with random input trials equal to 3. We also note that our implementation of `XSim-core` exploits bitwise parallelism for state restoration given in [4]. When implementing [3], custom parameter selection was done the same way as reported in [3]. All simulations ran on an Intel quad-core 3.4GHZ with 12GB memory.

Comparison of Runtime: Table I shows the comparison of runtime of our HYBR with SIM and METR algorithms for three buffer bandwidths of 8, 16, and 32 bits and a buffer depth of 4K. The reported runtimes are in seconds except for our implementation of [3] where the reported format is (hour:minute:sec). We note, the work [3] describes a GPU-based implementation of SIM which exploits a high degree of parallelism. However, our implementation of [3] which ran on a quad-core CPU is based on multi-threading and only up to 8 parallel threads could run simultaneously in our setup. Therefore, our reported numbers for SIM is higher than [3]. But anyhow, it gives a measure to highlight the significant speedups using HYBR.

As can be seen, HYBR has a comparable runtime to METR and is tremendously faster than SIM. Moreover, we note that metric-based algorithms are already verified to be much faster than simulation-based procedures, even when a GPU-based implementation is used, as reported in [3]. So we expect that HYBR is also much faster if a GPU-based implementation of SIM is used.

To analyze the fast runtime of HYBR and compare it with SIM, we compare the number of calls to the `XSim-core` procedure in both algorithms. This step is called repetitively and is the most time-consuming step in both cases. The analysis is as follows. In HYBR, at each step, computation of restoration ratio for *all* the state elements ($r_f\ \forall f \in F$) using Algorithm 3 requires a total of $M = 64$ calls to the `XSim-core` procedure. Furthermore, at each step of HYBR, we use SRR computation for the top 5% of state elements which have the highest Impact Weights. Each SRR computation requires 64

[1]For fair comparison, no critical state elements was specified in [6].

TABLE II
COMPARISON OF STATE RESTORATION RATIO (SRR) OF DIFFERENT ALGORITHMS

	METR: Metric-based [6]			SIM: Simulation-based [3]			HYBR-NOSIM: Hybrid w/o simulation for top candidates			HYBR: Hybrid with simulation for top candidates		
	Trace Size			Trace Size			Trace Size			Trace Size		
Benchmark	8	16	32	8	16	32	8	16	32	8	16	32
S5378	13.7	8.1	4.1	12.8	7.1	4.4	13.4	7.9	4	13.6 (-0.7%)	8.0 (-1.2%)	4.2 (-4.5%)
S9234	8.4	5.8	3.4	9.1	6.6	3.6	9.4	6.1	3.3	9.8 **(+4.3%)**	6.8 **(+3.0%)**	3.6 **(+0.00%)**
S13207	13.8	6.8	3.5	19.3	12.2	7.8	22.2	14.6	8.0	24.5 **(+10.4%)**	16.3 **(+11.6%)**	8.9 **(+11.3%)**
S15850	14.4	7.6	4.1	14.5	7.8	4.1	15.0	7.8	4.0	15.6 **(+4.0%)**	8.1 **(+3.8%)**	4.1 **(+0.00%)**
S35932	31.1	19.4	11.6	58.1	36.2	23.1	31.6	18.9	11.3	61.4 **(+5.7%)**	38.3 **(+5.8%)**	23.4 **(+1.3%)**
S38417	17.6	13.1	9.7	29.4	17.8	20.0	18.1	10.3	5.9	51.3 **(+74.5%)**	30.1 **(+12.9%)**	17.5 (-12.5%)
S38584	13.5	10.8	7.1	14.9	18.1	16.4	18.3	14.8	10.7	24.0 **(+31.1%)**	18.5 **(+2.2%)**	17.5 **(+6.7%)**

calls to the `XSim-core` procedure. Therefore the number of calls to `XSim-core` at each step of HYBR is dominated by the number of SRR computations which is at most 5% of the number of state elements and is a small number. In contrast, in SIM, at each step and for each untraced state element, an SRR is computed. Therefore the number of SRR computations are significantly larger.

While each step of HYBR is significantly faster than SIM, we also note that another reason for the difference in the runtimes is because the number of steps in SIM is much larger than HYBR. This is because SIM is based on elimination of the least promising state elements and the number of state elements are often much higher than the number of trace signals. For example, `S38584` has 1166 state elements which is much higher than the number of trace signals.

Comparison of Solution Quality: For this experiment, we report the results for a new variation of HYBR. Specifically, in this variation, when selecting the next trace signal using Method (i), we skipped the SRR-based evaluation of the top 5% state elements with the highest Impact Weights. Instead, we directly selected the state element with the maximum Impact Weight in order to measure the effectiveness of the Impact Weight metric. Recall the eliminated step for SRR evaluation involved simulation so we refer to this variation as HYBR-NOSIM.

Table II shows the comparison of the solution quality (i.e., SRR). (The SRR values are computed for an observation window which was set to M = 4K cycles corresponding to the buffer depth.) For HYBR, a percentage improvement in SRR is also reported which is with respect to the SRR of the remaining techniques listed per benchmark. As can be seen, HYBR results in a significantly higher solution quality compared to METR. This is for the majority of the benchmarks except the smallest one (`S5378`) in which the quality of solution is quite similar in all the algorithms.

Furthermore, HYBR has a consistently higher solution quality compared to SIM for small buffer bandwidths (i.e., 8 and 16 bits). For example in benchmark `S38417` and for the buffer bandwidth of 8 bits, the SRR of HYBR is 51.3 while the SRR of SIM is 29.4. For the bandwidth of 32 bits, the two algorithms have a quite similar SRR. The main reason that HYBR performs better for smaller bandwidths is because it is based on selecting the most promising state element at each step while [3] is based on eliminating the least promising one at each step. Therefore in [3] the error associated with the greedy backward elimination of state elements grows as the buffer bandwidth decreases. In contrast, in HYBR, the error associated with greedy forward addition of promising state elements grows with the increase in the buffer bandwidth.

When comparing HYBR and HYBR-NOSIM, we observe that by eliminating the simulation for the top trace candidates, the solution quality significantly degrades. For the first four benchmarks HYBR-NOSIM often maintains a better solution quality over SIM, however SIM outperforms HYBR-NOSIM for the remaining benchmarks. After further examining the first four benchmarks, we found out that the initially-generated reachability lists allow a more effective computation of the Impact Weights for these benchmarks. (Equation 2 shows direct dependency on the reachability lists.) This can be due to their smaller sizes or circuit topologies.

We conclude that simulation-based measurement of SRR provides a better method for evaluation of the *top* candidates of our algorithm compared to the Impact Weights. However, we note this SRR-based evaluation is only done for a small percentage of the state elements which are quickly identified using the Impact Weight metric. (In contrast [3] utilizes SRR-based evaluation for all the candidate state elements.) From the above arguments, we can conclude our Impact Weight metric is able to effectively and quickly identify the top candidates. This directly translates into significant reduction in the number of SRR-based evaluations and the portion of runtime spent on simulation while maintaining the high solution quality.

V. CONCLUSIONS

We presented a hybrid trace signal selection algorithm. In terms of solution quality measured by the state restoration ratio, it is comparable or better than a simulation-based algorithm which had the best solution quality among the existing approaches. In terms of runtime our hybrid algorithm is significantly faster than simulation-based algorithm and has a runtime comparable to metric-based algorithms which were shown to be the fastest among the existing approaches. The impressive runtime and solution quality of our hybrid algorithm is due to developing the right blend of simulation with a new set of proposed metrics which can be quickly evaluated. Together, they allow fast and effective measurement of the impact of each candidate state element which allows for their comparisons in order to select the next trace signal throughout our algorithm.

REFERENCES

[1] M. Abramovici, P. Bradley, K. N. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A reconfigurable design-for-debug infrastructure for SoCs. In *DAC*, pages 7–12, 2006.
[2] K. Basu and P. Mishra. Efficient trace signal selection for post silicon validation and debug. In *VLSI Design*, pages 352–357, 2011.
[3] D. Chatterjee, C. McCarter, and V. Bertacco. Simulation-based signal selection for state restoration in silicon debug. In *ICCAD*, pages 595–601, 2011.
[4] H. F. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *IEEE TCAD*, 28(2):285–297, 2009.
[5] X. Liu and Q. Xu. On signal selection for visibility enhancement in trace-based post-silicon validation. *IEEE TCAD*, 31(8):1263–1274, 2012.
[6] H. Shojaei and A. Davoodi. Trace signal selection to enhance timing and logic visibility in post-silicon validation. In *ICCAD*, pages 168–172, 2010.
[7] J.-S. Yang and N. A. Touba. Efficient trace signal selection for silicon debug by error transmission analysis. *IEEE TCAD*, 31(3):442–446, 2012.
[8] Y.-S. Yang, A. G. Veneris, and N. Nicolici. Automating data analysis and acquisition setup in a silicon debug environment. *IEEE TVLSI*, 20(6):1118–1131, 2012.