RATS: Restoration-Aware Trace Signal Selection for Post-Silicon Validation

Kanad Basu, *Student Member, IEEE*, and Prabhat Mishra, *Senior Member, IEEE*

***Abstract*— Post-silicon validation is one of the most important and expensive tasks in modern integrated circuit design methodology. The primary problem governing post-silicon validation is the limited observability due to storage of a small number of signals in a trace buffer. The signals to be traced should be carefully selected in order to maximize restoration of the remaining signals. Existing approaches have two major drawbacks. They depend on partial restorability computations that are not effective in restoring maximum signal states. They also require long signal selection time due to inefficient computation as well as operating on gate-level netlist. We have proposed a signal selection approach based on total restorability at gate-level, which is computationally more efficient (10 times faster) and can restore up to three times more signals compared to existing methods. We have also developed a register transfer level signal selection approach, which reduces both memory requirements and signal selection time by several orders-of-magnitude.**

***Index Terms*— Post-silicon validation, restoration, trace buffer, trace signals.**

I. Introduction

FUNCTIONAL validation is one of the most important tasks in integrated circuit (IC) design due to the combined effects of increasing design complexity and reduced time-to-market. Pre-silicon validation is the first step to detect bugs, which uses a combination of formal validation and simulation-based techniques. Since many physical parameters cannot be modeled correctly, a lot of errors escape the pre-silicon phase and affect the normal operation of the chip. This makes post-silicon debug an important step in any design methodology. Post-silicon validation is used to capture these errors that have escaped the pre-silicon phase. Recent reports [1] indicate that at 65 nm, the industry spends almost 50% of its total design cost in post-silicon debug.

Post-silicon debug is comprised of signal observation and analysis. A primary problem for post-silicon debug is the limited observability of internal signal states. Although we can observe the input and output signals, it is not possible to observe the internal signals since the IC is completely fabricated. Some recent techniques like embedded logic analysis (ELA) can be used to probe into the chip and record some internal logic states. The entire procedure is shown in Fig. 1. The signals to be traced are selected during design phase. When tests are applied to the design under test (DUT), these signal states are traced using ELA. The trace is then dumped into an on-chip trace buffer using efficiently designed interconnection network, from where the data is transferred to an offline debugger via some Joint Test Action Group interface. More trace data can be stored using some online compression algorithm [2]. The debugger is able to point out the actual error locations in the DUT. To limit the size of the trace buffer, only a few signal states can be traced. The rest of the states are usually reconstructed from the internal logic of the circuit. Therefore, the signals to be traced should be selected carefully in order to maximize the restoration. Existing signal selection approaches have two major disadvantages. They utilize partial restorability and therefore, cannot ensure best possible signal restoration. Also, these methods are computationally inefficient since they require long signal selection time.

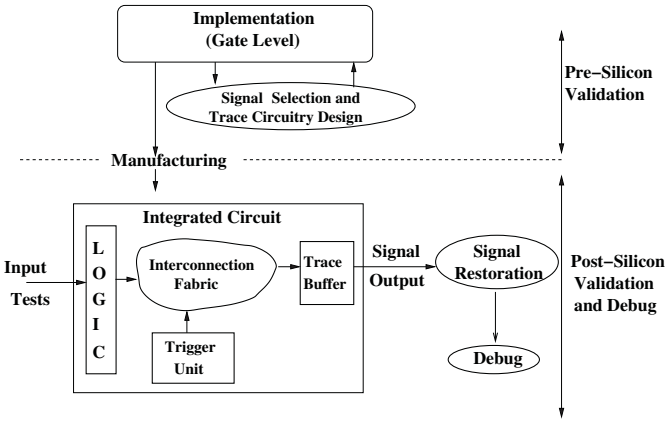


Fig. 1. Overview of the system validation and debug.

Signal selection techniques based on *partial restoration*[1] were proposed by Ko *et al.* [3] and Liu *et al.* [4]. If the trace buffer width is n, these approaches select n signals with highest partial restorabilities for tracing. As discussed in Section III, the partial restorability based approaches [3], [4] are not able to provide best possible signal reconstruction. This paper presents a *total restorability*[2] based signal selection algorithm to produce significantly better restoration performance compared to existing approaches. Our proposed method is also computationally more efficient than the existing approaches as shown in Section VI-B.

Manuscript received June 1, 2011; revised February 1, 2012; accepted March 3, 2012. Date of publication April 20, 2012; date of current version March 18, 2013. This work was supported in part by the National Science Foundation CAREER Award under Grant 0746261.

The authors are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611-6120 USA (e-mail: kbasu@cise.ufl.edu; prabhat@cise.ufl.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/TVLSI.2012.2192457

[1]Partial restorability of a signal refers to the probability that the signal state can be reconstructed using known states of some other traced signals.

[2]Total restorability measures whether a group of signals can definitely reconstruct a set of signal states.

The time requirement for gate-level signal selection (GSS) algorithms is high because of the excessive number of variables used to represent signals in the circuit. One promising alternative to reduce signal selection time is to perform signal selection at higher abstraction levels like register transfer level (RTL). We have proposed an efficient signal selection approach in RTL level. Our RTL-level signal selection (RSS) reduces both signal selection time and memory requirements with minor impact on restoration performance.

The rest of this paper is organized as follows. Section II presents related works in signal selection. Section III describes the signal selection problem using illustrative examples. Sections IV and V describe our signal selection technique in gate and RTL level, respectively. Section VI presents the experimental results. Finally, Section VII concludes this paper.

II. Related Work

A major problem concerning post-silicon debug is the limited observability of the internal signals. Once the signal states are known, they can be analyzed using some algorithms like failure propagation tracing [5] to identify the errors in the circuit. Formal analysis for post-silicon debug, proposed by De Paula [6], is not scalable to circuits with a large number of gates. Scan-based debugging techniques, such as [7] require to stop the circuit functionality when the scan data are being written. This is not beneficial in cases where the functional errors are drastically apart. Design-for-debug (DfD) techniques have been used extensively to increase the observability of internal signals of the silicon. Generally this is done by sampling the data, which is stored in on-chip trace buffers. Various DfD techniques like ELA [8] and shadow flip flops have been proposed over the years for post-silicon debug.

Recently, Ko *et al.* [3] and Liu *et al.* [4] have proposed a generic trace signal selection algorithms in which a few important signals can be traced and others can be reconstructed from them. Our proposed method is closest to their approaches and hence, throughout this paper, we have compared our proposed technique with their approaches.

RTL-level circuit analysis has already been used in different domains for solving design-related problems. An RTL fault grading approach was used to ameliorate the gate-level fault coverage by Mao *et al.* [9]. RTL-level tests were generated and reused for detecting gate-level stuck-at-faults by Yogi *et al.* [10]. Recently RSS algorithms were proposed by Ko *et al.* [11]. Similar to their approach, we use a control data flow graph (CDFG). However, there is a basic difference between the two approaches. While our proposed approach (Algorithm 2) works entirely on RTL-level description of a circuit for signal selection, [11] selects some signals from the RTL-level, and then the rest from the gate-level description of the circuit. Thus, both RTL-level and gate-level model of the design are necessary for the algorithm in [11] to select signals, while our proposed algorithm can operate with only the RTL-level model. Therefore, our proposed approach would be faster and still produce restoration performance comparable to gate-level netlists. We have proposed an RSS algorithm that is efficient both in terms of memory requirements and signal selection time compared to the existing state-of-the-art gate-level signal selection approaches.

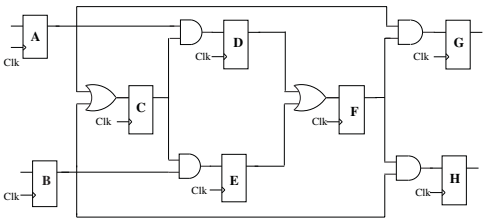


Fig. 2. Example circuit.

III. Background and Motivation

A. Signal Reconstruction

In post-silicon debug, unknown signal states can be reconstructed from the traced states in two ways-forward and backward restoration. Forward restoration deals with the restoration of signals from input to output, that is, knowledge of input states can provide the output. Backward restoration, on the other hand, deals with reconstructing the input from the output. Forward and backward restoration has been explained in detail in [3].

We now show by using a simple circuit how reconstruction is performed in [3] and [4]. An example circuit is shown in Fig. 2 having eight flip-flops. Let us assume that the trace buffer width is 2, that is, states of two signals can be recorded. We try to restore the other signal states by application of the methods presented in [3] and [4]. The results are shown in Table III-A. The "Xs" represent those states, which cannot be determined. The selected signals are shown in shades. Partial restorability calculations for both [3] and [4] are such that the signals selected are C and F, in that order. Restoration ratio, which is a popular metric of signal restorability is defined as

$$\text{Restoration Ratio} = \frac{\text{number of states restored + traced}}{\text{number of states traced}}.$$

Let us calculate the number of restored states in Table III-A ([3], [4]). If we consider the row corresponding to signal A, two entries have value 0, while the rest have value X (non-restored state). Thus, two states are known. Similarly, two states are known for the row corresponding to signal B. Since signal C is traced, all the states are known (no X in the row). For signal D, three entries in the row have value 0, hence three states are reconstructed. Computing in this manner, a total of 26 states are reconstructed. Out of them, ten entries (corresponding to signals C and F) are traced states. Therefore, Restoration Ratio $= 26/10 = 2.6$.

B. Motivational Example

We now employ our proposed method (described in Section IV) for selecting signals in the circuit in Fig. 2. The first signal that we trace is C. Note that this was the same signal that was chosen by [3] and [4]. The second signal that

TABLE I
RESTORED SIGNALS USING [3] AND [4]

Signal	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
A	X	0	X	0	X
B	X	0	X	0	X
C	1	1	0	1	0
D	X	X	0	0	0
E	X	X	0	0	0
F	0	1	1	0	0
G	X	0	0	X	0
H	X	0	0	X	0

TABLE II
RESTORED SIGNALS USING OUR METHOD

Signal	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
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	0	0

we choose is A, based on total restorability computations.[3] The results are shown in Table III-B. It can be seen that our method provides a restorability ratio of 3.2, which is better than [3], [4].

IV. GSS

Algorithm 1 shows our GSS that has five important steps. Edge and sequential element values are calculated in the first two steps. Total restorability computation is then used to create region and recompute sequential element values, accompanied by signal selection. The remainder of this section describes each of the steps in detail.

A. Computation of Edge Values

An edge between two sequential elements is the path taken to reach an element from another, while passing through a number of combinational gates between them, that is, there cannot be any sequential elements in between them. The edge may be in the forward or backward direction. In Fig. 2, an edge between the two flip-flops A and C passes through an OR gate. In a general case, there can be any number and type of combinational gates in an edge. To find the probability that C is influenced by A (which is the value of the edge AC), there can be two cases (independent and dependent) as discussed below.[4]

1) Independent Signals: Consider two edges AC and BC in Fig. 2. Here, the two input signals of the OR gate in front of flip-flop C are driven by flip-flops A and B, which are independent. Hence, the edges AC and BC are independent.

To calculate the edge values for an independent scenario, we use a generic example in Fig. 3. Later, we will show how the calculation works for the specific case in Fig. 2.

[3]Tracing A along with C gives a guarantee for restoring D, while F does not provide any such guarantees.

[4]We are showing calculations for forward restorabilities, however, those for backward restorabilities can be derived in similar lines.

Algorithm 1: GSS

Input: Circuit, Trace Buffer
Output: List of selected signals S (initially empty)
1: Compute the node-values(defined in Section IV-B) of all sequential elements.
2: Find the node with the highest value and add to S.
3: Create Initial Region.
while *trace buffer is not full* **do**
  4: Recompute the values of sequential elements.
  5: Compute Region growth by finding the sequential element with highest value not in S and add to S.
end
return S

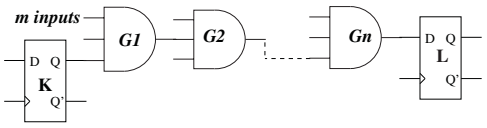


Fig. 3. Example circuit with n gates.

Fig. 3 has two flip-flops K and L. We want to find how the input of L is sensitized by the output of K. The input of L corresponds to the output of the gate G_n. The path from K to L is independent of any other paths through which the output of K propagates. Let's consider the gate G_1. We define four probabilities: ${P^I}_{0,N}$, ${P^I}_{1,N}$, ${P^O}_{0,N}$, and ${P^O}_{1,N}$. Here, ${P^I}_{0,N}$ indicates the probability that a node N (gate or flip-flop) has an input state of "0" when another node is controlling it. Similarly, ${P^I}_{1,N}$, ${P^O}_{0,N}$, and ${P^O}_{1,N}$ indicate the cases for input state of "1", output state of "0" and "1", respectively. The output of flip-flop K can influence the output of G_1 in two cases: 1) output of K is a controlling value and 2) all the inputs to G_1 are complement of the controlling value. Let us consider G_1 to be a two-input AND gate. We define P_{G_1} as the overall probability of K controlling G_1. According to [12]

$$P_{G_1} = {P^O}_{1,G_1} + {P^O}_{0,G_1}. \tag{1}$$

Now, let's define ${P^O}_{0,G_1}$ and ${P^O}_{1,G_1}$. Let $P_{\text{cond0,G}_1}$ and $P_{\text{cond1,G}_1}$ be the probability that the output of G_1 follows the output of K, i.e., the output of G_1 is 0(1), when the output of K is 0(1). For simplicity of calculation, in this example, we assume ${P^I}_{0,G_1} = {P^I}_{1,G_1} = 0.5$ (that is, occurrence of 0 or 1 follows equal probability at the input)

$${P^O}_{0/1,G_1} = P_{\text{cond0/1,G}_1} \times {P^I}_{0/1,G_1}. \tag{2}$$

Now, for a two-input AND gate, $P_{\text{cond0,G}_1}$ is 1, since 0 is the controlling input. Therefore, we obtain ${P^O}_{0,G_1} = 0.5$. Similarly, since one is the non-controlling input, $P_{\text{cond1,G}_1}$ is 0.5, which gives ${P^O}_{1,G_1} = 0.25$. From (1), it can be seen that $P_{G1} = 0.75$. Now, we return to our main goal, that is, to determine how K controls L. We first find the effect of the output from K as it propagates to the next gate G_2 and then extrapolate along the entire path to L. We use the same set of (1) and (2) again, except that the input is G_1 here and the output is G_2. Obviously, the values of ${P^I}_{0,G_2}$ and ${P^I}_{1,G_2}$ would be ${P^O}_{0,G_1}$ and ${P^O}_{1,G_1}$ obtained from (2). For

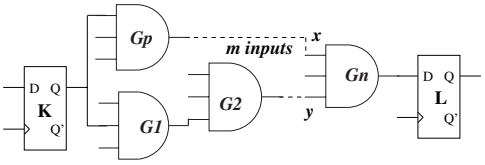


Fig. 4. Example circuit with dependent signals.

example if G_2 is also a two-input AND gate, applying (2), we obtain, $P^O{}_{0,G_2} = 0.5$, and $P^O{}_{1,G_2} = 0.125$. Therefore, we get $P_{G2} = 0.625$, where P_{G_2} is the probability for the gate G_2 defined in (1).

In this way, the calculation continues until we reach L, to obtain the value of the edge KL. If there are n combinational gates between K and L, we get

$$P^O{}_{0/1,G_n} = \prod_{1 \le i \le n} (P_{\text{cond0/1,G}_i}) \times P^I{}_{0/1,G_1}. \tag{3}$$

Finally, (1) is used to compute the probability P_{G_n}, which corresponds to the value of the edge KL.

We use these computations to show how an edge value is computed in case of the circuit in Fig. 2. Let's compute the value of edge AC. We name the OR gate in between the two as gate G and we assume that $P^I{}_{0,G} = P^I{}_{1,G} = 0.5$. Since it is an OR gate, $P_{\text{cond0,G}} = 0.5$ and $P_{\text{cond1,G}} = 1$. Therefore, (2) can be used to obtain $P^O{}_{0,G} = 0.25$ and $P^O{}_{1,G} = 0.5$. Equation (1) can now be used to obtain $P_G = 0.75$, which represents the value of the edge AC.

2) Dependent Signals: In case of dependent signals, we need to determine the probability of a sequential element output influencing an m-input gate, when the output of the sequential element affects l inputs ($l \ge 2$) of the gate.

We have used a generic example in Fig. 4 to calculate the edge value in case of dependent signals. It should be noted that dependent signals were not considered by [3] or [4].

Let's consider Fig. 4. It can be seen that two inputs (x,y) of the m input gate G_n are affected by flip-flop K. For this, our goal would be to combine the dependent edges so that the edge will have independent signals. We can then easily utilize the formula used in Section IV-A1 to compute the edge value. We desire to find $P^O{}_{1,G_n}$ and $P^O{}_{0,G_n}$, in lines with the parameter $P^{I/O}{}_{0/1,N}$ defined in Section IV-A1. Let us assume that G_n is an AND gate. For an AND gate, since 0 is the controlling value, having either of the inputs as 0 will ensure a 0 being propagated into the gate G_n. Therefore

$$P^I{}_{0,G_n} = P^O{}_{0,x} + P^O{}_{0,y} - P^O{}_{0,x\&y} \tag{4}$$

$P^O{}_{0,x\&y}$ subtracts the probability when both are 0, since it is being computed twice. Similarly, since 1 is the non-controlling input, we get

$$P^I{}_{1,G_n} = P^O{}_{1,x\&y} \tag{5}$$

where $P^O{}_{1,x\&y}$ is the probability when both x and y are "1". Let's evaluate the terms $P^O{}_{0,x\&y}$ and $P^O{}_{1,x\&y}$. Let $P_{\text{cond0/1,x/y}}$ be the probabilities that $x(y)$ is 0(1) when the output of K is 0(1). $P^O{}_{0/1,x\&y}$ can be defined as

$$P^O{}_{0/1,x\&y} = (P_{\text{cond0/1,x}} \times P_{\text{cond0/1,y}}) \times P^O_{0/1,K}.$$

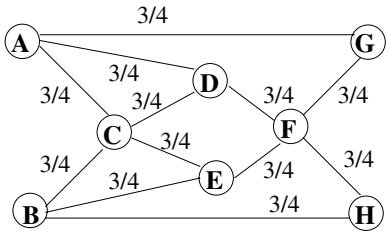


Fig. 5. Graphical representation of example circuit.

With the help of (2), this can be reduced to

$$P^O{}_{0/1,x\&y} = \frac{P^O{}_{0/1,x} \times P^O{}_{0/1,y}}{P^O{}_{0/1,K}}. \tag{6}$$

Since the paths from K to x and from K to y are assumed to be independent,[5] (3) can be used to obtain the values $P^O{}_{0/1,x/y}$. Application of (4) and (5) provides the values of $P^I{}_{0/1,G_n}$. The final P_{G_n} can be obtained using (1), and the information on the number of inputs to the gate G_n. This corresponds to the value of the edge KL.

3) Example: We now proceed to show how the calculations described in Sections IV-A1 and IV-A2 can be used to determine the edge values for the circuit in Fig. 2. A graphical representation of the circuit is shown in Fig. 5. Edge values are shown next to each edge.

The sequential elements are represented by nodes, and an edge between two sequential elements is represented by a straight line. It should be noted that there are no dependent edges in this example. All the edges have one two-input gate in between them, As a result, all the edge values are 3/4 (obtained from Section IV-A1). We will use this graph to explain our signal selection algorithm.

B. Initial Value Computation for Sequential Elements

We define the value of a sequential element as the sum of all the edges attached with it, in both forward and backward direction. For example, in Fig. 5, the value of flip-flop C is the sum of the weights of all edges connected with it, that is, CA, CB, CD, and CE. It is important to note that we have used a "threshold" in order to prevent combinational loops inside the circuit during edge value computation. This parameter was used by [3] as well.

Our computation of the sequential element values is independent of the sequential loops in the circuit. In a sequential loop, the output of a sequential element depends on another in both the previous and the next cycle. However, both cannot be true at the same clock cycle, that is, the same sequential element cannot determine the output of another in the same cycle by both forward and backward restoration. While forward restoration can determine the state in at least the next cycle, backward can determine it at most the previous cycle.

[5]If any one of these paths consists of dependent signals, the above procedure can be applied in a recursive manner until it becomes an equivalent independent path.

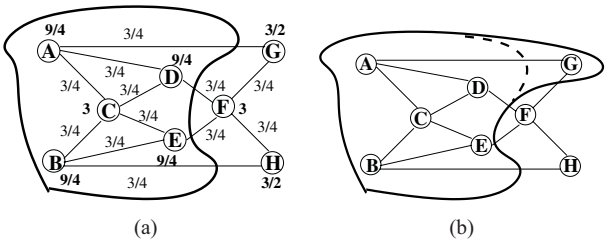


Fig. 6. Region creation and growth. (a) Initial region creation. (b) Region growth.

C. Initial Region Creation

A region is a collection of nodes attached together. It is not necessary that all the nodes have an edge with each other in the region. However, each node in the region must have at least one edge with another node in the region. In Fig. 5, the flip-flops $A-E$ form a region. The first node to be chosen is the one with the highest value, based on the calculations in Section IV-B. It is added to a list called "known." Now, all nodes which have an edge with the recently selected element are added to the region.

We show by an example in Fig. 6(a) how this portion of our algorithm is used to perform the selection of the profitable signals. The values of the nodes (addition of all it is edge values) are shown in bold alongside each node. For example, A has three edges AC, AD, and AG, each having a value 3/4. Therefore, the value for A is $(3/4)+(3/4)+(3/4)=(9/4)$. The node with the highest value in Fig. 6(a) is C. All the nodes which have an edge from C are included in the region. The region is represented by the spline in Fig. 6(a).

D. Recomputation of Node Values

The first node in Fig. 5 to be traced is already known (C in the previous example). However, there are other nodes that need to be traced as well. To select the subsequent nodes, their values are recomputed. The node whose value is being computed may have an edge to a node inside the region as well as one outside the region. Edges to nodes inside the region are given higher weight. As discussed in Section III, many restorability computations require knowledge of more than one signal of the input/output.[6] Therefore, it is better to gain more knowledge of the signals that are already in the region, thus increasing their restorability values and therefore, aiming for total restorability of those signals. Existing approaches [3], [4] recompute the restorability values after each iteration, which when translated to the graph in Fig. 5, would correspond to edge value recomputation, which is more computationally intensive.

E. Region Growth

The node with the highest restorability and not in the list "known" is determined. If two nodes have the same value, the one with the higher forward restoration is traced. This is because, backward restoration fails in some cases whereas forward restoration does not when all the inputs are known. For example in Fig. 6(a), the next node to be traced is A. It is included in the list "known." If the trace buffer is already full, calculations will stop, otherwise the region is continued to grow. All nodes having an edge to the recently selected node are added in the region. As shown in Fig. 6(b), in this case G is added since G is the only node connected to A and not in the region. The dotted line indicates the original region. Next, recomputation of node values as in Section IV-D is reconsidered and this process is iterated until the trace buffer is full. "Region growth" is found to be distributed uniformly across the entire circuit, and not clustered in a single area.

[6]For example, when all the inputs to a gate are complement of the controlling value.

F. Complexity Analysis

In this section, we compute the complexity of our algorithm. Let V be the number of nodes in the circuit, and E be the number of edges in the circuit. Let N be the number of signals to be traced, that is, the size of the trace buffer is N. The first step, that is, edge value computation takes $O(E)$ time, while flip-flop value computations for each time, a signal is selected takes $O(V)$ time. To select N signals, the time required is $O(NV)$. Therefore, the overall time complexity of our algorithm is $O(E+NV)$. On the other hand, the time complexity of existing algorithms is $O(NE)$. Since, $E >> V$, the time complexity of our proposed algorithm is less. The overall space complexity of our algorithm is $O(E+N+V)$. Since, $E >> N+V$, the space complexity reduces to $O(E)$.

V. RSS

To show how signal reconstruction can be efficiently performed in RTL level, let us consider the following Verilog design in Fig. 7(a). The design consists of three register-variables namely a, b, and c (each corresponds to a set of flip-flops) as well as two input signals d and e. There are also three other signals $m1$, $m2$, and $m3$. In the example, a and b are eight bits long, c and e are of seven bits, while d is just a one-bit signal. To show how reconstruction is performed, let us observe how each of these flip-flops is assigned—a is the concatenated value of d and c, b is the result of logical operations between a, $m1$, $m2$, and $m3$ while c attains the sum of an arithmetic operation between e and a constant number. Let us assume that we trace the value of a. We now explain how tracing of a in cycle k helps us to reconstruct the other states. The assignment of b shows that the state of b in cycle $k+1$ can be reconstructed from the state of a by forward restoration. From the assignment of a, the states of c and d in cycle $k-1$ can be reconstructed from state of a by backward restoration. Finally, from the last statement, that is, the assignment of c, state of e can be restored in cycle $k-2$ by backward restoration. Thus, we see that tracing of only one state of a can reconstruct the states of four other variables in different cycles.

Algorithm 2 shows our signal selection procedure that has six important steps. In the first step, a CDFG is generated to model the entire system. Since in the RTL description, each register variable represents multiple sequential elements, we

Algorithm 2: RSS

Input: RTL description of design, No. of trace entries
Output: List of selected signals S (initially empty)
1: Develop the CDFG of the RTL description.
2: Find the *relationship* between the register variables.
3: Find the initial values of the register variables.
while *trace buffer is not full* **do**
  4: Find the register variable with the highest value.
  5: Add all sequential elements corresponding to it to the list S.
  6: Recompute values for all the register variables.
end
return S

use the register variables for signal selection. However, the trace buffer width refers to the total number of sequential elements represented by these register variables. For example, the register variable $[7:0]$ a represents eight sequential elements, and therefore selection of variable a implies that eight trace buffer locations are needed. The relationship between the different register variables is obtained from the CDFG. These relations are used to produce the total restorability values for the variables. The register variable with the highest value is chosen for tracing. Once a variable is chosen for tracing, all the other variable values are recomputed in the same manner as in Algorithm 1. Steps $4-6$ are continued until the trace buffer is full. The remainder of this section describes each of the steps in detail.

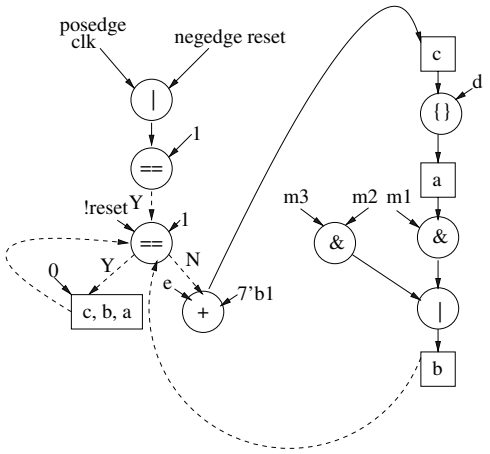
A. CDFG Generation

The first step of RTL level signal selection is to generate the CDFG from the RTL model. CDFG can be generated using any standard HDL parser. For our use, we have generated the CDFG by modifying the open source Icarus Verilog parser [13] for the Verilog circuits. Although, our studies are based on Verilog benchmarks, our approach is also applicable for VHDL designs. The format of our CDFG representation is similar to Mohanty *et al.* [14].

Fig. 7(b) shows the CDFG representation of the Verilog code in Fig. 7(b). The CDFG can represent both the movement of control signals as well as data values. The dotted arrows indicate the control-flow (transitions) in the CDFG, while the bold arrows represent the data flow (computations). For example, in the right-hand side of Fig. 7(b), there is a bold arrow from a to the AND gate. This is because a is an input of the AND gate. The circles in the CDFG represent computation and control nodes, while the boxes represent storage nodes. For example, the circle in the top represents an OR assignment for the conditional statement *always*, while the square at the bottom right represents the storage in the node b. It should be noted that direct assignments like $a <= 7'b0$ are just represented as a bold arrow with value 0 entering a box for storage of value a. In this case, since three variables a, b, and c are all being assigned 0 together, they are grouped in a single box. This basic representation can be further extended to represent the CDFG of a complex design. This CDFG

```
always @ (posedge clk or negedge reset)
begin
if (!reset)
 begin
  a <= 7'b0;   b <= 7'b0; c <= 7'b0;
end
else
 begin
  a <= {d, c}; b <= (a & m1) || (m2 &
m3); c <= e + 7'b1;
 end
end
```

(a)



(b)

Fig. 7. Verilog code and CDFG. (a) RTL Verilog example. (b) CDFG of Verilog code.

representation is used as input for the next step, relationship computation.

B. Relationship Computation

The relationship of a signal with others can be obtained from the CDFG. The relationship computation for a signal in the circuit provides the effect of that signal on others. To compute the relationship of the signals, we first note that there can be two main *relationship* types, namely *direct relationship* and *conditional relationship*. These two classes and their respective relationship computations are explained as follows.

1) Direct Relationship: Two signals are said to be directly dependent when they occur on the same line of a signal assignment. For example, in the RTL description shown in Fig. 7(a), the signal pairs (a, b) and (a, c) have direct relationship. This is because both the variable assignments occur inside the if block. Direct relationship can be of two types, namely $forward$ and $backward$ relationship. Forward relationship deals with the propagation of values in the forward direction, that is from the right-hand side of the assignment to the left-hand side. Backward relationship on the other hand deals with the reverse, that is from left-hand to right-hand side of the assignment. For example, in

the RTL description of the example in Fig. 7(b), a has a forward relationship on b, while b has a backward relationship on a. We will use a simple generic example to show how the direct relationship computation is performed. Later, we will consider a specific example shown in Fig. 5(a). A typical signal assignment statement looks like

$$y <= x_1\ OP_1\ x_2\ OP_2\ x_3\ OP_3 \ldots\ x_n$$

where OP represents any operation (e.g., AND, OR, etc.). We can see that there are n signals on the right-hand side of the assignment statement. We want to find out the relationship of each of these signals on y. Let us assume that each of these signals is k bits long and all the x_is are independent. We also assume that each of the OP_is are AND gates. Therefore, the assignment statement can be rewritten as

$$y <= x_1\ \&\ x_2\ \&\ldots\&\ x_n.$$

The same computation can be extended to other operations, as well as different operations for each OP_i. Let us compute the relationship of x_i $(1 < i < n)$ on y. The relationship of x_i on y is computed as a probability that y completely follows x_i. Therefore, the relationship can be found in the same way as the independent edge value computation in Section IV-A1. Essentially, the relationship of y and x, $P^y{}_{0/1,x_i}$ is equivalent to $P_{\text{cond}0/1,y}$ in (2). It should be noted that y follows x_i completely when either all the k bits of x_i are 0 (P_0), or when all the x_is have their k bits as 1 (P_1).[7] In all other cases, some of the k bits of y are different from x_i. The relationship of y on x_i is given by

$$P^y{}_{0,x_i} = \frac{2^{k\times(n-1)}}{2^{k\times n}} \tag{7}$$

$$P^y{}_{1,x_i} = \frac{1}{2^{k\times n}}. \tag{8}$$

According to [12] and assuming for ease of illustration[8] that 0 or 1 can occur with equal probability, we get

$$P^y{}_{x_i} = \frac{2^{k\times(n-1)}+1}{2^{k\times n}}. \tag{9}$$

The numerator on the right-hand side consists of two terms. The first term corresponds to the case when all the bits of x_i are 0. Therefore, each of the k bits of the other $n-1$ variables can have $2^{k\times(n-1)}$ values. The last term, 1 on the numerator, corresponds to the case when all the bits of all the x_is are 1. The denominator denotes all the possible cases of value assignments to the x_is. These calculations, as stated above, can be extended for other operations as well. For example, if OP was an OR operation, (7) and (8) will be modified as

$$P^y{}_{1,x_i} = \frac{1}{2^{k\times n}} \tag{10}$$

$$P^y{}_{0,x_i} = \frac{2^{k\times(n-1)}}{2^{k\times n}}. \tag{11}$$

In this section, we have described the direct relationship when the signals are independent. However, similar to Section IV-A2, we can have dependent signals as well. The nature of dependent signals is derived from multiple branches of the CDFG. Computations for dependent signals are similar to the computations in Section IV-A2.

2) Conditional Relationship: Conditional relationship corresponds to the non-assignment dependencies. For example, in the RTL code corresponding to Fig. 7(b), the signals a and b have conditional relationship on $reset$.[9] We generally do not consider backward conditional relationship, since, these are not in direct assignment statements. Conditional relationship are computed in the same way as in (7), however the operations are checked inside the conditional block. For example, we consider the following codes:

$$if(m\ or\ n)\ \ x <= y.$$

Here, the symbol x has a conditional dependence on m and n. Since there are only two variables m and n in the conditional dependency, the dependency value is $3/4$, as obtained from (1) and (2) in Section IV-A1. The conditional relationships are computed in this manner for all the signals in the circuit.

C. Signal Selection

Once the values of the variables are computed, the next step is to select the best one for tracing. The signal selection procedure is similar to the gate level signal selection. The variable with the highest value is selected and the rest of the values are recomputed using region growth. This part is similar to Algorithm 1 and hence not discussed here. The process continues until the trace buffer is full.

VI. Experiments

In this section, we first compare our approach with existing GSS techniques [3], [4]. Next, we demonstrate how our proposed RSS can further improve the signal selection time.

A. Experimental Setup

We applied our GSS approach on the ISCAS'89 benchmarks used by [3] and [4] to compare with their methods and hence show the effectiveness of our algorithm. The trace buffers used are same as that of [3], that is, $8\times4k$, $16\times4k$, and $32\times4k$. We have designed a simulator in the lines of the one described by [4] for our purpose, which conducts simulation in both forward and backward direction. We have implemented the simulator as an iterative process, which terminates when it is not possible to restore any more states. We have fed the simulator with ten sets of random values and noted the average restoration ratio.

Fig. 8 give an overview of our experimental setup to validate the RSS algorithm. For this purpose, we have used Verilog circuits obtained from Opencores website [15]. It should be noted that we have not used the ISCAS'89 benchmarks since an RTL description of these was not available. We have modified the Icarus Verilog parser [13] to generate the CDFG.

[7] Since 0 is the controlling input of AND gate.

[8] In actual experiments, we have used a profiling information to determine the probabilities.

[9] It should be noted that we do not consider conditional relationship of general control signals like clock (clk) or $reset$.

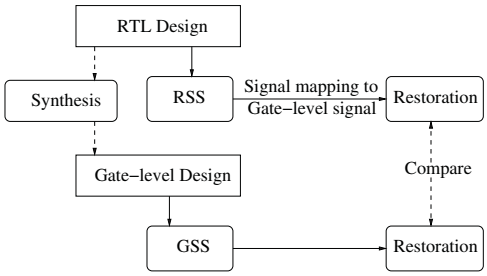


Fig. 8. Overview of our experiments to verify RSS.

TABLE III
COMPARISON WITH [3]

	Restoration ratio with random inputs			Restoration ratio with deterministic inputs		
Circuit	[3]	Our approach	Improvement	[3]	Our approach	Improvement
s38584	38	42	1.1	6	20	3.33
s38417	9	16	1.8	9	16	1.8
s35932	48	50	1.04	25	35	1.4

TABLE IV
COMPARISON WITH [4] WITH DETERMINISTIC INPUTS

	Restoration ratio		
Circuit	[4]	Our approach	Improvement
s38584	9	20	2.22
s38417	14	16	1.14
s35932	22	35	1.6

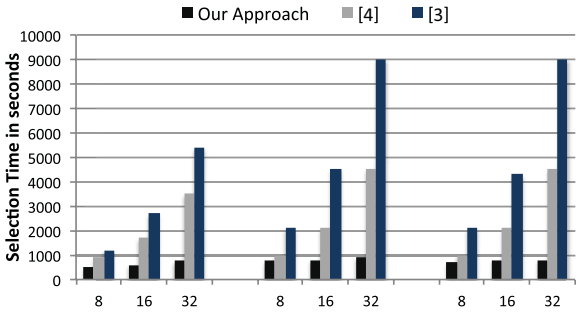


Fig. 9. Comparison of signal selection time.

The CDFG is then analyzed to provide the list of selected signals using Algorithm 2. As can be seen in Fig. 8, we have compared the results obtained using GSS and RSS on the same circuits to compare the restoration performance of each approach. The signals selected using RSS are mapped to gate-level and the restoration performance is noted. Simultaneously, the RTL design is synthesized to gate-level netlist, and GSS is applied on the netlist. The restoration performance of the two algorithms is then compared as discussed in Section VI-C. Our RSS algorithm is found not to incur any significant restoration penalty compared to the GSS algorithm.

B. Results on GSS

We would like to compare our signal selection approach with the other closely related methods. Table III compares the performance of our approach with the one proposed by Ko *et al.* [3] using the three largest ISCAS'89 benchmark circuits. All the experiments have been performed with a trace buffer of width 32. Table III is divided into three distinct parts. The first column indicates the circuit name. The next three columns compare the performance when random sets of inputs are used to drive the circuits. In this case, even the control signals are driven using random inputs. The improvement can be defined as the ratio between the restoration ratio using our approach and that of [3]. The third part of Table III compares our approach with [3] when the gates of the circuit are driven deterministically. This means that the control signals are driven using values that prevent it from going to a reset state, while the other signals are driven with random inputs. From Table III, it can be seen that the improvement obtained using random inputs is moderate (31% on average). On the other hand, considerable gain (117% on average) is obtained when we use our algorithm for deterministic inputs. As stated in [4], deterministic inputs are actually used in circuits during real-life applications. Hence, gain obtained with them is more significant.

Table IV compares the restoration ratio of our proposed approach with the one proposed by Liu *et al.* [4] for the three largest ISCAS'89 benchmarks. As before, a trace buffer width of 32 is chosen. In this case, the inputs are deterministic in nature. An average improvement of 113% is observed. It can be seen that the improvement here is less than the one obtained in Table III. This can be attributed to the fact that the algorithm proposed by [4] is more efficient than [3].

Fig. 9 compares our signal selection time against the time taken by [3] and [4] for the three largest ISCAS'89 benchmark circuits. It can be seen that our approach takes significantly less time (up to 90%) compared to them. This is primarily due to the fact that [3], [4] recompute edge values in every iteration whereas we compute them once. In summary, our GSS technique shows considerable improvement (up to 3 times) in signal restoration and significant reduction (up to 10 times) in signal selection time compared to the existing approaches.

C. Results on RSS

In this section, we discuss how our RTL level signal selection algorithm can further improve the signal selection time without compromising on restoration ratio. As discussed before, we have applied our approach on the designs obtained from the Opencores benchmarks. We have compared our RSS approach with the GSS procedure. The results are shown in Table V. Similar to the previous experiments, we have assumed a trace buffer of width 32.

The first column in Table V provides the circuit name. The second column shows the memory size reduction, which is the ratio of memory size in gate-level and RTL-level.

The last column gives the speedup obtained using RSS compared to GSS. Speedup can be defined as the ratio of GSS time to RSS time. As can be seen, RSS is up to 3600 times faster and requires up to 191 times less memory compared to GSS.

Fig. 10 compares the restoration performance of RSS and GSS for some of the benchmarks. As can be seen, the restoration performance is similar. The gate-level restoration

TABLE V
RTL-LEVEL VERSUS GSS

Circuit	Memory size reduction	Speedup
Total CPU	8.1	697
Wishbourne LCD controller	22.81	1923
dmx512 tranceiever	191.24	733
OPB onewire	3.22	3600
Simple RS232 Uart	3.8	500

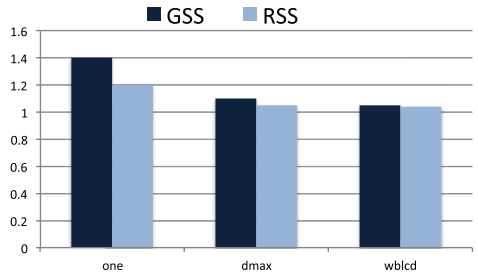


Fig. 10. Comparison of restoration performance.

performance is found to be slightly better than the RTL-level in some cases. The primary reason for this is the representation of sequential elements as arrays in RSS. Whenever we select a signal for tracing at RSS, we are actually tracing all the elements in the array. However, some of the signals in the array might not be as good. Some other signals could have been selected for better restoration performance.

VII. CONCLUSION

Post-silicon validation is extremely complex and time consuming in IC design methodology. Signal selection is an important aspect of post-silicon debug. We developed techniques employing total restorability for selecting the most profitable signals that were guaranteed to generate better restoration compared to existing approaches. Our experimental results demonstrated two major advantages—our approach can provide faster (up to 90%) signal selection as well as significantly better (up to 3 times) restoration compared to existing approaches. Our RSS approach can further improve the signal selection time by several orders-of-magnitude, and also requires less memory compared to the GSS techniques.

REFERENCES

[1] A. Nahir, A. Ziv, M. Abramovici, A. Camilleri, R. Galivanche, B. Bentley, H. Foster, A. Hu, V. Bertacco, and S. Kapoor, "Bridging pre-silicon verification and post-silicon validation," in *Proc. Design Autom. Conf.*, Jun. 2010, pp. 94–95.

[2] K. Basu and P. Mishra, "Efficient trace data compression using statically selected dictionary," in *Proc. IEEE 29th VLSI Test Symp.*, May 2011, pp. 14–19.

[3] 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*, vol. 28, no. 2, pp. 285–297, Feb. 2009.

[4] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation," in *Proc. Design, Autom. Test Eur. Conf. Exhibit.*, Apr. 2009, pp. 1338–1343.

[5] O. Caty, P. Dahlgren, and I. Bayraktaroglu, "Microprocessor silicon debug based on failure propagation tracing," in *Proc. Int. Test Conf.*, Nov. 2005, pp. 10–293.

[6] F. M. De Paula, M. Gort, A. J. Hu, S. Wilton, and J. Yang, "BackSpace: Formal analysis for post-silicon debug," in *Proc. Formal Methods Comput.-Aided Design*, Nov. 2008, pp. 1–10.

[7] G. J. Van Rootselaar and B. Vermeulen, "Silicon debug: Scan chains alone are not enough," in *Proc. Int. Test Conf.*, Sep. 1999, pp. 892–902.

[8] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for SoCs," in *Proc. Design Autom. Conf.*, 2006, pp. 7–12.

[9] W. Mao and R. K. Gulati, "Improving gate level fault coverage by RTL fault grading," in *Proc. Int. Test Conf.*, 1996, pp. 150–159.

[10] N. Yogi and V. Agrawal, "Spectral RTL test generation for gate-level stuck-at faults," in *Proc. 15th Asian Test Symp.*, Nov. 2006, pp. 83–88.

[11] H. F. Ko and N. Nicolici, "Automated trace signals selection using the RTL descriptions," in *Proc. Int. Test Conf.*, 2011, pp. 1–10.

[12] E. Taylor, J. Han, and J. Fortes, "Toward accurate and efficient reliability modeling of nanoelectronic circuits," in *Proc. IEEE-NANO*, Jun. 2006, pp. 395–398.

[13] *Icarus Verilog*. (2012, Apr.) [Online]. Available: http://www.icarus.com/eda/verilog/

[14] S. P. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. New York: Springer-Verlag, 2008.

[15] *OpenCores*. (2012, Apr.) [Online]. Available: http://opencores.org/

Kanad Basu (S'08) received the B.E. degree from the Department of Electronics and Telecommunication Engineering, Jadavpur University, West Bengal, India. He is currently pursuing the B.E. degree with the Department of Computer and Information Science and Engineering, Embedded Systems Laboratory, University of Florida, Gainesville.

His current research interests include functional and structural testing, design for test, and post-silicon validation.

Mr. Basu was a recipient of the Best Paper Award from the International Conference on VLSI Design in 2011.

Prabhat Mishra (S'00–M'04–SM'08) received the B.E. degree from Jadavpur University, West Bengal, India, the M.Tech. degree from the Indian Institute of Technology, Kharagpur, India, and the Ph.D. degree from the University of California, Irvine, all in computer science.

He is currently an Associate Professor with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville. He has published four books, nine book chapters, and more than 80 research papers in premier journals and conferences. His current research interests include design automation of embedded systems, energy-aware computing, and hardware verification.

Dr. Mishra serves as an Associate Editor of the *Association for Computing Machinery (ACM) Transactions on Design Automation of Electronic Systems,* the IEEE DESIGN AND TEST OF COMPUTERS, and the *Journal of Electronic Testing*. He is a Guest Editor of the IEEE TRANSACTIONS ON COMPUTERS, a Program/Organizing Committee Member of several ACM and IEEE conferences. His research has been recognized by several awards, including the National Science Foundation CAREER Award in 2008, two Best Paper Awards (VLSI Design in 2011 and CODES+ISSS in 2003), and the European Design and Automation Association Outstanding Dissertation Award from the European Design Automation Association in 2004.