Efficient Trace Signal Selection for Post Silicon Validation and Debug*

Kanad Basu and Prabhat Mishra
Computer and Information Science and Engineering
University of Florida, Gainesville FL 32611-6120, USA
email: {kbasu, prabhat}@cise.ufl.edu.

Abstract

Post-silicon validation is an essential part of modern integrated circuit design to capture bugs and design errors that escape pre-silicon validation phase. A major problem governing post-silicon debug is the observability of internal signals since the chip has already been manufactured. Storage requirements limit the number of signals that can be traced; therefore, a major challenge is how to reconstruct the majority of the remaining signals based on traced values. Existing approaches focus on selecting signals with an emphasis on partial restorability, which does not guarantee a good signal restoration. We propose an approach that efficiently selects a set of signals based on total restorability criteria. Our experimental results demonstrate that our signal selection algorithm is both computationally more efficient and can restore up to three times more signals compared to existing methods.

I. Introduction

Pre-silicon validation techniques are used by design engineers to get rid of functional errors. This is done before the actual manufacturing process and can be accomplished using simulation or formal verification techniques. However, with the increase in design complexity and decrease in time-to-market window, many errors escape the pre-silicon validation phase and manifest themselves during the actual operation. Manufacturing testing techniques are used to capture manufactural defects ([7], [8]), but they are not designed to detect any of the bugs escaping the pre-silicon phase. Post-silicon validation techniques are used to capture these escaped bugs.

Post-silicon debug comprises of signal observation and analysis. Since we have a completely fabricated silicon, it is not possible to observe each and every internal signal. Only a few selected signals can be observed. During post-silicon validation, a set of input tests are used, and the values of the selected signals are stored in a trace buffer. The data from the trace buffer is used to restore the unobserved signal states during debug. Different techniques of post-silicon validation have been proposed over the years keeping this problem in mind. Both Ko et al. [5] and Liu et al. [6] have proposed similar approaches of signal selection based on partial restoration[1]. For each signal, the sum of the partial restorabilities of all the signals in the circuit is computed. If the trace buffer width is n, the n signals providing highest sum of partial restorabilities are chosen for tracing.

Partial restoration probabilities (partial restorabilities, in short), used in the approaches proposed by [5] and [6] are not sufficient for signal reconstruction, as can be seen in Section III. Also, the approaches in [5] and [6] suffer from the fact that their restorability calculation is computationally intensive, which increases the complexity of their algorithm.

We have proposed a method that addresses these challenges. We have used total restorability[2] calculations, which can guarantee better restoration compared to partial restorability. Our method is found to provide both higher signal restoration ratio and significantly lower signal selection time than any of the existing approaches using the ISCAS '89 benchmarks as demonstrated in Section V.

The rest of the paper is organized as follows. Section II presents related works in signal selection. Section III describes the signal selection problem using illustrative examples. Section IV describes our signal selection technique. Section V presents the experimental results. Finally, Section VI concludes the paper.

II. Related Work

The primary problem concerning post-silicon debug is the limited observability of the internal signals. Once the values of signals are known, they can be analyzed using some algorithms like failure propagation tracing [9] to identify the errors in the circuit. Formal analysis for post silicon debug, proposed by De Paula [4], is of limited use as it is not applicable to circuits with a large number of gates. Physical probing techniques were proposed by Nataraj et al. [1]. Decrease in feature size and growing complexity of IC designs have rendered these techniques difficult in practice. A method for verification of memory subsystem in CMPs was proposed by DeOrio [10], which only emphasizes on the memory subsystem and not the entire circuitry of the chip. Scan based debugging techniques such as [2] are not appropriate since they require to stop the circuit functionality when the scan data is being written. This is particularly not beneficial in cases where the functional

*This work was partially supported by NSF CAREER award 0746261.

[1]**Partial Restorability** of a signal refers to the probability that the signal value can be reconstructed using known values of some other traced signals.

[2]**Total Restorability** of a group of signals refer to the fact that the signal states can be completely restored, that is, it is a special case of partial restorability with restorability value of 100%.

errors are drastically apart. Double buffering [12] of scan elements helps to mitigate this problem, but with a large area penalty. 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 embedded logic analyzer (ELA) [3], and shadow flip flops [12] have been proposed over the years; however, none of them are really effective.

A logic implication based trace signal selection method was proposed by Prabhakar et al. [11]. They used the primary inputs, in addition to the traced signals for restoration purposes. Recently, Ko et al. [5] and Liu et al. [6] have proposed a generic trace signal selection algorithm in which a few important signals can be traced and others can be reconstructed from them. Our technique is closest to their approach and hence, throughout this paper, and specially in Section V, we have compared our proposed technique with them. Our proposed technique is found to overcome the drawbacks of [5] and [6].

III. Background and Motivation

A. Signal Reconstruction

In Post-silicon debug, unknown signal states can be reconstructed from the traced states in 2 ways - forward and backward restoration. Forward restoration deals with the restoration of signals from input to output, that is, knowledge of input values can provide the output. Backward restoration, on the other hand, deals with reconstructing the input from the output. Details on forward and backward restoration have been explained in [5]. It is sometimes easier to restore signals using forward rather than backward restoration. If all but the unknown signal values are known, forward restoration can definitely determine the unknown, while backward might fail to do so. This information will be used later in Section IV-E.

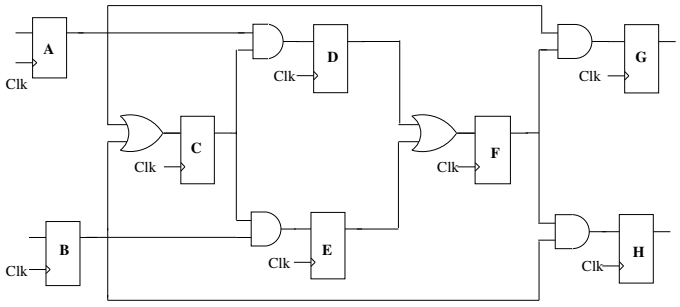


Fig. 1. Example Circuit

We now show by using a simple circuit how reconstruction is performed in [5] and [6]. An example circuit is shown in Figure 1 having 8 flip-flops. Let us assume that the trace buffer width is 2, that is, value of two signals can be recorded. We try to restore the other signal states by application of the methods presented in [5] and [6]. The results are shown in Table I. The 'X's represent those states which cannot be determined. The selected signals are shown in shades. Partial restorability calculations for both [5] and [6] are such that the signals selected are C and F, in that order. Restoration ratio, which is a popular metric for calculation of signal restorability is defined as: Restoration Ratio = (number of states restored + number of states traced)/(number of states traced). It can be seen that the restoration ratio using the methods of [5] and [6] is **2.6** for this example.

TABLE I. Restored signals using [5] and [6]

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

B. Motivational Example

We now employ our method for selecting signals in the circuit in Figure 1. The first signal that we trace is C. Note that this was the same signal that was chosen by [5] and [6]. The second signal that we choose is A[3], based on total restorability computations. The results are shown in Table II. It can be seen that our method provides a restorability ratio of **3.2**, which is better than the one provided by [5] and [6].

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

IV. Signal Selection for Post-Silicon Debug

Algorithm 1 shows our signal selection procedure that has five important steps. The remainder of this section describes each of the steps in detail[4].

Algorithm 1: Signal Selection Algorithm

Input: Circuit, Trace Buffer
Output: List of selected signals S (Initially Empty)
1: Compute the values of all edges and all flip-flops.
2: Find the flip-flop with the highest value and add to S.
3: Create Initial Region.
while *trace buffer is not full* **do**
  4: Recompute the values of flip-flops.
  5: Compute Region growth by finding the flip-flop in the region with highest value not in S and add to S.
end
return S

A. Computation of Edge Values

An edge between two flip-flops is the path taken to reach a flip-flop from another, while passing through a number of combinational gates between them, that is, there cannot be any

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

[4]Steps 4 and 5 correspond to computations for total restorability

flip-flops in between them. The edge may be in the forward or backward direction. In Figure 1, 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 the value at A (which is the value of the edge AC), there can be two cases (independent and dependent) as discussed below:[5]

1) **Independent Signals:** Consider two edges AC and BC in Figure 1. 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 Figure 2. Later, we will show how the calculation works for the specific case in Figure 1.

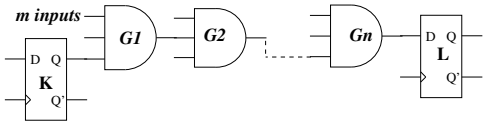


Fig. 2. Example circuit with n gates

Figure 2 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 path 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 value 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 value of '1', output value of '0' and '1', respectively. The output of flip-flop K can influence the output of G_1 in two cases: i) output of K is a controlling value, ii) all the inputs to G_1 are complement of the controlling value. Let us consider G_1 to be a 2-input AND gate[6]. We define P_{G_1} as the overall probability of K controlling G_1. According to [13],

$$P_{G_1} = P^O{}_{1,G_1} + P^O{}_{0,G_1} \qquad (1)$$

Now, let's define $P^O{}_{0,G_1}$ and $P^O{}_{1,G_1}$. Let P_{cond0,G_1} and P_{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 have assumed $P^I{}_{0,G_1} = P^I{}_{1,G_1} = 0.5$[7].

$$P^O{}_{0/1,G_1} = P_{cond0/1,G_1} \times P^I{}_{0/1,G_1} \qquad (2)$$

Now, for a 2-input AND gate, P_{cond0,G_1} is 1, since 0 is the controlling input. Therefore, we obtain $P^O{}_{0,G_1} = 0.5$. Similarly, since 1 is the non-controlling input, P_{cond1,G_1} is 0.5, which gives $P^O{}_{1,G_1} = 0.25$. From Equation (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 can extrapolate along the entire path to L. We use the same set of equations (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 equation (2). For example if G_2 is also a 2-input AND gate, applying equation (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 Equation (1).

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

$$P^O{}_{0/1,G_n} = \prod_{1 \le i \le n} (cond0/1, G_i) \times P^I{}_{0/1,G_1} \qquad (3)$$

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

We can use these computations to show how an edge value is computed in case of the circuit in Figure 1. 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, $P_{cond0,G} = 0.5$ and $P_{cond1,G} = 1$. Therefore, equation (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 know the probability of a flip-flop output influencing a m-input gate, when the output of the flip-flop affects l inputs ($l \ge 2$) of the gate.

We have used a generic example in Figure 3 to calculate the edge value in case of dependent signals. It should be noted that these calculations on dependent signals have not been done by [5] or [6].

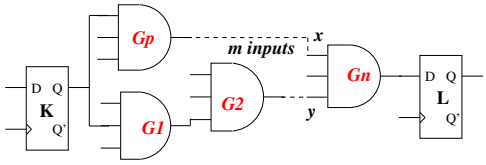


Fig. 3. Example circuit

Let's consider Figure 3. 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-A.1 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-A.1. 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

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

The last term subtracts the probability when both are 0, since it is being computed twice. Similarly, since 1 is the non-controlling input, we get

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

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

[6]The same method can be extrapolated for gates with higher number of fan-ins and different types like OR, NAND, etc.

[7]That is, occurance of 0 or 1 follows equal probability at the input.

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

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

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

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

Since the paths from K to x and from K to y are assumed to be an independent one, Equation (3) can be used to obtain the values $P^O{}_{0/1,x/y}$. Application of Equations (4) and (5) provide the values of $P^I{}_{0/1,G_n}$. The final P_{G_n} can be obtained using Equations (1) and(2), 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 Section IV-A.1 and Section IV-A.2 can be used to determine the edge values for the circuit in Figure 1. A graphical representation of the circuit is shown in Figure 4.

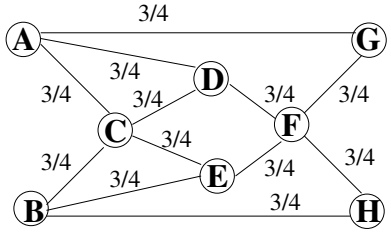


Fig. 4. Graphical representation of example circuit

The flip-flops are represented by nodes and an edge between two flip-flops by a straight line. In Figure 4, we have shown edge values on the top of each edge. 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 $\frac{3}{4}$.

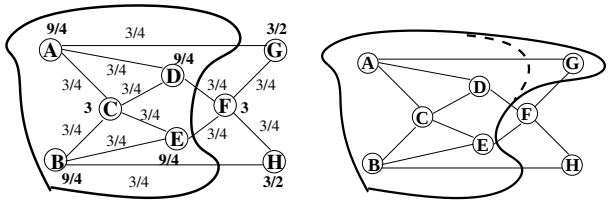
B. Initial Value Computation for Flip-Flops

We define the value of a flip-flop is the sum of all the edges attached with it, in both forward and backward direction. For example, in Figure 4, 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 [5] as well. The change in restoration ratio with a variation of this parameter is explained in Section V.

It is to be noted that our computation of the flip-flop values are independent of the sequential loops in the circuit. In a sequential loop, the output of a flip-flop depends on another in both forward and backward cycles. However, both cannot be true at the same clock cycle; that is, the same flip-flop can not determine the output of another in the same cycle by both forward and backward restoration. While forward restoration can determine the value in at least the next cycle, backward can determine it at most the previous cycle.

C. Initial Region Creation

A region is a collection of flip-flops attached together. It is not necessary that all the flip-flops have an edge with each other in the region. However, each flip-flop in the region must have at least one edge with another flip-flop in the region. In Figure 4, the flip-flops A, B, C, D and E form a region. The first flip-flop to be chosen is the flip-flop with the highest value, based on the calculations on Section IV-B. It is added to a list called "known". Now, all flip-flops which have an edge with the recently selected flip-flop are added to the region.



(a) Initial region creation (b) Region Growth

Fig. 5. Region creation and growth

We show by an example in Figure 5(a) how this portion of our algorithm is used to perform the selection of the best signals. The values of the flip-flops (addition of all it's edge values) are shown in bold alongside each flip-flop. For example, A has 3 edges AC, AD and AG, each having a value $\frac{3}{4}$. Therefore, the value for A is $\frac{3}{4}+\frac{3}{4}+\frac{3}{4}=\frac{9}{4}$. The flip flop with the highest value in Figure 5(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 Figure 5(a).

D. Recomputation of Flip-Flop Values

The first flip flop to be traced is already known (C in the previous example). However, there are other flip-flops that need to be traced as well. To select the subsequent flip-flops, the values of flip-flops inside the region are recomputed. The flip-flop whose value is being computed may have an edge to a flip-flop inside the region as well as one outside the region. Edges to flip-flops inside the region are given higher weight[8]. As seen in Section III, many restorability computations require knowledge of more than one signal of the input/output[9]. Therefore, it is better to gain more knowledge of the signals already in the region, thus increasing their restorability values and therefore, aiming for total restorability of those signals. Existing approaches [5] and [6] recompute the restorability values after each iteration, which when translated to the graph in Figure 4, would correspond to edge value recomputation, which is more computationally intensive.

E. Region Growth

The flip-flop in the region with the highest restorability and not in the list "known" is determined. If two flip-flops have the same value, the one with the higher forward restoration is traced. This is because, backward restoration fails in some

[8]Variation of restoration ratio with *weight* is discussed in Section V.

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

cases whereas forward does not when all the inputs are known. For the example in Figure 5(a), the next flip-flop 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 flip flops having an edge to the recently selected node are added in the region. As shown in Figure 5(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, Section IV-D is reconsidered and this process is iterated until the trace buffer is full.

With each call of the "Region Growth" procedure, the set of signals candidate increases, thus enlarging the area over which the potential signals can be selected. One may think that "Region Growth" clusters nodes in the circuit and only selects signals in a specific area. This is not true as the region growth is found to cover different parts, spanning across the entire circuit, with an increase in region size for each iteration.

V. Experimental Results

We applied our approach on the ISCAS'89 benchmarks used by [5] and [6] to compare with their methods and hence show the effectiveness of our algorithm. The trace buffers used are same as that of [5], that is, $8 \times 4k$, $16 \times 4k$ and $32 \times 4k$. We have designed an event driven simulator in the lines of the one described by [6] 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 10 sets of random values and noted the average restoration ratio.

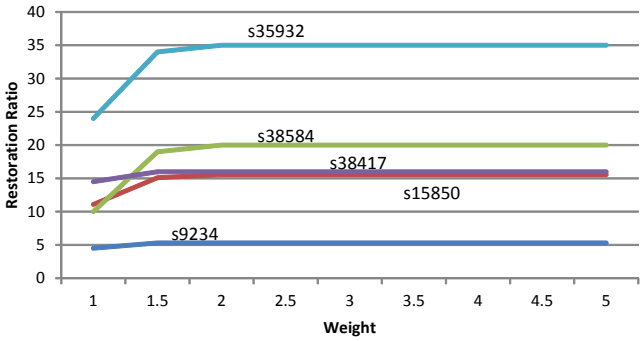


Fig. 6. Variation of Restoration Ratio with *weight*

Before we start comparing our approach with other relevant works, we intend to fix the two parameter values that we have used in our algorithm, namely *weight* and *threshold*. *weight* is used to ensure total restorability, while *threshold* keeps a check on gate count and prohibits loops in the circuit during computation. Figure 6 and Figure 7 show the variation of restoration ratio with *weight* and *threshold* respectively for the five largest ISCAS '89 benchmark circuits when the trace buffer width is 32 and the circuit is driven using deterministic inputs. It can be seen from Figure 6 that although the restoration ratio increases in the early stages, after reaching a *weight* value of 2, it remains constant with further increase in *weight*. This is because with a *weight* value of 2, enough weight is given to the flip-flops inside the region and hence, any increase in *weight* will only lead to the selection of the same flip-flops, resulting in the same restoration ratio. Thus, a *weight* value of 2 is used for the experiments in the remainder of this section.

In Figure 7, the restoration ratio is seen to remain constant after attaining a *threshold* value of 20. We choose a *threshold* value of 20 as a safe measure. It can be noted that an increase in *threshold* value leads to higher signal selection time. This is quite obvious since more time will be spent on the loops inside the circuit. Once the two parameters are set, we are

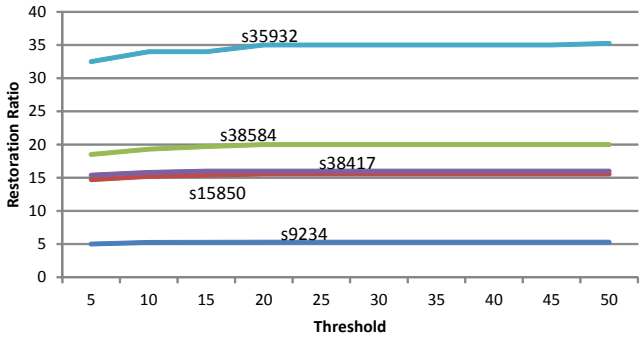


Fig. 7. Variation of Restoration Ratio with *threshold*

now ready to compare our signal selection approach with the other relevant methods. Table III compares the performance

TABLE III. Comparison with [5]

		Restoration Ratio with random inputs			Restoration Ratio with deterministic inputs		
Circuit	Width	[5]	Our approach	Impro-vement	[5]	Our approach	Impro-vement
s38584	8	135	155	1.1	19	78	4.1
	16	70	82	1.2	11	40	3.64
	32	38	42	1.1	6	20	3.33
s38417	8	19	55	2.9	19	55	2.9
	16	18	29	1.6	18	29	1.6
	32	9	16	1.8	9	16	1.8
s35932	8	180	188	1.05	42	95	2.3
	16	93	96	1.03	40	60	1.5
	32	48	50	1.04	25	35	1.4

of our approach with the one proposed by Ko et al. [5] using the three largest ISCAS '89 benchmark circuits. Table III is divided into three distinct parts. The first two columns indicate the experimental setup, that is, circuit name and width of the trace buffer. The next three columns compare the performance when random sets of inputs are used to drive the circuits. It is to be noted that in this case, even the control signals are driven using random inputs. As a result, the circuit might fall into one of the reset states. The improvement can be defined as the ratio between the restoration ratio using our approach and that of [5]. The third part of Table III compares our approach with [5] 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 (41% on average). On the other hand, considerable gain (151% on average) is obtained when we use our algorithm for deterministic inputs. As discussed earlier, random inputs might lead to reset states, which are responsible for high

restoration. Therefore, improvement obtained is less in this case. It should be noted, as stated in [6], that deterministic inputs are actually used in circuits during real-life applications. Hence, gain obtained with them are more significant.

Table IV compares the restoration ratio of our proposed approach with the one proposed by Liu et al. [6] for the three largest ISCAS'89 benchmarks. In this case, the inputs are deterministic in nature. It is to be noted that the values mentioned in [6] are slightly different from the values mentioned in Table IV. The reason is that the data inputs have been fed with random values, which differ from system to system. 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 [6] is a betterment over that proposed by [5].

TABLE IV. Comparison with [6] with deterministic inputs

			Restoration Ratio		
Circuit	FFs	Width	[6]	Our approach	Improvement
s38584	1426	8	20	78	3.9
		16	14	40	2.86
		32	9	20	2.22
s38417	1636	8	19	55	2.9
		16	18	29	1.61
		32	14	16	1.14
s35932	1728	8	64	95	1.48
		16	40	60	1.5
		32	22	35	1.6

We now compare our approach with the one proposed by [11]. It is to be noted that [11] have used the primary inputs along with the traced signals for signal restoration. Till now, we only used the trace signals to restore the rest of the signals on the chip. However, to enable fair comparison, we have included the primary inputs for signal restoration. The results are shown in Table V. It should be noted that the improvements are moderate (on an average 11%) in this case. When we use the primary inputs for restoration, most of the states at later clock cycles can be recovered. On the other hand, the states where the input test vectors can not reach due to sequential depth in early cycles can be restored using the traced data. As reported in [11], about 90-95% of the states were restored using their method. Hence, the scope for improvement is limited.

TABLE V. Comparison with [11]

			Restoration Ratio		
Circuit	FFs	Width	[11]	Our approach	Improvement
s5378	179	8	19.3	19	0.99
		16	9.7	9.9	1.02
		32	4.84	5.0	1.03
s9234	211	8	20.3	23.3	1.14
		16	10.3	11.8	1.14
		32	5.2	6.0	1.15
s15850	534	8	55.6	55.1	0.99
		16	27.8	29.8	1.07
		32	13.9	15.8	1.14
s38584	1426	8	130.1	151.2	1.61
		16	66.02	78.4	1.19
		32	34.8	40.5	1.16
s35932	1728	8	209.6	209.4	0.99
		16	104.8	105.8	1.01
		32	52.4	53.3	1.02

Figure 8 compares our signal selection time against the times taken by [5] and [6] 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 [5] and [6] recomputes edge values in every iteration whereas we only compute them once. In summary, our technique shows considerable improvement in signal restoration as well as significant reduction in signal selection time compared to the existing approaches.

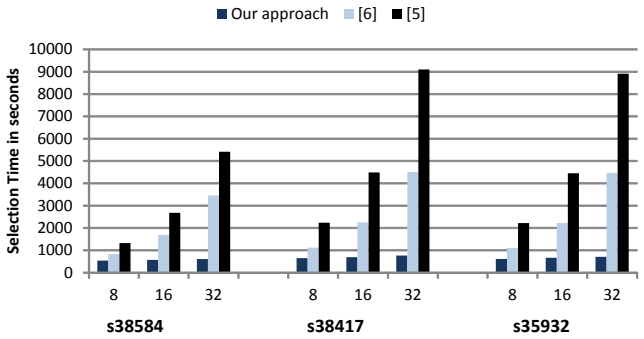


Fig. 8. Comparison of Signal Selection Time

VI. Conclusions

Post-silicon validation is extremely complex and time consuming in overall design methodology. Signal selection is an important aspect of post-silicon debug. We developed techniques to employ total restorability for selecting the most profitable signals that are guaranteed to generate better restoration compared to when signals are selected using partial restorability equations. We applied our algorithm on the largest ISCAS'89 benchmarks. 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 ratio compared to existing approaches.

References

[1] N. Nataraj, T. Lundquist, K. Shah, "Fault localization using time resolved photon emission and stil waveforms," in *ITC 2003*, pp. 254–263.

[2] G. J. Van Rootselaar and B. Vermeulen, "Silicon debug: scan chains alone are not enough," in *ITC 1999*, pp. 892–902.

[3] M. Abramovici et al., "A reconfigurable design-for-debug infrastructure for socs," in *DAC*, 2006, pp. 7–12.

[4] F. M. De Paula, M. Gort, A. J. Hu, S. Wilton, and J. Yang, "Backspace: Formal analysis for post-silicon debug," in *FMCAD '08.*, pp. 1–10.

[5] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE TCAD*, vol. 28, no. 2, pp. 285–297, Feb. 2009.

[6] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation," in *DATE*, 2009, pp. 1338–1343.

[7] K. Basu and P. Mishra, "Test Data Compression Using Efficient Bitmask and Dictionary Selection Methods," in *IEEE Transactions on VLSI*, vol. 18, no. 9, pp. 1277–1286, 2010.

[8] K. Basu and P. Mishra, "A novel test-data compression technique using application-aware bitmask and dictionary selection methods," in *ACM Great Lakes Symposium on VLSI 2008*, pp.83–88.

[9] O. Caty, P. Dahlgren, and I. Bayraktaroglu, "Microprocessor silicon debug based on failure propagation tracing," in *ITC 2005*, pp.10pp–293.

[10] A. DeOrio et al., "Dacota: Post-silicon validation of the memory subsystem in multi-core designs," in *HPCA 2009*, pp. 405–416.

[11] S. Prabhakar and M. Hsiao, "Using Non-Trivial Logic Implications for Trace Buffer-based Silicon Debug," in *ATS 2009*, pp. 131–136.

[12] D. Josephson and B. Gottlieb, "The crazy mixed up world of silicon debug [ic validation]," in *CICC 2004*, pp. 665–670.

[13] E. Taylor et al., "Towards Accurate and Efficient Reliability Modeling of Nanoelectronic Circuits ," *Proc. IEEE-NANO*,2006, pp. 395–398.