Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation

Xiao Liu[†] and Qiang Xu[†‡]
[†]CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[‡]CAS-CUHK Shenzhen Institute of Advanced Integration Technology
Email: {xliu,qxu}@cse.cuhk.edu.hk

Abstract

Today's complex integrated circuit designs increasingly rely on post-silicon validation to eliminate bugs that escape from pre-silicon verification. One effective silicon debug technique is to monitor and trace the behaviors of the circuit during its normal operation. However, designers can only afford to trace a small number of signals in the design due to the associated overhead. Selecting which signals to trace is therefore a crucial issue for the effectiveness of this technique. This paper proposes an automated trace signal selection strategy that is able to dramatically enhance the visibility in post-silicon validation. Experimental results on benchmark circuits show that the proposed technique is more effective than existing solutions.

1 Introduction

With the ever-increasing design complexity and the ever-shrinking time-to-market (TTM) window for today's integrated circuit (IC) products, errors are more likely to escape the pre-silicon verification process and manifest themselves after design tape-out. Needless to say, it is essential to identify these bugs before the ICs are shipped to customers. This step in the design flow, known as the post-silicon validation process (or silicon debug process), has become increasingly important due to its associated TTM delay and the high re-spin cost [1, 4].

The main difficulty in post-silicon validation lies in the limited visibility for circuit internal nodes, because the circuit under debug (CUD) is a piece of silicon that has already been fabricated. Physical probing tools (e.g., [6]) are of help for diagnosis, but their capabilities are diminishing with the high number of metal layers and the nano-scale feature size in today's advanced technology. Reusing internal scan chains for silicon debug is a widely-utilized technique in the industry nowadays [11, 13], but it needs to stop the operation of the CUD and hence only provides postmortem debuggability. Adding shadow flip-flops (FFs) to the circuit can mitigate this problem and enhance visibility during normal operation of the CUD [7], however, this method can only sample a few snapshots of the circuit's operational states and it also involves nontrivial design for debug (DfD) overhead.

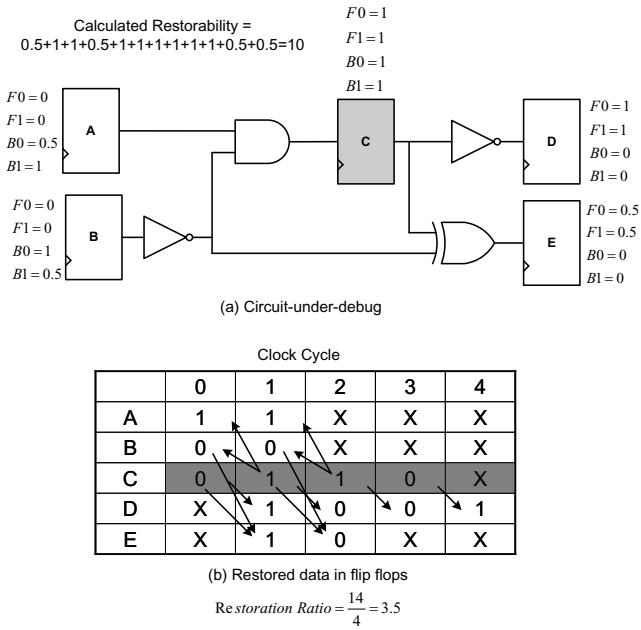
Many tricky bugs only manifest themselves after a long period of operations and hence are difficult to identify with the above techniques that reuse test structures for debug. In fact, it is essential to increase the observability of the design's internal nodes to a level that is much higher than what manufacturing test generally needs [4, 9]. One effective technique that provides real-time visibility to the CUD is to monitor and trace internal signals during its normal operation, which has been widely accepted in the industry recently (e.g., [2, 3, 10, 12, 14]). The trace data can then be stored in on-chip buffers and/or transferred off-chip via a trace port for later analysis. It is important to note that, as designers cannot afford very large DfD overhead associated with the trace buffers and/or trace ports, we can only trace a small number of signals in the design. Consequently, the effectiveness of this debug strategy highly relies on which signals are selected to trace in the system. That is, if we select the right signals to trace so that a bug leaves "evidences" on them, finding the root cause for the bug would be quite easy. Otherwise, the debug process can be quite challenging.

In current practice, designers manually select those signals that are considered to be vulnerable to bugs or important for analysis to trace, based on their design experience. While their knowledge about the design is of great help in trace signal selection, this ad-hoc process cannot guarantee the quality of the selected trace signals. More importantly, bugs often occur in unexpected scenarios and it is impossible to predict which signals will be related to them during the design phase. From this aspect, to achieve effective bug identification, we should at least add some trace signals that are selected in an automated manner without designers' intervention.

While we cannot afford to trace a large number of signals at the same time, it is possible to expand the logic states on the few trace signals to restore many missing states on those untraced signals, according to the circuit's logic structures. Based on the above *state restoration* concept, Ko and Nicolici [8] proposed the first automated trace signal selection method that tries to maximize the number of restored missing states. In this work, the authors defined the *forward restorability* and *backward restorability* for logic gates and applied circuit analysis to obtain the missing states. Their definitions for the gate-level restorabilities, however, lack theoretical basis and hence inaccurate in many cases. Also, their heuristic for state restoration at the circuit-level is not satisfactory.

In this paper, we present novel solutions to address the above problems. The main contributions include

- we define the gate-level restorabilities for visibility enhancement in a theoretically-precise manner;



	0	1	2	3	4
A	1	1	X	X	X
B	0	0	X	X	X
C	0	1	1	0	X
D	X	1	0	0	1
E	X	1	0	X	X

Figure 1. State Restoration Example in [8].

- we propose novel automated trace signal selection algorithms that are able to restore much more missing states when compared to [8];

The remainder of this paper is organized as follows. Section 2 reviews related prior work and motivates this paper. The proposed gate-level restorability definitions and our automated trace signal selection methodologies are shown in Section 3 and Section 4, respectively. In Section 5, we present our experimental results on benchmark circuits. Finally, Section 6 concludes this work.

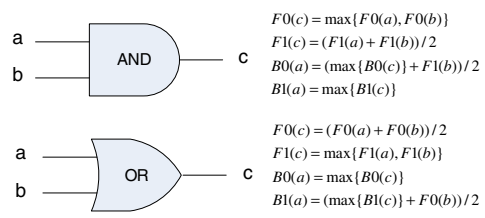
2 Preliminaries and Motivation

Ideally, we would like to be able to "see any signal at any time" in post silicon validation. Obviously, this is not achievable. With the help of limited DfD resources, however, it is possible to trace a few signals in the system to view parts of the system state and use them to reason the root cause of the design's erroneous behavior. Since the number of signals that can be concurrently traced is constrained, the key question is then how to select them intelligently to identify challenging bugs effectively. For processors in a system, we can simply select those signals that are enough for us to reconstruct the program flow. For random logic, however, this is a rather difficult problem and designers typically select trace signals manually according to their own experience and their knowledge about the design.

As bugs often occur in unexpected scenarios, it is important to have some trace signals that are selected in an automated manner without designers' intervention. To the best of our knowledge, [8] is the only attempt to address the above problem in the literature. In this work, the authors observed that it is possible to expand the logic states on the few trace signals to restore many missing states on those untraced signals by conducting structural analysis on the circuit[1]. The *restoration ratio*, calculated as $R = \frac{N_{traced}+N_{restored}}{N_{traced}}$, is then used as the evaluation metric to measure the quality of the selected trace signals. N_{traced} and $N_{restored}$ represent the number of traced states and that of restored states, respectively.

The state restoration process is conducted as follows. Consider a 2-input AND gate. If one of the inputs is '0', the output c can be inferred as '0' through forward propagation. Meanwhile, when the output c is '1', the two inputs a and b can be derived as '1' by backward justification. Based on the above simple operations, for an example sequential circuit in Fig. 1(a), when flip-flop (FF) C is traced for four clock cycles, the missing states can be inferred as shown in Fig. 1(b). The restoration ratio is hence $R = 14/4 = 3.5$ ($N_{traced} = 4, N_{restored} = 10$). It is important to note that the above state restoration is correct under the assumption that there are no timing errors between the traced signals and the restored signals (designers can vary the circuit operational speed during the debug process to make it true, if necessary), and hence is only effective for debugging functional errors.

Before observing the actual values on the traced signals, we can only estimate the restorability of each trace signal. [8] defined the *forward restorability* and *backward restorability* for logic gates as shown in Fig. 2. Based on their definition, the calculated restoration ratio equals $10/4 = 2.5$.



F1/F0 --- the probability of restoring data 1/0 of a node through forward propagation
B1/B0 --- the probability of restoring data 1/0 of a node through backward justification

Figure 2. Forward/Backward Restorability Definitions in [8].

The above definitions for the gate-level restorabilities, however, are lack of theoretical basis and hence are not accurate in many cases. Still consider a 2-input AND gate with two inputs a and b. Suppose we cannot restore the output $c = 0$ (i.e., $B0(c) = 0$), it is obvious that $a = 0$ cannot be restored through backward justification and $B0(a)$ should equal to 0. However, with the formula shown in Fig. 2, we have $B0(a) = (max\{B0(c)\} + F1(b))/2 = F1(b)/2$. Since the value of $F1(b)$ is dependent on the restorabilities of its driving gates, it can be any value between 0 and 1. The restorability calculated in [8] is hence inaccurate and it can be easily propagated to a large number of circuit nodes, which makes the circuit-level visibility calculation quite time-consuming. In addition, during the restorability calculation in [8], forward propagation and backward justification are applied separately to find missing states, which may under-estimate the possible restored states.

The limitations in [8] motivate us to propose new automated trace signal selection methodologies, as shown in the following sections.

[1] Similar concept has been mentioned earlier in [5] as "data expansion" technique, but for combinational logic only.

Selected Visibility ($SV1/SV0$)	For selected trace nodes, $SV1 = SV0 = 1$; otherwise $SV1 = SV0 = 0$
Forward Restorability ($FR1/FR0$)	The **conditional** probability to be restored as '1'/'0' by forward propagation **when** it is 1'/'0'
Backward Restorability ($BR1/BR0$)	The **conditional** probability to be restored as '1'/'0' by backward justification **when** it is 1'/'0'
Restorability ($R1/R0$)	The **conditional** probability that the node is restored as '1'/'0' from other nodes **when** it is 1'/'0'
Functional Probability ($P1/P0$)	The probability that the node is '1'/'0' in functional mode
Restored Visibility ($RV1/RV0$)	The probability that the restored value '1'/'0' is **actually** observed on the node
Visibility ($V1/V0$)	The probability that the value '1'/'0' is **actually** observed on the node

Table 1. Terminologies for Restorability Calculation.

3 Restorability Formulation

To calculate the restorability precisely, we first define the terminologies as shown in Table 1 and then we illustrate our calculations for gate-level restorabilities.

3.1 Terminologies

It is obvious the visibility for those signals that are selected to trace is 1 otherwise it is 0, and we use $SV1/SV0$ to denote it. For the untraced signals, they can be restored by two *independent* methods: forward propagation and backward justification, and we use $FR1/FR0$ and $BR1/BR0$ to represent the probabilities to restore them by the corresponding method. Consequently, the total possibility that the value '1'/'0' can be restored on the untraced signals, denoted as $R1/R0$, can be calculated as:

$$R1 = FR1 + BR1 - FR1 \times BR1 \tag{1}$$

$$R0 = FR0 + BR0 - FR0 \times BR0 \tag{2}$$

It is important to note that the above restorabilities are conditional probabilities and they do not represent the probability that the value '1'/'0' is *actually* observed on the node. For example, if we are able to fully restore value '1' on a particular node but cannot restore '0' on it (i.e., $R1 = 1$ and $R0 = 0$), and suppose this node stays at logic '0' most of the time, we can barely observe anything for this node. To address this problem, we need to consider the probability for a node to be '1'/'0' in functional mode (represented by $P1/P0$) and we calculate the restored visibility for this node as:

$$RV1 = R1 \times P1 \tag{3}$$

$$RV0 = R0 \times P0 \tag{4}$$

Finally, we unify the two kinds of visibilities obtained through direct trace or indirect restoration as follows:

$$V1 = \max\{SV1 \times P1, RV1\} \tag{5}$$

$$V0 = \max\{SV0 \times P0, RV0\} \tag{6}$$

3.2 Gate-Level Restorabilities

Let us first take the two-input AND gate as an example to explain our definition for gate-level restorability first. For forward propagation, the output (c) is '1' only when both inputs (a and b) are '1'; while it is '0' when no less than one input is '0'. For backward justification, both inputs (a and b) are '1' when output (c) is '1', and one input (e.g., a) is '0' only when the output c is '0' and at the same time the other input (b) is '1'. Based on the above, we define the restorabilities as follows.

$$FR1(c) = \frac{V1(a) \bigcap V1(b)}{P1(c)} \tag{7}$$

$$FR0(c) = \frac{V0(a) \bigcup V0(b)}{P0(c)} \tag{8}$$

$$BR1(a) = \frac{V1(c)}{P1(a)} \tag{9}$$

$$BR0(a) = \frac{V0(c) \bigcap V1(b)}{P0(a)} \tag{10}$$

Due to the computational complexity, however, it is not possible to record the precise relationship between every signal. Therefore, we assume the inputs to every gate are *independent*. Eq. (7) and Eq. (8) can then be simplified as follows.

$$FR1(c) = \frac{V1(a) \times V1(b)}{P1(a) \times P1(b)} \tag{11}$$

$$FR0(c) = \frac{V0(a) + V0(b) - V0(a) \times V0(b)}{1 - P1(a) \times P1(b)} \tag{12}$$

To calculate BR0(a) within the corresponding event $V0(c) \bigcap V1(b)$, let us first consider the case that b and c are not traced. We cannot obtain $V0(c) \bigcap V1(b)$ as $V0(c) \times V1(b)$ because the probability value on signal c strongly depends on the value on signal b. Fortunately, we notice that due to the nature of two-input AND gate, state $a = 0$ can be backward restored only in the event of *a=0, b=1, c=0*. The accurate probability for this event is $P0(a) \times P1(b)$ (see Table 2). Therefore, $V0(c) \bigcap V1(b)$ should be $\big(P0(a) \times P1(b)\big) \times \big(R0(c) \times R1(b)\big) = P0(a) \times R0(c) \times V1(b)$. Substituting it into Eq. (10) yields

$$BR0(a)|_{c \text{ is not traced}} = \frac{P0(a) \times R0(c) \times V1(b)}{P0(a)} = R0(c) \times V1(b) \tag{13}$$

Event	Probability	c	b	a(truly)	a(restored)
$a=1, b=0, c=0$	$P1(a) \times P0(b)$	0	0	1	X
$a=0, b=1, c=0$	$P0(a) \times P1(b)$	0	1	0	0
$a=0, b=0, c=0$	$P0(a) \times P0(b)$	0	0	0	X
$a=1, b=1, c=1$	$P1(a) \times P1(b)$	1	1	1	1

Table 2. AND Gate Backward Justification.

For the case when c is a traced signal, it is observable all the time and we can restore $R0(a)$ as long as $b = 1$ is visible. Therefore,

$$BR0(a)|_{c \text{ is traced}} = V1(b) \tag{14}$$

It can be verified that Eq. (13) and Eq. (14) also hold when b is traced. Based on the above principles, the restorability calculations for OR gate and XOR gate are presented as follows.

For OR gate (a,b–input, c–output)

$$FR0(c) = (V0(a) \times V0(b))/P0(c) \tag{15}$$

$$FR1(c) = (V1(a) + V1(b) - V1(a) \times V1(b))/P1(c) \tag{16}$$

$$BR0(a) = V0(c)/P0(a) \tag{17}$$

$$BR1(a)|_{c \text{ is not traced}} = (P1(a) \times R1(c) \times V0(b))/P1(a) = R1(c) \times V0(b) \tag{18}$$

$$BR1(a)|_{c \text{ is traced}} = V0(b) \tag{19}$$

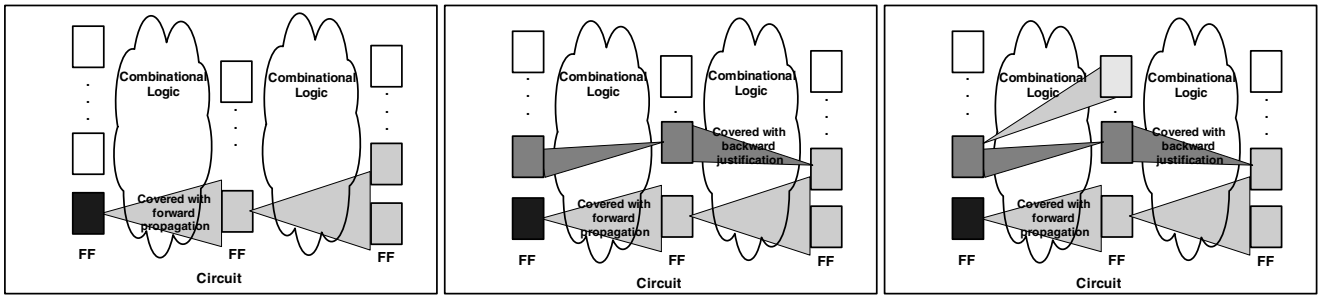
For XOR gate (a,b–input, c–output)

$$FR1(c) = (V1(a) \times V0(b) + V0(a) \times V1(b))/P1(c) \tag{20}$$

$$FR0(c) = (V0(a) \times V0(b) + V1(a) \times V1(b))/P0(c) \tag{21}$$

$$BR0(a)|_{c \text{ is not traced}} = R0(c) \times V0(b) + R1(c) \times V1(b) \tag{22}$$

$$BR0(a)|_{c \text{ is traced}} = V0(b) + V1(b) \tag{23}$$



(a) 1st forward propagation (b) 1st backward justification (c) 2nd forward propagation

Figure 3. Example for Circuit-Level Restoration Calculation.

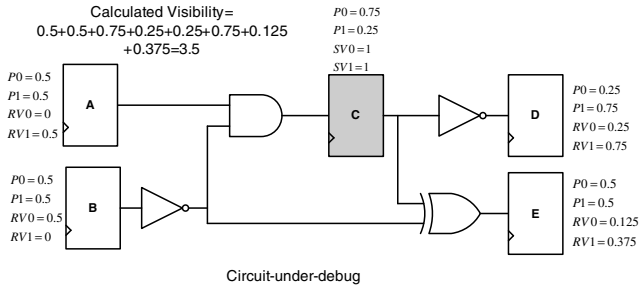


Figure 4. Example for Proposed Method.

$$BR1(a)|_{c\ is\ not\ traced} = R0(c) \times V1(b) + R1(c) \times V0(b) \quad (24)$$

$$BR1(a)|_{c\ is\ traced} = V1(b) + V0(b) \quad (25)$$

Finally, for gates with multiple fanouts, backward restoration from individual fanout is also treated as *independent* for the sake of simplicity. That is, the backward restorability for the corresponding output node is obtained by combining the restorabilities of its fanouts.

With our newly-introduced restorability definitions and calculation methods, for the same example shown in [8], the calculated total visibility when tracing FF *C* is 3.5 (see Fig. 4), which is the same as the actual restoration ratio obtained from simulation for this particular case. Such accuracy is of great help for selecting those "essential" signals that enhance the CUD's visibility, as shown in our experimental results in Section 5.

4 Trace Signal Selection

Based on the definitions in Section 3, the automated trace signal selection problem studied in this paper becomes: *how to set SV to be '1' for a constrained number of signals (FFs and/or input signals), so that the circuit's total visibilities ($TV = \sum V0 + \sum V1$) for all state elements is maximized.* To address this problem, this section presents how to calculate the circuit-level visibilities for trace signals and how to use them as guidance to select trace signals.

Our circuit-level visibility calculation procedure is shown in Fig. 5. Starting from the selected trace nodes, we iteratively use forward propagation and backward justification on combinational logic gates (depicted as $search-list$ in Fig. 5) to obtain the visibilities on other state elements, until TV converges (i.e., no more visibilities can be obtained). For the forward propagation process, we continuously apply the breath-first gate-level

INPUT: circuit, selected nodes
OUTPUT: TV: total visibility

```
1.  Set SV0, SV1 for selected nodes to be "1";
2.  while (TV is not converged with two directions' visibility calculation) {
3.    do {
4.      Put child nodes of FF_i with V_i! = 0 into search-list;
5.      while (Next FF level is not reached) {
6.        for each node in search-list {
7.          if (cur_node is passed by backward justification)
8.            Continue;
9.          Calculate visibility forwardly for cur_node;
10.         Mark cur_node to be covered by forward propagation;
11.         if (V on the child node of cur_node != 0)
12.           Put the child node of cur_node in search-list;
13.       }
14.     }
15.     Update TV with forward propagation;
16.   }while (TV is not converged with forward propagation);
17.   do {
18.     Put parent nodes of FF_i with V_i! = 0 into search-list;
19.     while (Previous FF level is not reached) {
20.       for each node in search-list {
21.         if (cur_node is passed by forward propagation)
22.           Continue;
23.         Calculate visibility backwardly for cur_node;
24.         Mark cur_node to be covered by backward justification;
25.         if (V on the parent node of cur_node != 0)
26.           Put the parent node of cur_node in search-list;
27.       }
28.     }
29.     Update TV with backward justification;
30.   }while (TV is not converged with backward justification);
31. }
32. return TV;
```

Figure 5. Procedure for Circuit-Level Visibility Calculation.

visibility calculation (see Section 3) for the driven gates connected to the traced signals (denoted as *child node* in Fig. 5) to obtain the visibilities for the FFs on the next logic level. The backward justification process is similar. The only difference is that it is implemented in the opposite direction, i.e., update

visibilities for the driving gates to the trace signals (denoted as *parent node* in Fig. 5) to calculate the visibilities for the FFs on the previous logic level.

For those untraced FFs with non-zero $V1$/ $V0$ after restoration, they can be utilized to further improve the visibilities of other state elements, as can be seen in Fig. 3. However, we have to be careful when applying the forward propagation and backward justification interleavedly for visibility calculation. This is because, if the visibility of a gate's output is obtained through forward propagation, this visibility should be not used to further improve the visibility of its inputs through backward justification. Otherwise, we have erroneously over-estimated visibilities with redundant information. To avoid this problem, in our procedure, whenever a gate is utilized for visibility calculation in the forward propagation process, it is labeled so that visibility calculation through it with backward justification is forbidden.

Similar to [8], the visibility estimation for sequential loops is inherently solved with the above iterative process. The difference from the procedure in [8] is that, they treat forward propagation and backward justification as independent processes, evaluate their restorabilities separately, and then sum them up together. Therefore, for the example in Fig. 3, [8] can only calculate the forward restorability in Fig. 3(a). While for our procedure, we evaluate the impact of the two directions interleavedly, and hence we can further calculate the extended restorabilities as shown in Fig. 3(b) and Fig. 3(c).

After obtaining the circuit-level visibility for given selected trace nodes using the above procedure, we use them to guide our trace signal selection, which is essentially a greedy heuristic, as shown in Fig. 6. In our algorithm, the trace nodes are selected one by one. For each selection, we try every un-selected node and always choose the one that results in the maximum TV after circuit-level visibility calculation (together with the already-selected nodes). One thing should be noted is that before the selection process, the nodes that prevent the CUD into functional mode (e.g., the reset signal) should be identified, and their effects should be blocked accordingly. Otherwise, the restoration effect may not be correctly estimated.

INPUT: : circuit, the number of selected nodes
OUTPUT: TV: total visibility, SNL: selected node list

```
1.  Identify nodes that prevent the CUD into functional mode (e.g., RESET);
2.  Block the un-functional effect for those nodes (e.g., De-assert RESET);
3.  while (more nodes can be selected) {
4.    for each un-selected node {
5.      Set SV0, SV1 for cur_node and selected nodes to be "1";
6.      Calculate circuit-level visibility;
7.      if (the returned cur_TV for cur_node is maximum) {
8.        Record cur_node & cur_TV;
9.      }
10.     Update the recorded node as selected one;
11.   }
12. }
13. return TV, SNL;
```

Figure 6. Procedure for Trace Signal Selection.

5 Experimental Results

5.1 Experiment Setup

We conduct experiments on ISCAS'89 benchmark circuits and compare against [8] to evaluate the effectiveness of the proposed solution (the authors in [8] provided their updated results for s38584, s38417 and s35932). As in [8], the trace buffer is defined as $8 \times 4k$, $16 \times 4k$ and $32 \times 4k$ in our experiments.

An event-driven simulator that is able to restore missing states is developed for experiments. Compared to traditional simulator, it conducts simulation in both forward and backward directions. Let us use AND gate again to describe how it works in backward simulation. If the output is logic '1', all the inputs of the gate are set as logic '1'. If the output is '0', only when all the other inputs are known to be logic '1', an input can be determined as logic '0'; otherwise, it is set as a 'X' bit. Known states of traced nodes, dumped from a commercial functional simulator with random input patterns (non-functional mode selection signals, e.g., RESET are de-asserted), are used as inputs to our simulator (missing states on the other nodes are initialized as 'X'). The whole simulation flow is also implemented as an iterative process. For each iteration, forward restoration is conducted from the first cycle to the last one. Next, backward restoration is applied in the reverse order. It should be emphasized that the number of cycles is more than $4k$ in our simulation, because extra states beyond $4k$ cycles are likely to be restored with the sampled signals (e.g., as Cycle 4 in Fig. 1(b)). The simulator ends when no more missing states can be restored.

The simulator is supplied with ten sets of random input patterns and we record their average visibility value.

5.2 Experimental Results

Table 3, Table 4, and Table 5 present our experimental results when 8 signals, 16 signals, and 32 signals are selected to trace, respectively. Column 2 presents the number of FFs in each circuit. The restoration results are described in Column 3 to Column 5 (for [8]) and Column 6 to Column 8 (for the proposed method). "# of VN" means the average number of node that is fully/patially restored, while "Amount of VS" illustrates the average amount of visible states. "Ratio" is the restoration ratio as defined in [8]. The "time" on the last column is the computational time spent for proposed signal selection.

In most cases, our automated trace signal selection method achieves a much higher restoration ratio when compared to [8], due to the more accurate visibility calculation. It should be noted that the presented results for [8] is for the case when higher restoration ratio is obtained with their design parameters $Threshold$ to be either 0.1 or 0.5. For s38584 and s38417, after selecting 8 nodes, our method is still able to restore large amount of missing states, while [8] cannot select the remaining "essential" nodes effectively (i.e., the restoration ration drops significantly). Similarly, for s35932, our method can still identify nodes that can restore many missing states after selecting 16 nodes, while the signals selected using [8] cannot restore any missing states (see Table 5).

The observed selection trend using our proposed technique is that the algorithm firstly selects input control node with many fan-outs, then it chooses those "essential" internal FF nodes. This is reasonable because these control signals can access a

		[8]			Proposed method				
Name	# of FF	# of VN	Amount of VS	Restoration ratio	# of VN	Amount of VS	Restoration ratio	Δ	Time(s)
s5378	179	—	—	—	144	480858	14.675	—	14.390
s9234	211	—	—	—	77	156189	4.767	—	26.313
s15850	534	—	—	—	165	652622	19.930	—	298.961
s38584	1426	60	339682	10.366	97	630405	19.238	85.6%	388.625
s38417	1636	156	637147	19.444	212	610315	18.625	-4.21%	2319.047
s35932	1728	160	1310720	40.000	256	2097152	64.000	60.0%	1407.616

Table 3. Experimental Result for ISCAS'89 (8 nodes).

		[8]			Proposed method				
Name	# of FF	# of VN	Amount of VS	Restoration ratio	# of VN	Amount of VS	Restoration ratio	Δ	Time(s)
s5378	179	—	—	—	160	589602	8.996	—	35.906
s9234	211	—	—	—	118	470714	7.182	—	75.188
s15850	534	—	—	—	428	1587340	24.221	—	764.453
s38584	1426	77	429749	6.557	162	914904	13.960	112.9%	802.922
s38417	1636	231	721152	11.004	487	1220222	18.619	69.2%	5285.281
s35932	1728	288	2359296	36.000	322	2498682	38.127	5.91%	5251.140

Table 4. Experimental Result for ISCAS'89 (16 nodes).

		[8]			Proposed method				
Name	# of FF	# of VN	Amount of VS	Restoration ratio	# of VN	Amount of VS	Restoration ratio	Δ	Time(s)
s5378	179	—	—	—	179	619393	4.726	—	74.953
s9234	211	—	—	—	155	612343	4.672	—	148.193
s15850	534	—	—	—	469	1743525	13.302	—	1654.563
s38584	1426	91	487093	3.716	221	1137593	8.679	133.6%	2826.015
s38417	1636	303	949210	7.242	652	1862074	14.206	96.2%	11731.828
s35932	1728	304	2424832	18.500	386	2760826	21.064	13.9%	10496.182

Table 5. Experimental Result for ISCAS'89 (32 nodes).

large amount of internal flip-flops and they are also on the front end of the circuit, which can easily restore many missing states.

Finally, it should be noted that, the reason why the restoration ratio reported in [8] is much higher than that are reported here is that [8] did not consider the effect of the non-functional mode selection signals (e.g., 'RESET'), and provide random values for them during simulation. With the same simulation strategy (which would not occur in functional mode), our method will also select such signals and the restoration ratio is also quite high, as depicted in Column 5 of Table 6. Please note that this table is provided here simply for comparison, this situation would not occur in real debug scenario.

Name	# of FF	# of VN	Amount of VS	Ratio	Ratio [8]
s38584	1426	1424	3654828	111.53	80.60
s35932	1728	1728	7079764	216.06	191.59

Table 6. Experimental Results for s38584 and s35932 with 8 selected nodes (Special Case).

6 Conclusion

In this paper, we propose a new automated trace signal selection methodology to enhance visibility in post-silicon validation. In particular, we define the gate-level restorabilities in a theoretically-precise manner and the proposed algorithm based on our definitions is able to restore much more missing states when compared to [8], as shown in our experimental results for ISCAS'89 benchmark circuits.

7 Acknowledgements

This work was supported in part by the General Research Fund 417406, 417807, and 418708 from Hong Kong SAR Research Grants Council (RGC), in part by National Science Foundation of China (NSFC) under grant No. 60876029, in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme, and in part by the National High Technology Research and Development Program of China (863 program) under grant no. 2007AA01Z109.

References

[1] M. Abramovici, et al. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pp. 7–12, 2006.

[2] Altera Inc. Design Debugging Using the SignalTap II Embedded Logic Analyzer. http://www.altera.com.

[3] ARM Ltd. How CoreSight Technology Gets Higher Performance, More Reliable Product to Market Quicker. http://www.arm.com.

[4] A. B. T. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems on-Chip: A Review. *IEE Proceedings, Computers and Digital Techniques*, 153(4):197–207, July 2006.

[5] Y. C. Hsu, et al. Visibility Enhancement for Silicon Debug. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pp. 13–18, 2006.

[6] D. D. Josephson. The Manic Depression of Microprocessor Debug. In *Proceedings IEEE International Test Conference (ITC)*, pp. 657–663, 2002.

[7] D. D. Josephson and B. Gottlieb. The Crazy Mixed up World of Silicon Debug. In *Proceedings IEEE Custom Integrated Circuits Conference (CICC)*, pp. 665– 670, 2004.

[8] H. F. Ko and N. Nicolici. Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation. In *Proceedings ACM/IEEE Design, Automation, and Test in Europe (DATE)*, pp. 1298–1303, 2008.

[9] X. Liu and Q. Xu. On Reusing Test Access Mechanisms for Debug Data Transfer in SoC Post-Silicon Validation. In *Proceedings IEEE Asian Test Symposium (ATS)*, pp. 303–308, 2008.

[10] MIPS Technologies Inc. EJTAG Trace Control Block Specification. http://www.mips.com.

[11] G. Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pp. 892–902, 1999.

[12] S. Tang and Q. Xu. A Multi-Core Debug Platform for NoC-Based Systems. In *Proceedings ACM/IEEE Design, Automation, and Test in Europe Conference (DATE)*, pp. 870–875, 2007.

[13] B. Vermeulen, T. Waayers, and S. Bakker. IEEE 1149.1-Compliant Access Architecture for Multiple Core Debug on Digital System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pp. 55–63, 2002.

[14] Xilinx Inc. Chipscope Pro Software and Cores User Guide. http://www.xilinx.com.