A Technique for Electrical Error Localization with Learning Methods During Post-silicon Debugging

Binod Kumar[1], Kanad Basu[2] and Virendra Singh[1]
Indian Institute of Technology Bombay, Mumbai[1], New York University, USA[2]
Email: {binodkumar, viren}@ee.iitb.ac.in[1], kb150@nyu.edu[2]

***Abstract*—Error localization is a challenging step in the process of post-silicon validation owing to modern design complexity. This is exacerbated by the limited visibility of internal signals at the post-silicon validation stage. Incorporated design-for-debug features and off-line techniques assist in system-level error localization for processor based systems. However, for general SoCs and special purpose IPs, error localization at the netlist level is a challenging problem. This paper proposes a machine learning based error localization methodology during the debug step. Using limited trace data, unknown signal states are discovered with the help of cluster formation by utilizing k-nearest neighbors algorithm. These clusters assist in enhancing the internal signal state visibility to the maximum extent. We derive features from the enhanced debug data set to understand the nature of the injected bug and the erroneous flip-flop responses, which are then utilized to achieve spatial error localization.**

***Index Terms*—Clustering, Post-silicon validation, Machine learning, Outlier detection, Error localization.**

I. Introduction

Due to the tremendous growth in design complexity of modern designs and the aggressive time-to-market window, more than one spin is invariably required in many design projects [1]. The average number of respins can be brought down by an effective post-silicon validation step. After an error (in the form of system crash etc.) is detected, the localization and identification of root cause of the problem is of utmost importance for the overall success of the post-silicon validation process. However, the major challenge in error localization at post-silicon stage is the highly restricted visibility of the internal signals of the design [2]. Therefore, design-for-debug structures like on-chip recorders, trace buffers or embedded logic analyzers are essential for at least partially overcoming the obstacle of the limited observability. Error localization can be attempted at different design abstraction levels. For processor based systems, this step becomes relatively easier when a carefully chosen set of internal signals are traced and a systematic methodology to analyze these dumps is adopted. However, the debugging and localization effort varies greatly with the abstraction level. High level error localization (in the form of transactions across soC interfaces or a modular approach based on the RTL model) can be achieved through interpretation of the obtained error traces. However, low-level error localization becomes very difficult for general (non regular, non processor based) designs. This is typically due to the minuscule amount of internal observability available through any on-chip mechanism because of associated area penalty issues. In this work, we consider incorporation of few (say, 32 in number) on-chip trace buffers (having depth corresponding to 1024 cycles) as the observability enhancement mechanism [3]. In spite of the diminished visibility of internal signals, state restoration (signal reconstruction based on application of logic implication principles on the netlist and utilizing the traced signal states) technique [4], [5] can be applied to increase the state visibility to a certain extent only (approximately 20-30% of the total signal states for large designs as observed in our experiments).

This paper proposes a methodology of netlist level error localization based on machine learning techniques which utilizes the expansion of debug data. Since we apply our methodology on the top of expanded restored data, we rely on the completely discovery of the internal signal (flip-flop) states. After the signal state of all flip-flops of the design are discovered, the error localization step becomes relatively easier. Since analysis of debug data ranging in a large number of cycles for a complex design becomes cumbersome, we derive important characteristics (*features*). We assume that a reference signature is available and thus golden features can be computed. After computation of features from the enhanced debug data, the infected flip-flop can be localized (for bit-flip and stuck-at errors). Specifically, the contribution of this paper can be outlined as follows:

- a methodology is presented to segregate large error traces into important traits to enhance error localization.
- different debug methodologies are proposed which assist in obtaining a list of suspects.

The remainder of the paper is organized as follows. Section II summarizes related work on post-silicon error localization methodologies. Section III presents the clustering-based visibility enhancement methodology and its applicability to the error localization step. Section IV describes the learning-based proposed debug methodology and the associated debug data analysis. Section V explains the details of our experimental set-up and presents the metrics we utilize for evaluation along with error localization results. Section VI finally concludes the paper with directions on some future research.

978-1-5386-7466-6/18 $31.00 © 2018 IEEE

II. Related Work

Friedler et al. [6] proposed a method for automatic architectural localization of post-silicon test-case failures by mismatch of states from chip execution and the states obtained from executing the similar test on an Instruction Set Simulator (ISS). Leveraging this information helps in identification of a set of instructions that could lead to the faulty final state in a buggy microprocessor design. Foutris et al. [7] utilized diversity in Instruction Set Architecture (ISA) to expose design and electrical bugs in processor systems by execution of random instruction tests and corresponding equivalent random instruction tests. Wagner et al. [8] proposed *Reversi* which involves inverse state computation for instruction blocks for detecting and subsequent localization of bugs. A major benefit of this self-checking approach is that time-consuming pre-silicon simulation results are not needed. Clearly, these kind of approaches are not applicable for general digital blocks within complex SoCs. Park et al. [9] proposed a technique named as *Instruction Footprint Recording and Analysis (IFRA)* for error detection in processor systems using low cost on-chip recorders as an observability mechanism. They perform program analysis after constructing bug localization graphs [10] for error localization to a block-level granularity. However, typically an architectural block may contain an SoC or IP. This makes it very difficult to localize/debug at the netlist level with techniques like *IFRA* [9] or that of *Reversi* [8].

In the recent years, machine learning techniques [11] have been utilized for the purpose of bug triaging or error localization at both the pre-silicon and post-silicon stage [12]–[15]. Poulos et al. [12] proposed grouping of different pre-silicon error traces obtained by SAT-based debugging into various groups through regression analysis. Since during post-silicon validation, a large number of tests can be applied, the amount of logs collected can become very large. Therefore, machine learning techniques like clustering, regression analysis etc., can be suitably deployed to extract hints for bug/error localization from the obtained test response logs. DeOrio et al. [13] proposed a post-silicon bug diagnosis methodology based on data collection from failing tests and then applying a clustering technique to form different signal groups. A mechanism to detect root-cause signals by identifying "anomalous" signals from post-silicon tests and iterative selection of signals to be monitored has been proposed by Bertacco et al. [14] on lines similar to that of [13]. For post-silicon bugs that manifest inconsistently over repeated executions of the same test and have non-deterministic behavior, Khudia et al. [15] have classified total internal signals into *passing groups* and *failing groups* for error localization. Mammo et al. [16] proposed a diagnosis approach where execution from a reference model (Instruction set simulator) and synthetically bug injected signatures are utilized to build a training model. In testing phase, the developed model assists in predicting the error location with the help of carefully engineered features (with the help of design functionality behavior) from the test execution log. Such feature engineering can not be easily adopted for uncore components/non-regular designs.

For general design blocks of SoC's, utilizing state restoration [4], [17] technique assists in enhancing the restricted visibility which in turn would ease the localization principles. However, increment in state restoration does not translate to the ease in error detection/localization in the same proportions [18]. Thus, complete discovery of internal signal states can overcome this bottleneck. Iwata et al. [19] analyzed signal selection for localizing electrical bugs using trace buffers. Jindal et al. [20] proposed an error localization approach for design bugs (gate-level error models) using regression analysis of pre-silicon error signatures for smaller regions (called flip-flop zones) of the netlist. However, this method is not applicable to other functional bugs/electrical bugs and it fails to achieve exact error localization. Vali et al. [21] proposed bit-flip detection aware selection of trace signals and a SAT-based localization (of particular flip-flop and the cycle) methodology involving traces for 100 clock cycles. Their methodology involves unrolling the design netlist for that number of clock cycles. Clearly, solving the SAT instance for a large design and a larger error trace (consequently unrolling for larger number of cycles) is prohibitive. Moreover, the assumption regarding availability of a restrictive error trace of only 100 cycles (in which the flip occurred) is not practical. Thus, the proposed methodology is directed at error localization for larger error traces. Singh et al. [22] proposed electrical error localization by dividing a large design into sub-blocks and applying property checking on these smaller blocks. The proposed methodology can also be applied in a similar fashion for any large design. The work most closely related to ours is that of [23] where the authors have identified clusters (of flip-flops) in the design and based on the logical relation between the cluster elements and the inputs to the cluster, signal states of flip-flops can be known. This also requires the knowledge of the initial values of the elements of the cluster which is available through tracing. They achieve significant improvement in state restoration ratio; however, our goal is to obtain all the unknown signal states which can lead to a completely enhanced internal visibility of the design. Furthermore, error localization is not attempted in [23]. Note that the proposed methodology is orthogonal to the clustering mechanism of [23] and it can be applied in conjunction with the proposed error localization technique.

III. Learning based Post-silicon internal visibility enhancement

Due to area overhead constraints, very few (typically, 1%) of the total number of internal signals can be traced. This leads to a highly limited quantity of debug data. Consider the circuit shown in Fig. 1 which has six flip-flops. If trace-buffer width (i.e., number of signals traced) and depth (i.e., number of clock cycles traced) is chosen as 2 and 10 respectively, only 2 of these six flip-flops can be traced for ten clock cycles. Let's assume that flip-flop B and F are selected for tracing and thus their signal states are known to us. We then apply the state restoration technique described in [4], so that few states of

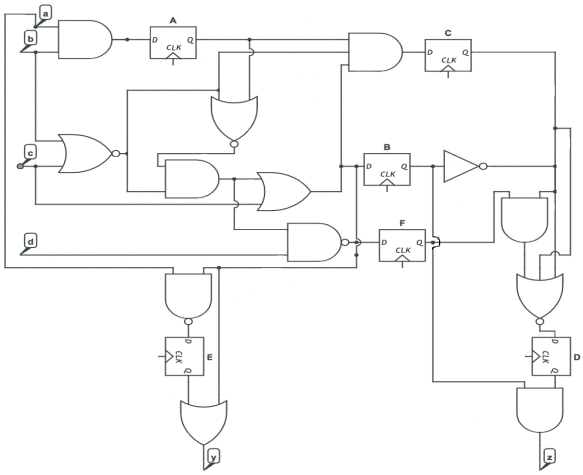

Fig. 1. Example circuit for illustrating methodology

other flip-flops can also be known. Table-I shows states of these six flip-flops are for ten clock cycles. The signal state of many flip-flops for many clock cycles can not be discovered. Those states which can not be restored are denoted as **X**. All the known signal value sum upto only 47% of the total signal states. Note that the primary inputs and outputs are not traced here for explicit signal state discovery.

TABLE I
RESTORED AND TRACED STATES FOR EXAMPLE CIRCUIT

Flip-Flop→	A	B	C	D	E	F
cycle 1	X	**0**	X	X	X	**0**
cycle 2	X	**0**	X	X	X	**1**
cycle 3	X	**1**	1	X	X	**1**
cycle 4	X	**1**	1	0	X	**1**
cycle 5	X	**0**	X	0	X	**1**
cycle 6	X	**1**	1	X	X	**1**
cycle 7	X	**0**	X	0	X	**1**
cycle 8	X	**1**	1	X	X	**1**
cycle 9	X	**0**	X	0	X	**1**
cycle 10	X	**0**	X	X	X	**1**

Assuming that we have simulation data corresponding to the design from its higher level abstraction (termed as reference/golden signatures), we need to alleviate the mismatch between the extent of pre-silicon data (full visibility) and the actual debug data (restricted visibility). Recently, Jindal et al. [24] have proposed a two step methodology for enhancing the restricted visibility. Their methodology involves a machine learning based clustering technique which assists identifying flip-flops similar to each other. Once the related signals (within the corresponding cluster) of a particular signal are identified, depending of their traced/restored states, the signal state of this flip-flop can be obtained. For the example circuit, we need to know the value corresponding to **X**'s (total 32 in number) depicted in Table-I in order to achieve complete visibility of internal signal states. Similarly, we can attempt a clustering wherein a set of flip-flops are identified for each flip-flop during the *training* phase (model building phase). The signal state of this target flip-flop can be discovered by a majority vote of the signals present in the cluster in which the particular signal is present. Maintaining a reasonable accuracy in signal state discovery is quite difficult task and depends on the clustering technique. Suppose a bit-flip occurs in some cycle at one of the traced signals and few other signal states are restored based on this flipped value. When unknown states are discovered based on these values, they would be different from their actual values. This unintended mismatching of some of the internal signal values can lead the debug process towards spurious cases increasing the overall debug effort. Thus, we want the state discovery process to be as accurate as possible.

A design simulation based clustering model is developed, details of which are mentioned in next subsection. Based on simulation of the design for certain number of clock cycles, the Euclidean distances between different signals (flip-flops) are computed. However, as stated before, it is extremely difficult to segregate these signals into non-overlapping clusters. This would have been if flip-flops in the design are separable with in disjoint groups (divisions). Clearly, this is not a characteristic of most designs. We performed simulation of the example circuit for n clock cycles.[1] Then, this simulation data is fed to the Nearest Neighbor clustering algorithm. The neighbors obtained for each of the flip-flop is shown in Table V where we have depicted three neighbors in each case. Thus, the unknown signal state is obtained based on the majority vote of the signal values of these three neighbors. Entries on X-coordinate and Y-coordinate indicate distances. The distances between different signals represent the similarity between them. Note that cluster corresponding to FF-E has only one element. In this case, the values for this flip-flop is to be discovered through near by clusters. Thus, the signal state (either 0/1) of a particular flip-flop in any cycle is assigned the value most common among states (which are known either through restoration/tracing) of the other signals of the cluster. Consequently, the signal state of all flip-flops for all the clock cycles can be obtained. In the experimental section, we provide detailed results of different options of this filling exercise and substantiate the merits of proposed methodology.

TABLE II
OBTAINED NEAREST NEIGHBORS FOR EXAMPLE CIRCUIT

FF→	nbr1	nbr2	nbr3
A	F	C	B
B	C	E	D
C	F	D	B
D	E	B	C
E	D	C	B
F	C	A	B

Table III shows the discovered signal states with this process. The italicized signal states for these signals are different from their actual values. Hence, a italicized 1 means the actual value is opposite (i.e., 0) and similarly for a italicized 0 value. After identifying the neighbors of each signal, the filling order becomes important for the process of internal signal state discovery. We assume here a filling order of the signals from a to f. Depending on the subsequent filling of internal signal

[1]This number would vary depending on size of circuit. For smaller ISCAS circuits, 50-100 cycles is enough.

states, a majority vote is decided based on these signal values. If both the values are equally probable, one of them need to be given priority depending on the debug engineer's choice. Note that it is difficult to show the merits of the clustering methodology because of smaller number of flip-flops. This is because cluster sizes would vary in larger quantities for large designs and decisions for discovery process need to be taken based on a variable number of flip-flops unlike this case.

TABLE III
RESTORED AND TRACED STATES FOR EXAMPLE CIRCUIT

Flip-Flop→	A	B	C	D	E	F
cycle 1	0	**0**	0	0	0	**0**
cycle 2	1	**0**	*1*	*1*	*1*	**1**
cycle 3	1	**1**	1	*1*	*1*	**1**
cycle 4	1	**1**	1	0	*1*	**1**
cycle 5	1	**0**	*0*	0	*0*	**1**
cycle 6	*1*	**1**	1	1	1	**1**
cycle 7	1	**0**	0	0	0	**1**
cycle 8	*1*	**1**	1	*1*	1	**1**
cycle 9	1	**0**	*0*	0	0	**1**
cycle 10	*1*	**0**	1	1	1	**1**

As we observed from Table III, we have wrongly discovered 13 signal states in total. We have chosen the signal state of 1 as the signal state in case of equal probability of both the signal values. Thus, we obtain complete internal signal visibility albeit some signal states are wrong from the actual.

IV. PROPOSED DEBUG METHODOLOGY

A. Feature identification for ease in debugging

Since the completely expanded internal visibility is not highly accurate, a straight forward comparison between enhanced test signatures (from execution trace) and reference golden signature is not possible. A cycle by cycle comparison for all the flip-flops for large traces would turn out to be tedious for the purpose of anomaly (i.e., the suspect flip-flop) detection. We extract important traits from the enhanced debug data, which are termed as *features* of the completely expanded internal visibility. We target functional (bit-flip) and electrical (stuck-at) errors leading to feature identification aimed at their localization. Although this extraction of features can be carried out in any manner, it is required to perform this extraction in a computationally effective fashion. In fact, an optimal feature identification is an important choice for the debug engineer for the overall success of the error localization process. Five features (ξ) are identified as depicted below for characterizing the enhanced debug data, for each flip-flop:

ξ_1-number of 1's across all the cycles
ξ_2-first cycle in which 1 is observed
ξ_3-cycle from which consecutive run of 1 is observed
ξ_4-number of occurrences of k-length consecutive run of 1
ξ_5-maximum length of consecutive run of 1

We chose k as 4 based on empirical observations for ξ_3. For this feature, we observed k-length consecutive run of 0 also yields same localization results. In fact, all the above features would have similar trend in variation if we consider 0 instead of 1. This is because of the discrete nature of data where in absence of 1 denotes presence of 0 resulting in an overall similar pattern (to the case considered above) for different signals of the design. Note that it is difficult to illustrate all the above features with the example circuit (shown in Fig.1) and the corresponding discovered states considered in previous section. Hence, we consider a hypothetical example with eight flip-flops (FF1, FF2, FF3, FF4, FF5, FF6, FF7 and FF8) and consider a visibility expansion for 10 clock cycles.

TABLE IV
FEATURE VALUES FOR ILLUSTRATED TRAINING VECTOR

FF→	FF1	FF2	FF3	FF4	FF5	FF6	FF7	FF8
$\xi 1$	6	4	5	7	5	6	6	0
$\xi 2$	1	2	1	2	1	3	2	0
$\xi 3$	3	2	3	2	0	3	4	0
$\xi 4$	1	0	2	3	2	1	2	0
$\xi 5$	2	3	2	2	0	3	4	0

Example values of features of these eight flip-flops are shown in Table IV. Note that for flip-flop FF8, the feature values have zero values as signal states are 0s across all the ten clock cycles. For this scenario, all the features have their value as zero. Note that features ξ_1 and ξ_5 have a maximum value of 10 for this scenario. For large error traces ranging from 10000/20000 clock cycles and beyond, we create smaller chunks consisting of t cycles (such that T, length of complete error trace can be expressed in multiples of t). Then, features are computed based on the separate analysis of these chunks of t cycles. This simplifies the debug process as comparative analysis needs to be done for N (total number) flip-flops each with r (=T/t) feature blocks. For instance- if T is 10000, and if features are computed over 400 clock cycles, then there are 25 feature blocks in total. Thus, as per terminology introduced earlier, t is 400 and r is 25 for this example scenario.

After calculation of the individual features, we obtain the average values of i^{th} signal (ff_i) in the following manner:

$$avg.\ feature(ff_i) = \frac{\sum_{j=1}^{j=max} \xi_j(ff_i)}{max} \quad (1)$$

Note that if we consider only four features as described above, then the average value can be obtained as above by setting max as 4 instead of 5. We observed that individual feature analysis (instead of averaged) also succeeds in error localization (i.e., anomaly detection). However, the number of suspect candidates obtained vary with separate consideration of each feature as compared to the case when localization is done with their average values.

B. Proposed methodology for anomaly detection

The proposed methodology utilizes the fact that the feature values would change due to the propagation of the error (functional/electrical bug). Table V shows an illustration of the debug data obtained after feature computation from the completely enhanced visibility (with the clustering methodology outlined in [24]). The numbers in this table represent

the corresponding averaged feature values $avg.\ feature(ff_i)$, computed over a block size of t clock cycles. For instance, the averaged value of the features for a duration of t clock cycles for 1^{st} flip-flop is 155.5 and so on. A feature block contains the averaged features corresponding to N flip-flops, computed over a duration of t clock cycles. In total, there are r feature blocks shown in 1^{st} column of Table V where FBN stands for feature block no.[2]

TABLE V
DEBUG DATA AFTER FEATURE COMPUTATION

FBN↓ \ FF →	ff_1	ff_2	..	..	ff_{N-1}	ff_N
1	155.5	164.3	..	..	172.2	164.3
2	132.7	142.3	..	..	139.4	162.7
..	..	..	..	..	..	..
...	..	..	..	..	..	..
r	168.4	145.7	..	..	158.6	111.9

With debug data as represented above in the form of averaged feature values, comparisons between golden (reference signatures) and the error trace (completely expanded) can be done. We propose 3 methods to find out the set of anomalous (suspect) candidates which are the flip-flops at which bit-flips happened or stuck-at error error happened (details of these error models are in Section V-B). Since we are targeting spatial localization, the ideal case is to obtain one flip-flop (corresponding to exact localization), we obtain a small subset of flip-flops (suspect candidates) which includes the actual infected signal. With the help of the complete visibility enhanced debug data, localization of a bit-flip (or, stuck-at) error becomes a multi-class classification problem which can be solved by distance or density based segregation technique [25], [26]. We utilize different set of approaches for the proposed debug methodologies. The first approach is different from these traditional methodologies and utilizes the concept of isolation [26]. The second debugging approach is based on measurements with *Euclidean* distance.

1) ***First method for anomaly detection from debug data****:* Anomalies (erroneous behavior) can be described as data patterns that have different data characteristics as compared to the normal data points [11]. Thus, when a model is built using features of reference (non-buggy) execution, it can catch anomalies given the features computed from buggy debug data. We utilize an isolation based method which measures susceptibility of individual data points to be isolated. Thereby, anomalies can be identified as those that have the highest susceptibility [26]. This method is based on an approach called *Isolation Forest (IsoForest)* which builds an ensemble of isolation trees for the given data set. Thus, anomalies are those instances which have short average path lengths on the isolation trees. In practice, when a forest of random trees collectively produces shorter path lengths for particular samples, they are highly likely to be anomalies. This approach is presented in $IsoForest\ based\ detection$ as Algorithm 1.

[2]This Table is not related to Table IV and the numbers shown here are just for illustration of the averaged feature values.

Algorithm 1: $IsoForest\ based\ detection(Method1)$

Input: CVM, $reference\ signatures$
Output: $suspect\ ff$

1 $CVM \leftarrow$ debug data i.e., completely filled visibility matrix (having N rows and T columns);
2 $feature\ blocks \leftarrow$ average features for t cycles;
3 $ref.\ feature\ blocks \leftarrow$ average features for t cycles calculated from $reference\ signatures$;
4 $M \leftarrow$ initialized $Isolation\ forest$ (with maximum sample parameter as N);
5 Train model M for fitting with $ref.\ feature\ block$;
6 $suspect\ ff \leftarrow$ Predict outliers for given debug data, $feature\ block$ utilizing M;

As we aim at localizing transient errors (modeled as bit-flips), the challenge here lies in the fact that a single bit-flip may cause only subtle changes in the debug data which becomes difficult to localize because of inaccurately discovered states (due to the clustering process). This also applies to single permanent errors (modeled as stuck-at 0/1). Hence, a machine learning based method can assist in refining the debug data and pin-pointing to the suspect location (flip-flops). However, as illustrated in experimental result section, it is extremely difficult to localize only to the actual flipped signal. With the help of this approach, we succeed in obtaining a set of suspect signals ($suspect\ ff$) including the actual one.

2) ***Second method for anomaly detection from debug data****:* We propose a second method which is based on distance measure between the features of reference signatures and features computed from the visibility enhanced execution data (traced+restored signal states). One of the most commonly used measures, Euclidean distance is utilized for this purpose, which is computed by the following equation:

$$d_{ff_i} = \sqrt{\sum_{k=1}^{k=r}(p_i - q_i)^2} \tag{2}$$

Here, d_{ff_i} refers to the Euclidean distance corresponding to i^{th} flip-flop for which p_i and q_i represent feature blocks computed from debug data and reference signature respectively. Flip-flops (signals) having highest Euclidean distance are most likely to be outliers (suspect candidates). Using the set-up similar to what presented in $Method1$, this methodology is shown next as Algorithm 2.

3) Spatial error localization with proposed methodology: One of the important issues to address towards the targeted spatial localization is the similarity/dissimilarity among the suspect candidates provided by $Method1$ and $Method2$. The ideal case is that the two methods contain the actual (bit-flipped) signal, or the stuck-at signal (for the second error model) and the intersection of the candidates from these methods give a very smaller list of suspects (ranging between 1 to 10). Suppose $Method1$ and $Method2$ provide suspect

Algorithm 2: *Euclidean dist. based detection*($Method2$)

Input: CVM, $reference\ signatures$
Output: $suspect\ ff$

```
Obtain feature blocks from CVM and
ref. feature blocks;
for each signal 1 to N do
    for each feature block 1 to r do
        Calculate Euclidean distance from reference
        signature feature block;
    end
end
Sort all signals as per their distance;
suspect ff ← top ranked signals;
```

candidates as S_{M1} and S_{M2} respectively where in each of the suspect list contains ff_{actual}, the actual signal where bit-flip occurred (or, stuck-at error happened). The final list of suspect flip-flops (S_{final}) is given by the following equation:

$$S_{final} = S_{M1} \cap S_{M2} \tag{3}$$

However, in some of difficult to localize errors, it may happen that only one of the methods contain the actual signal, ff_{actual} and the other method gives a suspect list containing signals in the proximity of ff_{actual}. In this case, finding intersection of two lists, S_{final} is not much profitable. We observed that such cases are not very frequent. However, these methods ensure that actual suspect is caught by at least one of them. The detailed experimental findings are reported in next section.

C. Complete flow of the proposed methodology

We present the complete flow of the proposed methodology of visibility enhancement-based error localization in Fig. 2. Note that for finding clusters to enhance the internal visibility, simulation data for a small number of clock cycles (represented as t' in the range of 100-1000) is sufficient where as the actual error trace may be as long as 10000 cycles (represented as T) and beyond. As stated earlier, features are computed for smaller block of clock cycles (represented as t). Such blocks total r in number. One shortcoming of this methodology is that reference (golden) signature are required for comparison. In fact, embedding self-checking approach based localization is one of the major direction of our future investigation.

V. Experimental results and observations

A. Experimental Setup

We have chosen 10 large circuits from a variety of benchmark suites (ISCAS'89, ITC'99 and Opencore [27]) are selected. The characteristic of these benchmark circuits are noted in Table VI. Because of varying number of sequential and combinational elements in each of them, they serve as representatives of SoC blocks. We performed most of the scripting tasks in Python and utilized the learning algorithms implementation of *Scikit-learn* [28] which also utilizes Python.

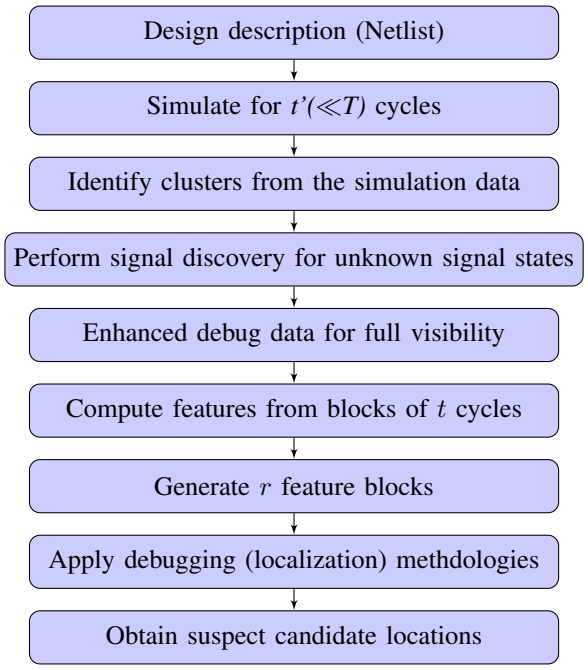

Fig. 2. Overall flow of the proposed methodology for internal signal visibility enhancement-based debugging

TABLE VI
Characteristics of benchmark circuits

Benchmark Suite	*Name*	*No. of FFs*	*No. of Gates*
ITC'99	$b17$	864	13988
ITC'99	$b21$	429	13098
Opencore	$p16c5x$	739	4107
Opencore	$mips$	1859	38190
Opencore	usb	1761	10650
Opencore	$softusb$	317	4718
ISCAS'89	$s15850$	597	9772
ISCAS'89	$s35932$	1728	16065
ISCAS'89	$s38417$	1636	22179
ISCAS'89	$s38584$	1452	19253

For all logic simulation tasks, we developed our own simulator in C language that takes in circuit descriptions (in .bench format). The same simulator was extended to perform signal restoration on the traced data.

B. Error models utilized in experiments

We considered two error models for representing functional and electrical failures observed during post-silicon validation of typical ICs. Single bit-flip is widely utilized in previous work [7] for modeling functional errors. As a crude approximation, stuck-at error model can be used to represent electrical bugs/errors [29]. The bit-flip error model is fairly straight forward and our goal is to localize to that particular flip-flop at which bit-flip occurred. For the stuck-at error model, we need to localize to the flip-flop if the stuck-at error happens at the input of the flip-flop otherwise it suffices to localize the flip-flops in the corresponding inward logic cone if the stuck-at error happens at any combinational net (wire).

C. Error Localization results

As is mentioned in Section IV-B3, the two methods of anomaly detection provide a suspect list which is represented

by S_{M1} and S_{M2} respectively. We perform error injection experiments for 4 different length of error traces (T) as 2000, 2500, 5000 and 10000 clock cycles respectively. In total, we perform 10 error injection experiments for each of the error models (bit-flip and stuck-at) for each benchmark circuit. For the bit-flip experiments, the flip-cycle and the particular flip-flop are chosen randomly. We report the average number of each of S_{M1}, S_{M2} and their corresponding intersection, S_{final} in Table VII. Note that all the values reported are in integers as they have been accordingly rounded off to the corresponding next value. We verified that each of the suspect list contains ff_{actual} in almost all the cases.

TABLE VII
LOCALIZATION RESULTS FOR SINGLE BIT-FLIP ERROR

$Name$	N	S_{M1}	S_{M2}	S_{final}
$b17$	864	34	18	8
$b21$	429	28	20	6
$p16c5x$	739	38	26	11
$mips$	1859	52	34	24
usb	1761	56	46	31
$softusb$	317	22	18	9
$s15850$	597	30	26	13
$s35932$	1728	63	42	23
$s38417$	1636	52	38	27
$s38584$	1452	69	53	23

As is evident in above Table, the number of suspect signals in S_{M2} are least among the two for all cases. However, in cases when it fails to localize the actual suspect, Table VIII shows the average number of each of S_{M1}, S_{M2} and their corresponding intersection, S_{final} for single stuck-at error cases. We injected stuck-at error at the input of a randomly selected flip-flop. In some of the iterations, we attempted stuck-at error at any randomly selected wire. In this case, the proposed technique localizes to the flip-flops in the corresponding inward logic cone. This is the reason why the values are higher in Table VIII. Similar to the bit-flip scenario,

TABLE VIII
LOCALIZATION RESULTS FOR SINGLE STUCK-AT ERROR

$Name$	N	S_{M1}	S_{M2}	S_{final}
$b17$	864	30	22	13
$b21$	429	22	27	11
$p16c5x$	739	34	34	17
$mips$	1859	47	39	33
usb	1761	52	42	25
$softusb$	317	26	27	9
$s15850$	597	46	34	15
$s35932$	1728	67	58	23
$s38417$	1636	71	56	19
$s38584$	1452	81	62	31

S_{M2} has least candidates however in fewer cases only. Higher values in case of S_{M1} indicate that the model M could not be properly trained with $Method1$.

D. Attempting temporal localization

It is worth to note that although we did not explicitly target temporal localization, we attempted finding the bit-flip cycle in terms of the particular feature block. Since a feature block corresponds to actual data points for t cycles, localizing the particular feature blocks hints that bit-flip cycle lies with in these t cycles. In many of our experiments, applying the proposed methodologies by slight adaptation (testing for feature block cycles i.e., *rows* instead of flip-flops i.e., *columns* of Table V) yields correctly the flipped feature block cycle. However, a block duration of t cycles may not be very useful for the debug process. Hence, finer temporal localization (say, within 10-50 cycles of the flip-cycle) is needed. Self-checking approaches can assist in capturing the flipped cycle by analyzing deviations within r feature blocks.

E. Directions for extending the proposed methodology

The proposed methodology of electrical error localization can be attempted with different signal selection techniques, either at gate-level [4], [5], [30] or at register-transfer level [31], [32]. Recently, Mandouh et al. [33] have proposed an approach of bug triaging at register-transfer level (RTL) by utilizing big data analysis. We believe that a similar big data analysis can be adopted for localization of electrical bugs. Note that we do not report difference in the error localization results for different signal selections. However, we observed minuscule difference in error localization of bit-flips for different trace signal selection methodologies. However, it has been reported that different trace signal selection techniques have different error localization/detection capabilities for design error models [34]–[36]. Liu et al. [37] have proposed signal restoration algorithms to increase the amount of restored signal states which can definitely assist in enhanced error localization. Note that since the signal discovery methodology of [24] leads to inaccurate discovery to some extent, improvement in accuracy would enhance the proposed error localization methodology. We believe that a joint application of [23] and [37] would assist in the accuracy of unknown signal state discovery methodology proposed in [24].

VI. Conclusion

This paper proposed a methodology for error localization of functional and electrical errors under the limited post-silicon visibility. To achieve error localization, internal signal visibility is enhanced through a recently proposed clustering based methodology. In this visibility enhancement methodology, the unknown signal states are discovered based on the values of closely related known signals which are identified through nearest neighbor algorithm. The enhanced debug data is analyzed into features defined for assisting in error localization. Machine learning based approaches are proposed which build a model based on these features extracted from the traced and restored debug data. The localization techniques succeed in providing a small list of suspect candidates for functional bit-flip errors and stuck-at errors resulting due to electrical effects. As pointed out in previous subsection, the debug methodologies presented in this paper can be extended into many directions. Since the proposed methodology achieves spatial localization only, it needs to be extended for temporal localization. This can be achieved through better machine learning-based training of the model. In the proposed methodology, we assume the presence of golden responses available

from simulation of a high level model of the design, which is very difficult to obtain. So, another important direction of extension of the proposed technique is to achieve localization without requiring reference (golden) signatures. Additionally, it is needed to refine the proposed machine learning based debug techniques to further prune the list of suspect candidates and achieve exact spatial error localization. We expect that better training of the learning model with the completely expanded internal visibility can achieve both temporal and spatial error localization in an enhanced manner.

REFERENCES

[1] H. Foster, "2014 wilson research group functional verification study," https://blogs.mentor.com/verificationhorizons/blog/2015/01/21, 2014.

[2] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 12–17.

[3] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for socs," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 7–12.

[4] K. Basu and P. Mishra, "Rats: Restoration-aware trace signal selection for post-silicon validation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 605–613, April 2013.

[5] M. Li and A. Davoodi, "Multi-mode trace signal selection for post-silicon debug," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2014, pp. 640–645.

[6] O. Friedler, W. Kadry, A. Morgenshtein, A. Nahir, and V. Sokhin, "Effective post-silicon failure localization using dynamic program slicing," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.

[7] N. Foutris, D. Gizopoulos, M. Psarakis, X. Vera, and A. Gonzalez, "Accelerating microprocessor silicon validation by exposing isa diversity," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2011, pp. 386–397.

[8] I. Wagner and V. Bertacco, "Reversi: Post-silicon validation system for modern microprocessors," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, Oct 2008, pp. 307–314.

[9] S. B. Park and S. Mitra, "Ifra: Instruction footprint recording and analysis for post-silicon bug localization in processors," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, June 2008, pp. 373–378.

[10] S. B. Park, A. Bracy, H. Wang, and S. Mitra, "Blog: Post-silicon bug localization in processors using bug localization graphs," in *Design Automation Conference*, June 2010, pp. 368–373.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[12] Z. Poulos and A. G. Veneris, "Clustering-based failure triage for RTL regression debugging," in *2014 International Test Conference, ITC 2014, Seattle, WA, USA, October 20-23, 2014*, 2014, pp. 1–10.

[13] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco, "Machine learning-based anomaly detection for post-silicon bug diagnosis," in *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, 2013, pp. 491–496.

[14] V. Bertacco and W. Bonkowski, "Ithelps: Iterative high-accuracy error localization in post-silicon," in *33rd IEEE International Conference on Computer Design, ICCD 2015, New York City, NY, USA, October 18-21, 2015*, 2015, pp. 196–199.

[15] A. DeOrio, D. S. Khudia, and V. Bertacco, "Post-silicon bug diagnosis with inconsistent executions," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2011, pp. 755–761.

[16] B. Mammo, M. Furia, V. Bertacco, S. A. Mahlke, and D. S. Khudia, "Bugmd: automatic mismatch diagnosis for bug triaging," in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016*, 2016, p. 117.

[17] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 285–297, Feb 2009.

[18] S. Ma, D. Pal, R. Jiang, S. Ray, and S. Vasudevan, "Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection," in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, Nov 2015, pp. 1–8.

[19] K. Iwata, A. M. Gharehbaghi, M. B. Tahoori, and M. Fujita, "Post silicon debugging of electrical bugs using trace buffers," in *2017 IEEE 26th Asian Test Symposium (ATS)*, Nov 2017, pp. 189–194.

[20] A. Jindal, B. Kumar, K. Basu, and M. Fujita, "Elura: A methodology for post-silicon gate-level error localization using regression analysis," in *2018 31st International Conference on VLSI Design(VLSID'18)*, Pune, India, Jan 2018.

[21] A. Vali and N. Nicolici, "Bit-flip detection-driven selection of trace signals," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[22] E. Singh, C. W. Barrett, and S. Mitra, "E-QED: electrical bug localization during post-silicon validation enabled by quick error detection and formal methods," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, 2017, pp. 104–125.

[23] Y. Cheng, H. Li, Y. Wang, and X. Li, "Cluster restoration based trace signal selection for post-silicon debug," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[24] A. Jindal, B. Kumar, N. Jindal, M. Fujita, and V. Singh, "Silicon debug with maximally expanded internal observability using nearest neighbor algorithm," in *2018 IEEE Computer Society Annual Symposium on VLSI*, July 2018.

[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335388

[26] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ser. ICDM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 413–422. [Online]. Available: http://dx.doi.org/10.1109/ICDM.2008.17

[27] http://www.opencores.org/.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] J. S. Yang and N. A. Touba, "Automated selection of signals to observe for efficient silicon debug," in *2009 27th IEEE VLSI Test Symposium*, May 2009, pp. 79–84.

[30] K. Basu, P. Mishra, and P. Patra, "Efficient combination of trace and scan signals for post silicon validation and debug," in *2011 IEEE International Test Conference*, Sept 2011, pp. 1–8.

[31] H. F. Ko and N. Nicolici, "Automated trace signals selection using the rtl descriptions," in *IEEE International Test Conference*, 2010, pp. 1–10.

[32] B. Kumar, K. Basu, M. Fujita, and V. Singh, "Rtl level trace signal selection and coverage estimation during post-silicon validation," in *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, Oct 2017, pp. 59–66.

[33] E. El Mandouh and A. G. Wassal, "Application of machine learning techniques in post-silicon debugging and bug localization," *Journal of Electronic Testing*, vol. 34, no. 2, pp. 163–181, Apr 2018.

[34] B. Kumar, K. Basu, A. Jindal, M. Fujita, and V. Singh, "Improving post-silicon error detection with topological selection of trace signals," in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2017, pp. 1–6.

[35] B. Kumar, A. Jindal, M. Fujita, and V. Singh, "Combining restorability and error detection ability for effective trace signal selection," in *Proceedings of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '17. Banff, Alberta, Canada: ACM, May 2017.

[36] B. Kumar, A. Jindal, and V. Singh, "A trace signal selection algorithm for improved post-silicon debug," in *2016 IEEE East-West Design Test Symposium (EWDTS)*, Oct 2016, pp. 1–4.

[37] X. Liu and R. Vemuri, "Effective signal restoration in post-silicon validation," in *2017 IEEE International Conference on Computer Design (ICCD)*, Nov 2017, pp. 169–176.