Multiple-Defect Diagnosis for Logic Characterization Vehicles

Ben Niewenhuis, Soumya Mittal, and R. D. (Shawn) Blanton
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213
`https://www.ece.cmu.edu/~actl/`

Abstract*— Previous work on the Carnegie Mellon Logic Characterization Vehicle (CM-LCV) has emphasized the diagnosability properties of a specific class of regular circuits called functional unit block arrays (FUB arrays). This paper describes a multiple-defect, two-level diagnosis procedure that leverages these unique properties of the FUB array to significantly improve diagnosis. This custom diagnosis procedure is implemented and evaluated against commercial diagnosis using a simulated fault-injection experiment involving pairs of injected faults. Custom diagnosis is accurate in 98.2% of the simulations with resolution of at most five for 94.1% of the simulations, improving upon the commercial results of 70.7% accurate and 85.0% with the same level of resolution.

I. Introduction

The semiconductor manufacturing industry utilizes a wide variety of test structures to monitor the development of new fabrication processes. For digital circuits, these test structures range from the very simple, such as via arrays and comb drives meant to estimate layer defectivity, to the very complex, such as SRAMs and product-like test chips (PLTCs) which are meant to catch the complex, systematic defects that occur in flagship designs. A PLTC is typically a logic circuit that exhibits some of the features and functionality of commercial designs. Previous work has culminated in a new type of PLTC, called the Carnegie Mellon Logic Characterization Vehicle (CM-LCV) [1][2], which has improved testability and diagnosability compared to conventional PLTCs. Improvements in testability, which is the ease with which defects can be detected, have been demonstrated in [3]. Improvements in diagnosability, or the ease with which defects can be distinguished, have been demonstrated in [4]. An unexplored aspect of diagnosability is how well multiple defects can be isolated and characterized. This paper describes a two-level multiple-defect diagnosis procedure for the CM-LCV, and evaluates its efficacy using an experiment involving failures caused by two simultaneous defects.

The remainder of the paper is organized as follows: Section II provides background on the CM-LCV. Section III discusses how the properties of the CM-LCV can be exploited to gain more insight concerning multiple-defect localization. Section IV details a custom, two-level diagnosis procedure for the CM-LCV that leverages these properties. Section V describes the diagnosis experiments used to evaluate the custom diagnosis procedure, while Section VI draws some conclusions.

II. Background

The CM-LCV improves on the conventional PLTC by implementing an innovative functionality. The motivating insight behind this approach is that the manufacturing process is sensitive only to the physical features of a design. Although the functionality (logic) and the physical features (layout) are correlated, it is in general possible to implement an arbitrary logic function using many different layouts. Thus the CM-LCV uses a functionality that is designed to be highly testable and diagnosable and implements this functionality with a layout that best approximates the desired characteristics. For example, a CM-LCV layout may be designed to mimic a desired standard-cell usage distribution [5], or a product layout may be rerouted to form a CM-LCV [6]. Previous research sought to discover an optimal functionality for these purposes, and resulted in a two-dimensional array of functional unit blocks (FUBs) (e.g. Fig. 1) with the following properties:

- **Regularity** - A set of FUBs with regular connections can be designed to be C-testable [7], a property which provides strong guarantees about the test-set size and fault coverage over the entire array, regardless of its size.
- **Two-dimensions** - Arranging FUBs in a two-dimensional array with vertical and horizontal connections enables error propagation in two directions, allowing for better localization of defects within the array.
- **VH-bijective FUBs** - A VH-bijective function [1] is a bijective boolean function with two sets of inputs (vertical and horizontal) and two corresponding sets of outputs. Additionally, the function is constrained such that any value change on exactly one of the sets of inputs must result in a change in the value on both sets of outputs. Requiring that the FUB function be VH-bijective guarantees both vertical and horizontal propagation of errors and simplifies test-set construction.

A CM-LCV is thus formally defined as a collection of one or more FUB arrays along with the necessary test access mechanisms (e.g., DFT, BiST, etc.). However, it is the FUB array that determines all of the properties of interest in the current work. Fig. 1 is an example diagram that demonstrates how errors propagate through the FUB array. Inputs to the array are applied on the top and left edges, and values propagate "downstream" (i.e., down and to the right) through each of the FUBs to the array outputs on the right and bottom edges. Assume the FUB instance in the array of Fig. 1 marked with an "$\times$" is defective, and produces an erroneous value (represented by the dark shading) for some test pattern at its horizontal output. All FUBs in the same row as the defective FUB

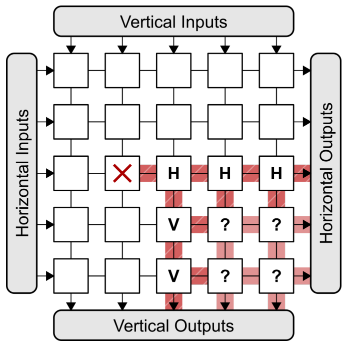


Fig. 1. VH-bijectivity demonstrated in the presence of a defective FUB in a two-dimensional array. Note that signal propagation moves left to right, top to bottom. Guaranteed error propagation is represented by the darkly shaded connections, while unknown error propagation is represented by the lightly shaded connections.

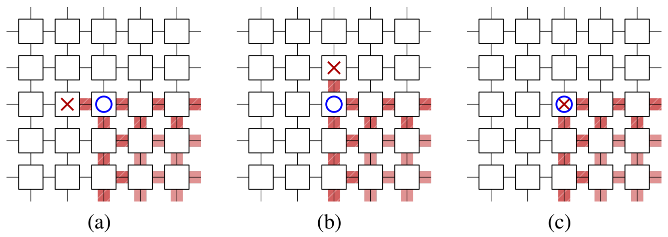


Fig. 2. The three cases that exist for the relationship between the location of a single defective FUB (represented by the "×") and the location of the intersection of the errors observed on the array edges (represented by the "○").

(marked with an "H") fall under the VH-bijective constraint: a difference in value only present at a horizontal input will propagate to both outputs. Furthermore, all FUBs in the first downstream column (marked with a "V") also fall under the VH-bijective constraint: a difference in value that appears only at a vertical input will propagate to both outputs. The remaining downstream FUBs may experience errors on both inputs; because they are bijective the error will propagate to at least one of the outputs, but this case is not covered by VH-bijectivity and thus the error propagation cannot be predicted (represented by the light shading) without knowledge of the test pattern and error values. Based on this analysis, a useful property emerges for the FUB array: given a single defective FUB, the first-row and first-column errors observed along the output edges of the array closely approximate the location of the defective FUB. This property is key to the custom diagnosis described in this paper, and will be discussed in greater detail in the following sections.

III. Error Analysis

This section further describes the FUB array behavior relevant to the objectives of this paper. Specifically, Section III-A elaborates on how VH-bijectivity can be used to localize a single defective FUB in a FUB array. Section III-B examines how the inverse of a VH-bijective function can be used to improve defective FUB localization. Section III-C extends these concepts to arrays with multiple defective FUBs.

A. Forward Bound

As noted at the end of Section II, the location of the intersection of the errors observed on the array outputs for a failing pattern is closely related to the location(s) of defective FUBs in the array. Assuming there is only one defective FUB, Fig. 2 describes the three cases that can occur:

1) The defective FUB is horizontally adjacent to the error intersection (Fig. 2a).
2) The defective FUB is vertically adjacent to the error intersection (Fig. 2b).
3) The defective FUB is at the error intersection (Fig. 2c).

The array location derived from the errors observed at the array outputs is referred to as the *forward error intersection*. If only one defective FUB is present in the array, it must fall into one of the three cases observed in Fig. 2; these three implicated FUB locations are referred to as the *forward bound*. Calculation of the forward bound is thus a very simple method that can be applied to any failing pattern to localize defects down to three candidate FUBs in a FUB array.

B. Backward Bound

One property of bijective functions is that they are "reversible", that is, an inverse function is guaranteed to exist that maps the outputs of the bijective function to its inputs. This inverse function is also guaranteed to be bijective, and it can be easily proven that the inverse of a VH-bijective function is also VH-bijective. These properties allow for a failing test pattern and its response to be simulated *backwards* through the FUB array. Backward simulation of a test pattern uses the observed response as the array input and uses the inverse FUB function to calculate the internal (fault-free) FUB array values and the array output. The internal FUB array values produced by this backward simulation will be accurate until a defective FUB is reached, at which point differences will emerge between the backward simulation and the actual FUB array values. Because the inverse FUB function is VH-bijective, these differences will propagate all the way to the FUB array inputs. Thus, the differences between the FUB array input values obtained from the backward simulation and the applied test pattern can be used to generate a *backward bound* with the same properties as the forward bound.

This point is both subtle and powerful. Every failing test pattern and its observed response can be simulated both forwards and backwards, resulting in two bounds on the location of defective FUBs in the array. Fig. 3 is an example of how these bounds can be compared to localize a single defective FUB with even greater accuracy. In this example, the original test pattern and observed response can be used to construct the forward error intersection as shown in Fig. 3a. At this point the forward bound consists of three FUBs: the FUB at the forward error intersection (marked with the larger "○") and its *upstream* horizontal and vertical neighbors (marked with the smaller "○"). Performing the backward simulation results in the backward error intersection as shown in Fig. 3b. The backward bound also consists of three FUBs: the FUB at the backward error intersection (marked with the larger "□") and its *downstream* horizontal and vertical neighbors (marked with the smaller "□"). Intersecting these two bounds results in only two possible locations for the defective FUB. In this way, the

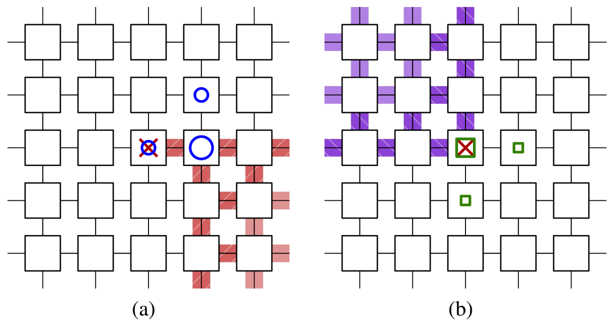


Fig. 3. (a) Forward and (b) backward bounds can be used together to localize a single defective FUB in a FUB array with greater accuracy. In this example only two FUBs (the defective FUB marked with the "×" and its horizontal neighbor) are implicated by both the forward and backward bounds.

backward bound improves the resolution from three to two candidate FUBs.

C. Multiple Defects

It is necessary to examine how multiple defects can affect the FUB array. There are two cases to consider: (i) multiple defects in a single FUB, and (ii) multiple defects in multiple FUBs. For the first case, the previous discussion on the forward and reverse bounds remain valid. For the second case, the bounds remain useful. For a failing test pattern for an array with N defective FUBs, 1 to N FUBs may be active, where each active defective FUB produces an erroneous FUB output due to defect activation. For these cases, the bounds can be used to differentiate between one active defective FUB and multiple active defective FUBs. Fig. 4 is an example of two different failing patterns for an array with two defective FUBs, each marked "×". In Fig. 4a, only one defective FUB is active, resulting in error intersections that are adjacent in the array. In contrast, Fig. 4b illustrates a test pattern with both defective FUBs active, resulting in bounds that correctly and accurately implicate two non-overlapping regions of the array, a result that can only occur when multiple defective FUBs are active.

It is possible however that the forward and backward bounds are incorrect due to error masking. Error masking in the presence of multiple defective FUBs can be mitigated by increasing the test set size, that is, a larger test set reduces the likelihood of multiple active FUBs that produce error masking. However, a larger test set has less impact when adjacent FUBs are defective. Section V explores these issues with a large fault injection experiment.

IV. Two-Level Custom Diagnosis

The FUB-array properties discussed in Section III can be exploited to create a custom, two-level diagnosis. Fig. 5 illustrates the two diagnosis levels. In the first level (Fig. 5a), diagnosis performed at the array level results in a list of defective FUBs with a characterization of each FUB's precise faulty behavior (e.g., a truth table for each defective FUB). In the second level (Fig. 5b), each defective FUB is diagnosed to derive the defect-level candidates. Both the array and FUB levels of this diagnosis procedure are described in greater detail in the remainder of this section.

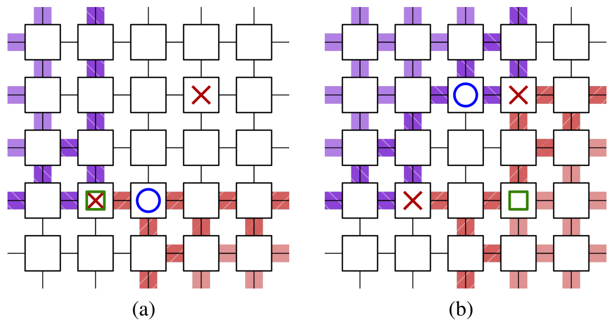


Fig. 4. Demonstration of the forward and backward error intersections (marked "○" and "□", respectively) of two different failing test patterns for an array with two defective FUBs (marked "×"). The two examples shown represent (a) a test pattern with only one defective FUB active, and (b) a test pattern with both defective FUBs active.

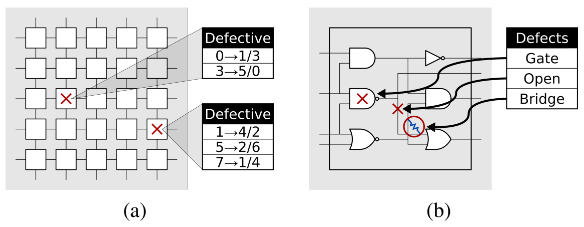


Fig. 5. Two-level diagnosis consists of (a) an array level where the array behavior is mapped to individual defective FUBs, followed by (b) a FUB level where each defective FUB behavior is analyzed to derive the defect candidates.

A. Array Diagnosis

Array level diagnosis relies on the forward and backward bounds described in Section III. Recall that the bounds are determined for a given test pattern by simulating a model of the FUB array (which can be either defect-free or a model that captures some or all characteristics of the defective array) in both the forward and backward directions, and then comparing the simulation results with the applied test pattern and observed response. The relationship between the two bounds is used to classify a test pattern into one of five categories:

1) **Match** - There is no difference between the behavior of the array model and the observed array response.
2) **Single Detect** - The intersection of the forward and backward bounds results in exactly one FUB. It is highly likely that unexplained behavior at the identified FUB is the source of the difference between the array model and the observed array response.
3) **Probable Single Detect** - The intersection of the forward and backward bounds results in exactly two FUBs. It is highly likely that unexplained behavior at one of those two FUBs is the source of the difference between the array model and the observed array response.
4) **Multiple Detect** - The intersection of the forward and backward bounds is empty, and the forward error intersection is upstream of the backward error intersection. The conclusion is that more than one FUB must be exhibiting unexplained behavior for this test pattern.
5) **Inconsistent** - Any other relationship between the forward and backward bounds is classified as inconsistent, indicating that the array model is incompatible with the observed array response. An example of this would be an empty bounds

```
array_diagnosis(T, R, M)
// T = List of tests
// R = List of array responses
// M = Defect-free array model
begin
  final_candidates = NULL // list of final diagnosis candidates
  n = 1 // Number of simultaneous defective FUBs
  for each (t, r) in T, R
    init_class(t) = classify_pattern(t, r, M)
  while(final_candidates is empty) // outer diagnosis loop
    S = all sets of n FUBs implicated by pattern classification
    for each set of suspect FUBs s in S
      DM = copy of defect-free array model M
      mark FUBs in s as defective in DM
      current_class = copy of init_class
      while(True) // inner diagnosis loop
        // Update array model with observed defective behavior
        if("Single Detect" in current_class)
          select t where current_class(t) == "Single Detect"
          update DM with behavior observed in t
        else if("Probable Single Detect" in current_class)
          select t where current_class(t) == "Probable Single Detect")
          update DM with behavior observed in t
        else if("Multiple Detect" in current_class)
          select t where current_class(t) == "Multiple Detect")
          update DM with behavior observed in t
        endif
        for each (t, r) in T, R
          updated_class(t) = classify_pattern(t, r, DM)
        if(updated_class is all "Match")
          add DM to final_candidates
          break
        else if(compare(current_class, updated_class) indicates error)
          break // give up on current defective array model
        endif
        current_class = updated_class
      end while
    // Try again with more defective locations if no final candidates
    n = n+1
  end while
  return final_candidates
end
```

Fig. 6. Pseudocode for custom array diagnosis.

intersection with the forward error intersection downstream of the backward error intersection.

Pseudocode for the full array-level diagnosis is presented in Fig. 6. The first step is to classify all of the test patterns using a defect-free model of the FUB array. These classifications are used to identify likely locations of defective FUBs in the array. The inner diagnosis loop of the algorithm begins by modifying an array model with defect(s), *DM*, based on the behavior observed in one of the failing test patterns. The modified model is then used to reclassify all of the test patterns. If the defective array model perfectly matches the observed behavior, then it is added to the final list of diagnosis candidates. Otherwise, the changes in test-pattern classification due to the modified model are analyzed; test patterns that change from "Single Detect" to "Match", for example, indicate the defective array model is accurately reflecting the observed behavior. However, other changes in classification, for example from "Match" to "Multiple Detect", indicate that an incorrect modification has been made to the array model; if any such change occurs, the array model is discarded. Initially, the diagnosis procedure tries to find exactly one defective FUB that can replicate all of the observed behavior of the failing FUB array; if this fails, the array model is modified to include larger numbers of defective FUBs until either diagnosis is successful or some other stopping criterion (not shown in pseudocode) is reached.

B. *FUB Diagnosis*

The result of the array-level diagnosis described in Section IV-A is a list of (potentially) defective FUB locations and their corresponding defective behavior in the form of a truth table for each FUB. FUB-level diagnosis maps these defective FUB behaviors to the same defect-level candidates that state-of-the-art diagnosis tools produce. Two general techniques can be used to accomplishing this mapping:

- **Cause-Effect** - The most straightforward approach for FUB-level diagnosis is to create a fault dictionary for each FUB in the array. The defective FUB behaviors produced by array-level diagnosis can then be referenced against the individual FUB dictionaries to derive the final defect candidates. The traditional downsides to fault dictionaries, namely their storage overhead and computational cost, are mitigated by the small size of the individual FUBs and the minimal number of tests for each FUB.
- **Effect-Cause** - The second approach is to use an effect-cause analysis on each implicated FUB. The defective FUB behaviors produced by array-level diagnosis can be interpreted as input and output values for each individual FUB netlist, which can then be processed by a diagnosis procedure to determine the final defect candidates. The small size of the individual FUBs is again useful in that it allows for the employment of more sophisticated approaches without significant runtime overhead, resulting in improved diagnosis outcomes.

V. Experiment

This section describes the implementation and evaluation of the multiple-defect diagnosis procedure presented in this paper. Specifically, Section V-A covers the implementation details of the custom diagnosis. Section V-B describes the setup for the simulated fault injection and diagnosis experiment employed to evaluate the proposed custom diagnosis. Finally, Section V-C presents the results of the experiment.

A. *Implementation*

A program written using the Python language implements the array-level diagnosis. Fault dictionaries based on the input pattern (IP) fault model [8] are used to handle FUB-level diagnosis. The IP fault model is a functional fault model that allows one or more rows of the truth table of a circuit module (e.g., a standard cell, a FUB, etc.) to change. IP faults can be divided into either single or multiple; a single IP fault is defined as a change in the output(s) of exactly one truth-table row. For example, a two-input NAND that produces a 0 instead of a 1 only when an input pattern of 00 is applied is modeled by a single IP fault. Conversely, a multiple IP fault changes the output(s) of more than one truth-table row. The fault dictionaries used for FUB-level diagnosis are created for all single and multiple IP faults applied to the standard cells within each FUB in the array.

B. *Experiment Setup*

To verify the advantages of custom diagnosis, a simulated fault-injection and diagnosis experiment is performed. This experiment utilizes a CM-LCV design created for verifying a commercial 7nm standard-cell library. The LCV contains 8,308 standard cells, organized into 144 individual FUBs that are arranged into a 12×12 FUB array, where each FUB implements the same 6-bit VH-bijective function[1]. A test set

[1]Ideally the size of the FUB array should be such that the expected number of defective FUBs does not exceed the diagnosis capabilities. Given a defectivity (per unit area, per standard cell, etc.), standard techniques can be used to determine an appropriate FUB array size.

consisting of 512 test patterns with 100% fault efficiency for multiple models (stuck-at, standard-cell-level IP, FUB-level IP, and 3-detect [9]) is constructed for this FUB array. Two faults are simultaneously injected into this array and simulated to create a virtual fail log for diagnosis. Injected faults are randomly-selected single IP faults applied to the standard cells in the design. A total of 5,397 virtual fail logs are generated and diagnosed using custom and commercial diagnosis. Two filters are applied to the commercial diagnosis results: first, bridge-type defect candidates are removed, leaving only open, stuck-at, and standard-cell defect candidates. This filtering ensures that the commercial diagnosis tool is effectively diagnosing to the same types of faults used to create the virtual fail logs. Second, only the top-scoring candidates are kept for each suspected defect in order to improve the diagnostic resolution without significantly compromising accuracy.

C. Results

Before presenting the results of the simulated fault-injection experiment, there are two points worth mentioning. First, it is helpful to examine what an ideal diagnosis result would be in this context. Diagnosis is evaluated based on two criteria: *resolution*, which is defined as the number of defect candidates reported, and *accuracy*, which is defined as whether those defect candidates subsume the actual defect(s). Because two faults are injected into the array for each virtual fail log, an ideal diagnosis result should have exactly two defect candidates, with each defect candidate corresponding to one of the injected faults. A resolution lower than two cannot be perfectly accurate for both injected faults; a resolution greater than two indicates a loss of precision.

Second, it should be noted that the location of the injected faults can have an effect on the custom diagnosis. Two cases in particular deserve special consideration: when both faults are injected into the same FUB, and when the two faults are injected into adjacent FUBs. Of the 5,396 fault injections performed, 145 fell into one of these two cases. These fault injections have been filtered out and will be addressed at the end of this section. This leaves a total of 5,251 virtual fail logs where the injected faults are in non-adjacent FUBs.

Fig. 7 is a histogram of the diagnostic resolution for the commercial and custom diagnosis procedures. Note that the vertical axis is logarithmic to better represent the range of the two distributions. Commercial diagnosis has a minimum diagnostic resolution of 1, a maximum of 43, and an average of 3.50. Custom diagnosis has a minimum diagnostic resolution of 2, a maximum of 19, and an average of 2.85. It is clear that, on average, the custom diagnosis improves upon the diagnostic resolution achieved by commercial diagnosis. Note that this result in no way diminishes the value of commercial diagnosis. In particular, the custom diagnosis is developed to leverage the special properties of the FUB array. While this specialization results in improved diagnosis outcomes, it is also limited to the FUB array. The commercial diagnosis, on the other hand, is capable of handling a wide variety of designs.

Table I is a comparison of the diagnostic accuracy. Given that the faults are injected into individual standard cells in the array, a defect candidate is defined to be accurate if it includes (i) the standard cell used for injection, or (ii) a net that is connected to the standard cell. Diagnostic accuracy can thus be either 100% (defect candidates include the two injected sites), 50% (defect candidates include one of the two injected sites), or 0% (defect candidates include neither injected site) for each fault-injection simulation. Table I indicates that custom diagnosis achieves ideal accuracy (i.e., 100% for all 5,251 fail logs), a nearly 30% improvement over state-of-the-art commercial diagnosis.

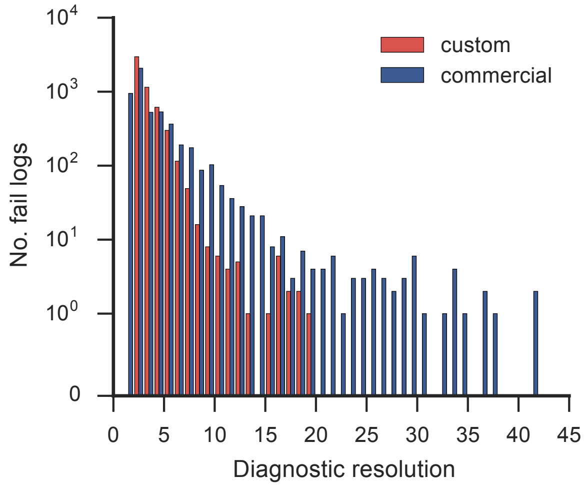


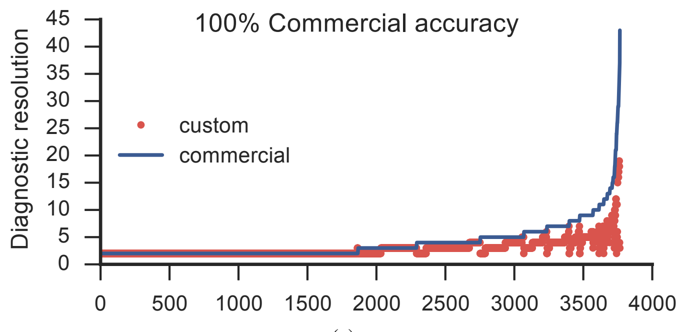
Fig. 7. Diagnostic resolution for both commercial and custom diagnosis (note that the vertical axis is logarithmic).

TABLE I
ACCURACY COMPARISON FOR COMMERCIAL AND CUSTOM DIAGNOSIS

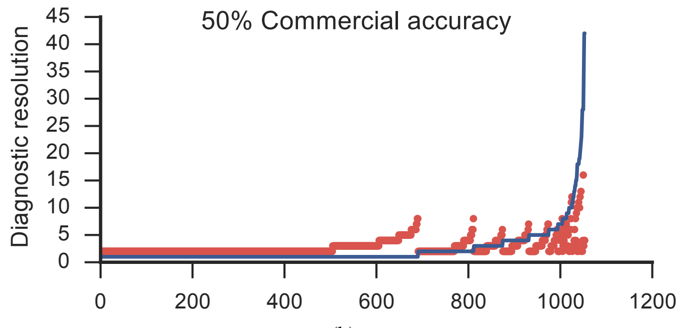
	Custom			Commercial		
Accuracy	Count	Fraction	Avg. res.	Count	Fraction	Avg. res.
100%	5,251	1.000	2.85	3,766	0.717	3.97
50%	0	0.000	NA	1054	0.201	2.38
0%	0	0.000	NA	431	0.082	2.04

The comparisons presented thus far are summaries over all fail logs. Fig. 8 compares the diagnostic outcomes for each fail log individually. Each plot in Fig. 8 sorts the fail logs from the minimum to the maximum commercial diagnosis resolution. In particular, Fig. 8a only contains fail logs where the commercial diagnosis is 100% accurate, while Figures 8b and 8c correspond to 50% and 0% commercial diagnosis accuracy, respectively. Fig. 8a clearly shows that for every fail log but one (99.97%) where the commercial accuracy is 100%, custom diagnosis either matches or improves the diagnostic resolution without compromising the 100% accuracy. Figures 8b and 8c show that, while the commercial diagnosis can improve upon the custom diagnosis resolution for some cases, it can only do so by compromising diagnosis accuracy.

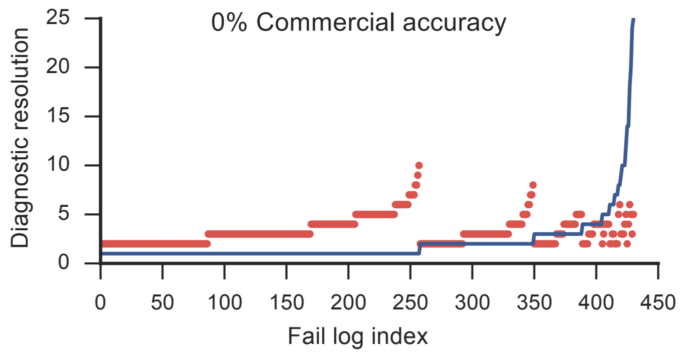
Finally, the 145 filtered fail logs are examined. Table II summarizes the commercial and custom diagnosis results for the 26 fail logs generated with both injected faults in the same FUB. The custom diagnosis is disadvantaged in this case because the fault dictionaries used for FUB-level diagnosis



(a)



(b)



(c)

Fig. 8. Diagnostic resolution comparison for the custom and commercial diagnosis for each virtual fail log. The virtual fail logs have been separated into three populations based on commercial diagnosis accuracy: (a) fail logs with 100% commercial accuracy, (b) fail logs with 50% commercial accuracy, and (c) fail logs with 0% commercial accuracy.

TABLE II
ACCURACY COMPARISON FOR COMMERCIAL AND CUSTOM DIAGNOSIS FOR TWO FAULT INJECTIONS INTO A SINGLE FUB

	Custom			Commercial		
Accuracy	Count	Fraction	Avg. res.	Count	Fraction	Avg. res.
100%	0	0.000	NA	2	0.077	3.50
50%	1	0.038	1	3	0.115	1.33
0%	25	0.962	0	21	0.808	1.14

assume a single faulty standard cell. The array-level diagnosis, on the other hand, handles single defective FUBs without difficulty regardless of how many faulty standard cells it contains. Either expanding the fault dictionaries or using an effect-cause approach for FUB-level diagnosis would significantly improve custom diagnosis performance for these cases.

Table III summarizes the commercial and custom diagnosis results for the remaining 119 fail logs involving faults injected into adjacent FUBs. These cases are more difficult for custom diagnosis due to the high likelihood of significant defect interaction, including, in the worst case, full-error masking. However, custom diagnosis still delivers perfect accuracy with good resolution for 38.7% of these fail logs. Furthermore, when custom diagnosis fails to report any candidates the commercial diagnosis can still be used as a fall-back method.

TABLE III
ACCURACY COMPARISON FOR COMMERCIAL AND CUSTOM DIAGNOSIS FOR ADJACENT-FUB FAULT INJECTIONS

	Custom			Commercial		
Accuracy	Count	Fraction	Avg. res.	Count	Fraction	Avg. res.
100%	46	0.387	2.80	49	0.412	3.61
50%	1	0.008	1.00	39	0.328	2.05
0%	72	0.605	0.00	31	0.261	1.42

VI. Conclusions

This work proposes a custom, two-level diagnosis procedure for multiple defects that takes advantage of the unique properties of the FUB array used in the CM-LCV. A fault injection experiment with two simultaneous faults is used to compare custom and commercial multi-defect diagnosis performance. Results indicate that custom diagnosis achieves perfect accuracy for 98.2% of the fail logs and diagnostic resolution of five or less for 94.1% of the fail logs, improving upon the commercial diagnosis results that are 70.7% perfectly accurate, and 85.0% with resolution of five or less. Future work will focus on improvements to the diagnosis procedure that allow it to better handle difficult cases, particularly adjacent defective FUBs. Finally, custom diagnosis will soon be applied to real silicon data from state-of-the-art fabrication facilities produced in ongoing collaboration with industry partners.

References

[1] R. D. Blanton, B. Niewenhuis, and C. Taylor, "Logic characterization vehicle design for maximal information extraction for yield learning," *International Test Conference*, pp. 1–10, Oct 2014.

[2] B. Niewenhuis and R. D. Blanton, "Efficient built-in self test of regular logic characterization vehicles," *VLSI Test Symposium*, pp. 1–6, April 2015.

[3] Z. Liu, B. Niewenhuis, S. Mittal, and R. D. Blanton, "Achieving 100% cell-aware coverage by design," *Design, Automation Test in Europe Conference*, pp. 109–114, March 2016.

[4] S. Mittal, Z. Liu, B. Niewenhuis, and R. D. Blanton, "Test chip design for optimal cell-aware diagnosis," *International Test Conference*, Nov 2016.

[5] R. D. Blanton, B. Niewenhuis, and Z. D. Liu, "Design reflection for optimal test-chip implementation," *International Test Conference*, pp. 1–10, Oct 2015.

[6] P. Fynan, Z. Liu, B. Niewenhuis, S. Mittal, M. Strojwas, and R. D. Blanton, "Logic characterization vehicle design reflection via layout rewiring," *International Test Conference*, Nov 2016.

[7] A. D. Friedman, "Easily testable iterative systems," *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1061–1064, Dec 1973.

[8] R. D. Blanton and J. P. Hayes, "Properties of the input pattern fault model," *International Conference on Computer Design*, pp. 372–380, Oct 1997.

[9] S. C. Ma, P. Franco, and E. J. McCluskey, "An experimental chip to evaluate test techniques experiment results," *International Test Conference*, pp. 663–672, Oct 1995.