Automatic Classification of Bridge Defects

Jeffrey E. Nelson, Wing Chiu Tam, and R. D. (Shawn) Blanton
Center for Silicon System Implementation
Carnegie Mellon University, Pittsburgh, PA 15213
http://www.ece.cmu.edu/~actl
{jnelson, wtam, blanton}@ece.cmu.edu

Abstract
A technique is proposed to automatically predict whether a failing chip has a bridge defect. Logic diagnosis is performed using scan test results to identify candidate nets. Several relevant features of the test data are measured for net pairs that consist of the diagnosis candidates and other nets in close physical proximity. Based on these features, rules are constructed to identify defects that fully exhibit classic bridge behaviors, while the remaining chips are classified using a forest of decision trees. Results indicate that a population of chips failing due to bridges can indeed be extracted with very high accuracy. Finally, the method correctly classifies 41 commercially-fabricated chips that underwent PFA.

1 Introduction
Commercial semiconductor manufacturing is becoming increasingly complex as process engineers push themselves to keep up with the self-imposed goal of a new process node every two years. Transistors not only continue to scale to unfathomably small dimensions, but to compensate for the resulting physical effects, new materials and construction techniques are used. Smaller dimensions demand a new level of computational lithography to build masks capable of pushing the limits of 193nm wavelength light. All of this increasing complexity in manufacturing means it is becoming harder and harder to yield working chips.

Any investment that advances processing techniques must be supported by an investment in yield learning technologies. Inline inspection tools that optically examine wafers between key processing steps present a potential bottleneck as the number of steps increases. As on-chip dimensions decrease, so too does the effectiveness of optical inspection techniques, leading to reports of a glut of potential defects that ultimately have no effect on yield. Discrete test structures provide excellent resolution and accuracy of defects, but at the precious cost of silicon area, forcing a significant tradeoff between the ability to learn about defects and satisfy product-volume requirements. Late in the process lifecycle, discrete test structures become relegated to the scribe lines, significantly decreasing the number of defects they observe.

The natural evolution of yield learning techniques is to use the product itself as a characterization vehicle in conjunction with failing test results generated by defective chips. The International Technology Roadmap for Semiconductors describes test for yield learning as a high-priority, difficult challenge that is “critically essential for fab process and device learning below optical device dimensions [1].” Early in the process lifecycle, SRAMs will no doubt continue to be the learning vehicle of choice, but even at this stage where baseline process parameters are established, new, advanced techniques to leverage test results can provide valuable insight into both systematic and random yield detractors.

Using the product as a yield learning vehicle has three significant advantages over other approaches. First, it covers the entire wafer, maximizing defect observability. Second, working products can be sold for profit, whereas non-product characterization vehicles are not themselves profitable if they yield, but can later lead to yield improvements for customer products. Finally, the product itself provides the best diversity of layout geometries and the exact conditions that must be manufacturable. While the extreme regularity of SRAMs is convenient for early process debugging, ultimately the entire product (i.e., ASIC, microprocessor, GPU, etc.) is what needs to yield.

In this paper, a method is proposed to aid product-test-based yield learning by providing an automatic way to determine the defect type that leads to chip failure. In particular, this work emphasizes the automatic classification of chips as either failing due to a bridge defect, or failing due to some non-bridge defect. This is a key enabling technology for related work that aims to measure defect density and size distributions (DDSDs) of bridge-causing defects using only product test results [2, 3]. The approach is designed to be time-efficient to allow a significant number of defective chips to be analyzed to satisfy sample size requirements of any subsequent analyses (such as DDSD measurement). It should be noted that it is not necessary that every bridge defect be identified, nor is it necessary that there be zero misclassification. Some level of imperfection is tolerable in many applications (e.g., DDSD measurement has been shown to tolerate a mis-classification rate of 10% [4]).

Having improved methods to automatically group failing chips by defect type could open up new opportuni-

*This work supported in part by an IBM/SRC fellowship.
ties for automatic Pareto generation as well as drive decisions about which chips should undergo costly and time-consuming physical failure analysis (PFA).

In the next section, background is given on both test-based yield learning and prior art in bridge diagnosis. The details of the automatic defect classification method are given in Section 3. The method is applied to SPICE-generated test responses for defects injected into a benchmark circuit in Section 4. In Section 5, 41 failing industrial chips that have undergone PFA are automatically classified using the models built in Section 4. In Section 6, some general observations about the method are made and worthwhile directions for future work are identified. Finally, the method, and the various experiment results are summarized in Section 7.

2 Background

Several product-test-based yield learning techniques are either currently in use today, or have been proposed. IBM has published an overview of how their “Grand Pareto” is generated, which is heavily dependent on test results and logic diagnosis [5]. Even in that work, however, diagnosis is limited to isolating the location of the defect, and is not used to provide any concrete insight into the defect mechanism. Ultimately, PFA has the final say in determining the defect mechanism.

Another test-based yield learning technique was proposed by Mentor Graphics and LSI Logic. Failure rates of specific features in the layout—such as parallel wires and wire corners diagonally positioned from each other—are measured using logic test results and logic diagnosis [6].

A diagnosis technique that uses single location at a time (SLAT) [7] is described in [8] for generating signatures for each defect suspect of varying types (stuck, open, dominant bridge, wired-OR, wired-AND, 3-way bridge, and unknown) that are then ranked according to how closely they match observed failures. Test results for 200 failing cycles are used. No method is described to determine the defect type when several different models are ranked very high. However, in contrast, this work scores various fault models and considers them along with additional parameters to automatically determine defect type. Lavo et al. also present a novel diagnosis technique, but with an emphasis on localization and not quantifiably accurate classification [9].

Desineni et al. present the idea that any defect that leads to scan-test failures can have a logic model that exactly explains how each failure is activated [10]. Using fail data and logic synthesis, a model of the fault behavior that explains the conditions necessary to activate the fault is created. While this is a novel approach to logic diagnosis and provides insight into the behavior of the fault, the new logic model must still be manually interpreted in order to conclude, if possible, what type of defect caused the failure.

Several researchers have proposed bridge diagnosis techniques based on full or partial SSL dictionaries, or dynamically evaluated stuck behaviors [11, 12]. Others have proposed performing bridge diagnosis using $I_{DDQ}$ measurements [13–16]. In each case, the focus of the work is on efficiently using results from structural testing or $I_{DDQ}$ patterns to localize the bridge defect. The emphasis on localization is common since ultimately, the chip will undergo PFA and the precise defect location and mechanism will (possibly) be revealed. However, localization on its own is insufficient if the goal is to perform automated analyses on a large volume of chips failing due to bridges.

A technique to measure defect density and size distributions in high-volume manufacturing using logic test results and logic diagnosis has been proposed [2, 3]. One of the key enablers of that technology is the ability to first identify a high volume of chips that fail due to a specific defect type. The method in [2, 3] focuses on bridge defects, and is a significant motivator of the work presented in this paper.

3 Identifying Chips with Bridge Defects

Although many logic diagnosis techniques exist, they traditionally focus on localizing a defect. The emphasis in this work is on determining, with some degree of confidence, whether the defect causing a chip to fail is a bridge or some other defect type. This work was motivated by a previously presented strategy to measure DDSDs of a process using product test results [2, 3]. In that work, bridge defects in a manufacturing process are characterized by analyzing a set of chips that fail due to bridges. Thus, as a prerequisite, all chips that are analyzed must have a bridge defect. There currently exists no way to identify with confidence a set of failing chips that fail due to bridge defects without using PFA. The work presented in this paper addresses that problem.

It should be noted that this problem is harder than it may first appear. Many defects may exhibit behavior that can appear to stem from a bridge defect. However, the requirement in this work is much more stringent in that it is not sufficient to identify defects that exhibit misbehavior that could be produced by a bridge defect. Instead, the aim is to identify actual bridge defects.

The remainder of this section (Section 3.1) presents the overall flow of the bridge-identification method, while Sections 3.2 and 3.3 detail the various features that must be measured from the logic model and test results to support defect classification, respectively.

3.1 Bridge Classification Flow

Figure 2 shows the overall flow of the bridge classification method. Logic diagnosis is performed on scan-test results to identify candidate nets that could potentially be affected by a defect within a failing chip. Candidate nets resulting from diagnosis serve as the starting point for this work. This is a much better choice than initiating an analysis that uses all nets from the entire design. The bridge classification method presented in this paper allows for many different types of diagnosis at this stage. That is, a precise diagnosis technique can be employed to narrow the set of initial candidates to a very small number, or a more conservative diagnosis approach can be used that allows a large
The ability of both the bridge rules and the decision forest to classify chips as failing due to a bridge or some other defect type depends significantly on having a set of features that reveal details about the logical behavior of the defect. A description of each feature is provided in the following sub-sections.

### 3.2 Logic Features

Each of the features described in this section can be measured for a candidate net pair \{A, B\} by examining only the logic model. While the selection of the net pair is dependent on failing test data, the features are not. (Each feature is highlighted in bold.)

**Feedback** is a boolean feature that indicates whether or not there is a structural path from A to B or from B to A. If such a path exists, it means that a bridge between the net pair will create feedback. Existence of a feedback path has a significant impact on the testability of a bridge defect. First, it is possible for the bridge fault to be activated for test patterns where A and B are driven to the same logic value. This counter-intuitive behavior can occur when the feedback path creates a latching effect that captures a faulty value. Second, a feedback path can cause oscillation leading to unpredictable and varying test results. Because of these erratic behaviors, candidate net pairs that have a feedback path between them are discarded because even if they are classified as a bridge, confidence in that classification is low.

**Drive-parity-gate** is a boolean feature that is true if A and B are inputs to a parity gate (XOR or XNOR). The truth tables in Table 1 show how a wired-AND and wired-OR bridge fault on the gate inputs are both equivalent to an output stuck-at-0 (stuck-at-1) fault on an XOR (XNOR) gate. Due to this equivalence, any stuck fault on the output of a parity gate will look like a bridge fault on the inputs, and vice versa, thus making it difficult to have any confidence in the decision to classify such a net pair as having a bridge defect. It should be noted that this feature is even measured for a group of gates that implement a parity function.

**Logical-correlation** is true if for every test pattern the good simulation value of A is either always the same or always the opposite of B. If both A and B are inputs to a parity gate, this is true if and only if they are non-bridge defects. For a group of gates that implement a parity function, the good simulation value of A is always the same if and only if both A and B are inputs to a parity gate.

**Logical-correlation** is true if for every test pattern the good simulation value of A is either always the same or always the opposite of B. If both A and B are inputs to a parity gate, this is true if and only if they are non-bridge defects. For a group of gates that implement a parity function, the good simulation value of A is always the same if and only if both A and B are inputs to a parity gate.

Table 1. Truth tables in the presence of an output stuck-at fault, wired-AND, and wired-OR faults that affect the gate inputs A and B for (a) an XOR gate, and (b) an XNOR gate.

<table>
<thead>
<tr>
<th>(AB)</th>
<th>(Y)</th>
<th>(Y_{WAND})</th>
<th>(Y_{WOR})</th>
<th>(Y_{sa-0})</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>(AB)</th>
<th>(Y)</th>
<th>(Y_{WAND})</th>
<th>(Y_{WOR})</th>
<th>(Y_{sa-1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b)
from the fact that non-feedback bridges are only activated when $A$ and $B$ have opposite logic values. The ideal test set will have tests that drive $A$ and $B$ to the same logic value for some sensitizing patterns and to opposite logic values for other sensitizing patterns. If $A$ and $B$ are always the same logic value, the bridge will never be activated, and thus cannot be identified as a bridge. If $A$ and $B$ are always opposite logic values, it cannot be determined if the chip would fail when $A$ and $B$ are driven to the same logic value. Failing patterns that exhibit these characteristics would decrease confidence that there is actually a bridge, and aid chip classification.

**Drive-same-gate** is a boolean feature indicating whether two nets $A$ and $B$ are both inputs to the same logic gate. The motivation behind this feature is illustrated in Table 2. The truth tables show the effect of (a) a wired-OR fault affecting the inputs of an AND gate, and (b) a wired-AND fault affecting the inputs of an OR gate. It can be seen that each input bridge fault creates a gate-mutation fault, where the logic function of the AND gate becomes an OR and the logic function of the OR gate becomes an AND. Because the fault effect of each bridge is visible only at the gate output, a bridge defect between $A$ and $B$ when $A$ and $B$ drive the same gate is not distinguishable from a cell defect within the gate. This relationship is identified so it can be accounted for when the chip is classified.

### 3.3 Test-Dependent Features

The features described in this section are derived from the test results for a failing chip. Chips with catastrophic or scan-chain failures are discarded and not considered. Scan-test failure results are examined and features are measured for each candidate net pair $\{A, B\}$. Each feature is highlighted in bold.

**Wired-AND-score** indicates how closely the failing tester response matches the simulated failing response for a wired-AND fault between $A$ and $B$. Each test-response bit (the logic value on a single scan cell or primary output for a single test pattern) is compared to the bit resulting from fault simulation. For each bit, the parameters below show the four possible outcomes that can result from each comparison:

- $tfsf$: tester failed, simulator failed
- $tfsp$: tester failed, simulator passed
- $tpsf$: tester passed, simulator failed
- $tpsp$: tester passed, simulator passed

The match score $\in [0, 1]$ is calculated using the very-common formula [18, 19]:

$$\text{match score} = \frac{tfsf}{tfsf + tfsp + tpsf}$$  \hspace{2cm} (1)

Note that the parameter $tpsp$ is not used to calculate the match score because it provides no information about the simulated defect nor the tested defect. A match score of 1.0 indicates that the chip behaves identically to a wired-AND fault between $A$ and $B$ for all patterns. A value of 0.0 is only possible if there is no overlap among the failing tester and simulated response bits. **Wired-OR-score** is calculated in a similar way, except the wired-OR fault model is used. **Wired-score** is defined as the maximum of the wired-AND-score and the wired-OR-score.

For a single test pattern, if the wired-AND simulation response for a net pair $\{A, B\}$ perfectly matches the tester response, then one of the nets has a logical 0, and the other has an error. The feature **wired-AND-Adom** tracks how many times $A$ dominates $B$. For example, if $A$ has a logical 0, and $B$ is driven to a 1 but tested as a 0, $A$ dominated $B$ for.
this pattern, and wired-AND-Adom is incremented. Likewise, when $B$ dominates $A$, wired-AND-Bdom is incremented. Wired-OR-Adom and wired-OR-Bdom are calculated similarly, except using the wired-OR fault model. When all bits from all test responses have been compared to the simulated test response bits, these features give an indication of whether only one net is failing, or if multiple nets fail.

**Wired-AND**-spread is not directly measured from the test data, but rather it is constructed from the previous features using the formula:

$$\text{Wired-AND-spread} = AB + BA \quad AB \quad BA$$

where $AB$ is wired-AND-Adom, and $BA$ is wired-AND-Bdom. Thus, wired-AND-spread is zero when either $AB$ or $BA$ is zero, and assigned its maximal value $(AB + BA)$ when $AB$ equals $BA$. The intuition behind this formula is that a bridge defect that causes only one of the two nets to ever fail provides little evidence that there is in fact a bridge defect. This corresponds to the case where either $AB$ or $BA$ is zero. Alternatively, one test pattern may cause net $A$ to fail, and another pattern may cause net $B$ to fail. This corresponds to the case where both $AB$ and $BA$ are greater than zero, and thus wired-AND-spread is also greater than zero. A non-zero value for wired-AND-spread indicates that both nets $A$ and $B$ fail during testing. A larger value of wired-AND-spread provides greater confidence that there is a defect affecting multiple nets, vastly increasing the likelihood that the defect is a bridge. Wired-OR-spread is calculated the same way as wired-AND-spread, except the wired-OR fault model is used. Wired-bridge-spread is the maximum of wired-AND-spread and wired-OR-spread. Finally, positive-wired-bridge-spread is a boolean feature that is true only if wired-bridge-spread is greater than zero. It is beneficial to use a boolean property because the range of wired-bridge-spread values will vary from circuit to circuit, but the most significant difference observed in our analyses is between a wired-bridge-spread of zero and a wired-bridge-spread greater than zero.

**Adom-score** is calculated in a similar fashion to wired-AND-score. The simulated response for the bridge fault that has $A$ dominating $B$ is compared to the tester response and the match score is calculated using Equation (1). Bdom-score is calculated the same way, except the bridge fault where $B$ dominates $A$ is used. Dom-score is the maximum of Adom-score and Bdom-score.

**4-way-score** is the match score for the bridge fault where for any test pattern, either $A$ can dominate $B$ or $B$ can dominate $A$. This is also known as the 4-way bridge fault model [20]. Similar to wired-bridge-spread, the 4-way-spread is calculated as:

$$\text{4-way-spread} = AB + BA \quad AB \quad BA$$

where $AB$ is the number of patterns where $A$ dominates $B$ and $BA$ is the number of patterns where $B$ dominates $A$. Likewise, positive-4-way-spread is a boolean that is true only if the 4-way-spread is greater than zero.

The simulated response for the fault $A$ stuck-at-$0$ is compared to the tester response to calculate the match score for $A$-stuck-at-$0$, $A$-stuck-at-$1$, $B$-stuck-at-$0$, and $B$-stuck-at-$1$ are calculated similarly.

A-stuck-at-$0$-or-$1$ is the match score for either stuck-fault affecting $A$ that can be activated (obviously) by either a logical zero or one. In other words, every test pattern activates either $A$ stuck-at-$0$ or $A$ stuck-at-$1$. B stuck-at-$0$-or-$1$ is calculated similarly. This is a more flexible version of the stuck-at fault model that allows a net to fail on any pattern, but may not necessarily be sensitized to an observable point. This model can capture an open defect that can cause a net to not only fail when driven to a logical zero, but also when driven to a logical one.

**Same-Polarity-Fails** is a boolean that is true if there is at least one test pattern where $A$ and $B$ are driven to the same logic value, yet the test pattern still failed. If this feature is true, it is highly indicative of a non-bridge since having opposite polarity values on $A$ and $B$ is a prerequisite for non-feedback bridge activation.

### 4 Classification

In the previous section, many features of the logic model and test measurement data were discussed. This section describes the two steps of classification based on those features. The first step is the application of rules based on knowledge of the test and manufacturing domain. The second step uses a data-trained decision forest which is described in detail in Section 4.2.

The simulation data used in this work is generated using a system that automatically creates chip-failure populations [21]. Extensive details of how the populations are generated can be found in [21], but are briefly summarized here.

Simulated defects fall into one of five categories: signal bridge (metal short modeled in SPICE as a 10Ω resistor), open (metal void), resistive via, cell defect, and supply bridge (to net). Even though the defect mechanism could be the same, supply bridges are considered a separate category from net-to-net bridges because the supply will likely always dominate the signal, making the logical behavior appear as a true stuck-at-$0$ or stuck-at-$1$ fault. It is therefore impossible to identify a supply bridge as a bridge defect with confidence, thus they are not considered bridges for the purposes of this work. The breakdown of the number of defects generated in each category is shown in Table 3.

The column marked “Diagnosed as Bridge” indicates the number of chips whose SPICE-generated test response exactly matches the simulated test response of one of the following bridge fault models: wired-AND, wired-OR, or dominating. Only 544 SPICE-simulated defects are actually bridges out of 1,642 SPICE-simulated defects diagnosed as a bridge. It is clear that a chip with a non-bridge defect can lead to a tester response that exactly matches the simulated response of a bridge fault. This emphasizes the need for a more rigorous approach to defect classification—one that does not rely exclusively on how closely the tester response of a chip matches the response of a simulated bridge fault.
### Table 3.

<table>
<thead>
<tr>
<th>Defect type</th>
<th>Count</th>
<th>Diagnosed as bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal bridge</td>
<td>938</td>
<td>544</td>
</tr>
<tr>
<td>Open</td>
<td>3,368</td>
<td>659</td>
</tr>
<tr>
<td>Resistive via</td>
<td>1,638</td>
<td>332</td>
</tr>
<tr>
<td>Cell</td>
<td>441</td>
<td>50</td>
</tr>
<tr>
<td>Supply bridge</td>
<td>297</td>
<td>57</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>6,682</strong></td>
<td><strong>1,642</strong></td>
</tr>
</tbody>
</table>

Breakdown of type, count, and number of SPICE-simulated defects whose test response perfectly matches a bridge fault model.

### Table 4.

<table>
<thead>
<tr>
<th>Defect</th>
<th>Net pairs</th>
<th>Feedback</th>
<th>Non-feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridge</td>
<td>938</td>
<td>557</td>
<td>381</td>
</tr>
<tr>
<td>Non-bridge</td>
<td>244,434</td>
<td>137,995</td>
<td>106,439</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>245,372</strong></td>
<td><strong>138,552</strong></td>
<td><strong>106,820</strong></td>
</tr>
</tbody>
</table>

Candidate net pairs by defect class, number of feedback pairs, and non-feedback pairs.

In this work, the circuit is a synthesized version of the ISCAS benchmark circuit c1355 [22] using 180nm process technology. The defects described in Table 3 represent most of the possible defects that can occur in the benchmark circuit. For each defect selected from the defect universe, the defect is injected into the layout, parasitics are extracted, and the new circuit model is SPICE-simulated using a 100% stuck-at coverage test set. It should be noted that it is not strictly necessary to SPICE-simulate the entire circuit. To decrease the required simulation time, it is possible to only SPICE-simulate small portions of the circuit that are affected by a defect, then use that defective response to determine the overall test response.

The SPICE-simulated test responses in the form of analog waveforms are converted to binary logic values for further processing—as if they were generated from automatic test equipment. As described in Section 3.1, logic diagnosis is performed using these resulting “test responses,” and candidate nets are identified from the diagnosis output. Candidate net pairs are formed by pairing each candidate net with each of its physical neighbors in the layout. Only two-net bridges are considered here because in order to bridge three or more nets, a much larger defect is required. As defect sizes increase, the probability of such a defect decreases exponentially. Thus, a large majority of anticipated bridges are covered by examining only net pairs. It is, however, possible to generalize the entire methodology for bridges involving more than two nets.

Table 4 shows the total number of candidate net pairs generated from bridge defects and non-bridge defects. The second column of Table 4 shows the total number of net pairs, including those that create feedback if a bridge actually existed between the two nets. The third column labeled “Feedback” indicates the number of nodes that have feedback, and the “Non-feedback” column shows the final count of net pairs after the ones that cause feedback have been removed. One reason the number of net pairs for bridges is much smaller than the non-bridge category is because of the plethora of other defect types included, namely opens, resistive vias, cell defects, and supply bridges. Additionally, only the correct net pair is used to build the decision forest described in Section 4.2.

Figure 3 shows a histogram of wired-score values for bridges (in blue) and non-bridges (black outline). It is clear from the plot that bridges tend to have greater wired-scores whereas the bulk of non-bridges have a wired-score of zero. However, it is also evident that if wired-score is used alone to separate bridge and non-bridge defects, there would be a small percentage of non-bridges that would be mis-classified as bridge defects; this is even true for the extreme case when wired-score must be one for a net pair to be classified as a bridge. Note that even a small percentage of non-bridge net pairs for c1355 will likely dwarf the number of bridges, rendering the final data set useless. When a sufficient number of features such as this one are examined simultaneously, significantly improved classification accuracy can be achieved.

### 4.1 Rule-Based Classification

The first step of classification uses two rules derived from well-known bridge fault models (Section 3.3). The first is the bridge rule, and is constructed to identify failing chips that have extremely strong indicators that the defect is a bridge. The feature values that must be satisfied for the bridge rule are shown in Table 5. A net pair is classified as a bridge defect if its features satisfy the criteria shown in the table.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wired-score</td>
<td>1.0</td>
</tr>
<tr>
<td>Wired-bridge-spread</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Same-polarity-fails</td>
<td>No</td>
</tr>
<tr>
<td>Logical-correlation</td>
<td>No</td>
</tr>
<tr>
<td>Drive-same-gate</td>
<td>No</td>
</tr>
<tr>
<td>Drive-parity-gate</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5. Bridge classification rule criteria.

The second rule is the non-bridge rule. The non-bridge
rule classifies any net pair as a non-bridge if same-polarity-fails is true. The failure cannot be caused by a non-feedback bridge between the two nets if a failure occurred when the two nets were driven to the same logic value.

The features and their corresponding values used in both rules are selected based on knowledge of the test and manufacturing domain as well as a demonstrated ability to effectively classify defects. The bridge rule is specifically designed to exclude all non-bridges, even if that means only identifying a small number of bridges. It identifies 100 of the 381 (26.2%) bridges for c1355. However, 11 of the 106,439 (0.01%) non-bridge net pairs are mis-classified as bridges. While 0.01% would in many cases be an acceptable error rate, the 11 mis-classified non-bridges account for a significant portion of the final data set (10%). Upon further investigation, the common factor among the mis-classified non-bridges is that none of the net pairs is tested thoroughly. In other words, there are logic states that could be established in the circuit that would reveal that the defect is not a bridge. Future work should investigate a method to ensure net pairs are tested sufficiently for classification.

Although the bridge rule consisting of the criteria shown in Table 5 identifies a relatively small number of non-bridges compared to bridges, it is of course desirable to increase the number of bridges identified. If the wired-score is allowed to be less than 1.0, then the number of bridges can be increased to 138 (36.2%) without mis-classifying any additional non-bridges. Figure 4 shows the tradeoff in bridge and non-bridge classification as the threshold value used for wired-score is changed with all other rule features remaining the same.

There is an optimal threshold of 0.69 where the greatest number of bridges are correctly identified while achieving the minimum number of mis-classified non-bridges. For the other features shown in Table 5, if the requirement is relaxed (i.e., wired-bridge-spread can be 0, same-polarity-fails can be true, drive-same-gate can be “yes”, etc.), the number of mis-classified non-bridges greatly exceeds the number of correctly classified bridges.

All chips not classified by either the bridge rule or the non-bridge rule are passed on to the second, more comprehensive classifier that uses a decision forest (Section 4.2).

4.2 Decision Forest Classification

The second step of classification uses a decision forest trained using SPICE-simulated defects injected into c1355. A decision forest is a machine-learning technique that can be used for data mining. A decision tree is a supervised learning technique, meaning the classifier is built (or trained) using data where each sample point has a vector of feature values and a known class [17]. In this work, each data point is a net pair, the vector of features consists of those presented in Sections 3.2 and 3.3, and the class is either bridge or non-bridge.

The tree is built automatically by identifying values of features that can be used to differentiate the classes of the data points. A decision is performed at each node in the tree based on the selected feature and value. For example, the decision at one node in the tree might be, “does this net pair have a wired-score greater than 0.75?” Each net pair that has a wired-score of 0.75 or greater would proceed to the right child of the node, whereas all net pairs with a wired-score less than 0.75 would proceed to the left child. This process continues until each net pair reaches a leaf-node class, which is labeled either bridge or non-bridge.

Decision trees have two significant advantages over other machine-learning techniques. First, they easily handle a combination of different data types. For example, one feature may have continuous values in the range [0,1] while another feature may be only true or false. Second, decision trees are easy to understand, which is a characteristic that many classifiers such as neural networks and support vector machines do not possess. The final model can be viewed graphically and easily understood by a human. This is beneficial as it allows one to examine the tree and determine what paths through the tree result in classification as a bridge.

The primary weakness of a decision tree is that it can lack stability. Small changes to the training data can lead to a different tree and thus different classifications. Bootstrap aggregation mitigates this problem by taking several random samples (with replacement) from the net pairs and building a separate tree for each random sample [17]. This collection of trees forms a decision forest. A net pair is classified by each tree in the forest, and a final, overall classification is typically determined by a majority vote from all the trees in the forest.

A net-pair set consisting of 106,820 pairs is used to train a decision-forest classifier using bootstrap aggregation (bagging [23]) for enhanced robustness. Five-fold cross validation is used to verify the model accuracy using different training (80% of the data) and validation (20% of the data) sets, a standard technique in machine learning [24]. Each data sample is a vector for a net pair consisting of a subset of the feature values described in Section 3. The features used to build the decision forest, and subsequently classify net pairs are: A-stuck-at-0, A-stuck-at-1, A-

Figure 4. Effect of wired-score threshold on net-pair classification using the rules of Table 5. All other features are kept at their nominal values.
Figure 5 shows the results of the cross-validation experiment. Specifically, Figure 5a shows the proportion of bridges correctly identified from the total set of bridges. The dark blue regions indicate the portion of bridges identified by the bridge rule discussed in Section 4.1. The light blue regions indicate the portion of bridges identified by the decision forest. The mean bridge retention rate of the five cross validation folds is 80%. Thus, on average, 80% of the non-feedback virtual bridge defects injected into c1355 are correctly identified.

Even if the bridge retention rate was 100%, it would be meaningless if a relatively significant number of non-bridges were incorrectly identified as bridges. Thus, in order to create a set of chips that only fail due to bridges, it is critical to have a very small number of mis-classified non-bridges. Figure 5b shows the proportion of bridges in the final set of chips. Bridges are shown in blue, non-bridges that were mis-classified as bridges by the bridge rule are shown in red, and the white regions represent non-bridges that were misclassified as bridges by the decision forest. The figure shows that more than 91% (on average) of the net pairs classified as bridges are in fact bridge defects. Only a small number of non-bridges are erroneously placed into the bridge set.

Less than half of the correctly classified bridges are identified by the bridge rule. If a follow-on application requires a large sample size, it is then likely tolerable to have some error in order to have a larger sample. Using a well-trained decision forest increases the retention rate of bridges from 36% to 80% over using just the bridge rule of Table 5.

In the previous experiment, every possible net pair (except those with feedback) consisting of at least one net identified by logic diagnosis was considered. While this experiment uses a realistic diagnostic output from a commercially available tool, it is important to consider the effect of the quality of diagnosis on the final classification results.

Figure 6 shows cross-validation results for the same experiment, except only net pairs where at least one net is an actual defect site are considered for classification—a situation that would only result if diagnosis is perfect. For example, for an open defect typically only one net is reported. For a cell defect, only the nets driven by the affected cell are reported. For a two-net bridge defect, either net is reported, or both are reported.

As a result of the reduced number of net pairs, the decision forest has fewer scenarios where it is misled. Figure 6 shows a significantly improved classifier for both bridges and non-bridges. The average bridge retention rate for the five cross-validation folds is now over 91%, while the average percentage of bridges in the final data set is over 97%. Note that in Figure 6, no bridges are mis-classified by the rules. These improvements indicate that the presented bridge classification strategy can directly and significantly benefit from precision diagnosis techniques that provide few misleading diagnoses. In other words, very accurate diagnosis leads to fewer net pairs that can potentially be misclassified.

## 5 Industrial Chip Experiment

In Section 4.2, a decision forest classifier was constructed using thousands of test responses generated from SPICE-simulated virtual defects that were injected into the 180nm benchmark circuit. The rules described in Section 4.1 and the decision forest model are used to classify 41 failing chips of a large, industrial design that have all undergone PFA. The 0.77cm² design is fabricated using a 90nm process, contains 800,000 scan cells, and nearly 10 million primitives in the logic model. Extensive fail logging is performed for 1,000 scan-test patterns.

Table 6 shows the results for six of the 41 chips that underwent PFA where a commercial diagnosis tool reported the chip is affected by a dominant bridge. PFA, the assumed golden standard, indicates that five of the six failures are due to an interconnect open. The remaining failure was caused by a trench issue. The table illustrates how classifying a chip as failing due to a bridge defect based solely on a tool outcome that reports a bridge fault would lead to six misclassified chips. Our approach, however, correctly classifies all six. From these results, it is clear that creating an automatic defect classification system for enabling test-based yield learning requires rigorous test-data analyses that go beyond logic diagnosis.

Unfortunately, none of the PFA results reported the existence of a bridge. While the lack of reported bridges among the PFA results prevents an evaluation of bridge-retention rate among the industrial chips, significantly, none of the 41 chips are classified as a bridge by the method presented in this paper. Thus, we achieve a mis-classification rate of 0% for non-bridges—an extremely positive result.

## 6 Discussion

In Section 4.2, a decision forest classifier was built using the test responses from SPICE-simulated defects injected into a 180nm benchmark circuit. Training the decision forest requires that the defect that leads to a test response must be...
known. Thus, SPICE-simulation is ideal because the exact nature of the defect is known, and at the same time realistic test responses are generated. The training data need not come exclusively from SPICE-simulation, however. Chips that undergo PFA also offer a source of training data that have known defect types and their associated test responses. Such chips have the advantage that the test responses are not simulated but are from an actual failing chip. PFA results can be used to supplement SPICE-simulated defects, or in a large enough quantity can replace it altogether (although this is costly and time-consuming). Any source of training data is acceptable as long as the defect causing chip failure is known, or can be determined with extremely high confidence. For example, diagnosis results correlated with thoroughly-investigated inline inspection data is another possible approach for producing high-quality training data.

The design used to train the decision forest uses a 180nm process, whereas the 41 industrial chips are manufactured using a 90nm process. It is unclear what effect the process technology has on the logical behavior of a net-to-net bridge. It is possible that differences between bridge behavior from technology to technology are not significant enough to lead to poor classification when a decision forest is built using one technology and is used to classify defects manufactured in another. Still, a more conservative approach is to ensure that the technology used to construct rules and train a decision forest is the same as the technology used in manufacturing. For this to be tractable for industrial circuits, a new hybrid SPICE/logic simulation approach would be necessary that thoroughly characterizes the defect behavior using SPICE, but performs the remainder of simulation at the logic level.

The rules and the decision forest in this paper targeted a subset of the classifications that could be performed. The problem of classifying defective chips into groups such as opens, cell defects, resistive vias, etc., was simplified into a binary one, where chips are classified based on whether they fail due to a bridge or some other defect type. The classification method presented in this paper may be extended to perform similar binary classifications (i.e., open or non-open), or a multinomial classification where each chip is classified into one of several defect classes. This approach is not exclusive to bridges, but it may be necessary to adjust the features described in Sections 3.2 and 3.3 that are used to build the decision forest.

Future work should investigate the impact of including additional defect types in the training data for the decision forest, as well as classifying chips that are known (as a result of PFA) to fail due to a bridge. Future work should also include applying the classifier to thousands of failing industrial chips to identify chips with bridges, then using those bridges to measure the DDSDs of the manufacturing process.

### 7 Summary

Logic test is poised to play a much more prominent role in process development, yield ramp, and characterization of defects that occur in a high-volume manufacturing environment. Many yield-learning activities require failing-chip samples that have some pre-selected characteristics for statistical analysis. A two-step, automated approach was pre-
sented to identify failing chips caused by upper-metal, non-feedback, two-net bridge defects. The approach uses information from the design netlist, its layout, and its tester responses.

Rules based on tester responses that closely match known bridge behaviors with additional constraints were applied to chips to identify easy-to-find bridge defects, while easy-to-find non-bridge defects were identified using a converse rule. Hard-to-find bridges were identified using a decision forest trained on thousands of virtual test responses created from defects injected at the layout level which were then extracted and SPICE simulated. Net pairs were identified for each failing chip using logic diagnosis and layout analysis. Potentially relevant features of the test data and logic design were extracted for each net pair and used as a feature vector to classify the net pair as a bridge or non-bridge using the decision forest.

The two-step approach to classification identified 36% of all bridges using the model-based rules. Once the second step was applied, an average of 80% of all bridges were identified correctly with a final data set consisting of an average of over 91% bridges. Classification involving net pairs that were formulated using only actual defect sites led to 91% of all bridges, on average, being correctly identified. The final data set consisted of an average of 97% bridges. This particular result shows that very good diagnosis will lead to outstanding classification results. However, the resolution offered by today’s diagnostic tools are more than adequate to produce significantly large populations of failing chips that are nearly homogeneous.

Forty-one industrial chips manufactured using a 90nm process and containing nearly 10 million logic primitives were correctly classified as non-bridges. Six of the 41 chips were reported to be bridges by logic diagnosis, but PFA results reported either a net open or a trench issue.

Overall, these results indicate that the method presented to classify defects can identify a significant portion of net bridges, while preventing non-bridges from being misclassified. The ability to automatically create a set of chips that fail due to bridges will enable product-test-based yield learning techniques such as automatic measurement of defect density and size distributions (DDSs) [2, 3].

References