A Learning-Based Cell-Aware Diagnosis Flow for Industrial Customer Returns

Safa Mhamdi, Patrick Girard, Arnaud Virazel, Alberto Bosio, Aymen Ladhar

A Learning-Based Cell-Aware Diagnosis Flow for Industrial Customer Returns

S. Mhamdi P. Girard A. Virazel
LIRMM, Univ. of Montpellier / CNRS
Montpellier, France
<lastname>@lirmm.fr

A. Bosio
INL, École Centrale de Lyon
France
alberto.bosio@ec-lyon.fr

A. Ladhar
STMicroelectronics
Crolles, France
aymen.ladhar@st.com

***Abstract*— Diagnosis is crucial in order to establish the root cause of observed failures in Systems-on-Chip (SoC). In this paper, we present a new framework based on supervised learning for cell-aware defect diagnosis of customer returns. By using a Naive Bayes classifier to accurately identify defect candidates, the proposed flow indistinctly deals with static and dynamic defects that may occur in actual circuits. Results achieved on benchmark circuits, as well as comparison with a commercial cell-aware diagnosis tool, show the effectiveness of the proposed framework in terms of accuracy and resolution. Moreover, the proposed flow has been experimented and validated on industrial circuits (two test chips and one customer return from STMicroelectronics), thus corroborating the results achieved on benchmark circuits.**

Keywords—Diagnosis, Customer Returns, Machine Learning

I. INTRODUCTION

The ultimate goal when developing and providing high quality (e.g. automotive) products is to have zero customer returns. A customer return is a circuit that passed the entire manufacturing test flow but failed on the customer's side [1]. The two main causes of a customer return are test escape during manufacturing test or latent defect mechanisms during lifetime. When a customer return is identified, it is important to reproduce the failure mechanism in the lab with the appropriate test conditions (temperature and voltage) and original test set. In case of test escape, efforts must be spent on finding new test patterns that will exhibit the failure in the same test conditions. In case of latent defect, the task will often succeed and a diagnosis program made of several routines is used to identify, step by step, the failing part and, finally, the suspected defects. Each routine coincides with the application of a diagnosis algorithm at a given hierarchy level. SoC level diagnosis is the first routine used to identify the core(s) in the SoC that can explain the failure [2-3]. Core level (inter-cell) diagnosis is then used to identify the possible failing cells within the core(s) [4-7]. Cell-Aware (CA) diagnosis is finally used to pinpoint the possible defect candidates within the failing cell(s) [8-10]. Note that in this case, the key metrics that characterize diagnosis performance are accuracy, i.e., the physical defect is indeed in the list of candidates, and resolution, i.e., the number of candidates reported by diagnosis for a given defective SoC.

Physical Failure Analysis (PFA) usually follows diagnosis. PFA is a time-consuming process for physically exposing the defect, and hence characterize the failure mechanism. Due to the high cost and destructive nature of PFA, diagnosis accuracy and resolution are very critical. Unfortunately, diagnosis resolution is typically far from ideal today due to SoC complexity. Especially with the advent of very deep submicron technologies (i.e., 7 nm), a high resolution (very few or one candidate) is not always reachable by today's intra-cell logic diagnosis tools based on conventional methods (effect-cause / cause-effect) [11]. For this reason, considerable effort has been spent to improve resolution by using machine learning techniques, initially through the extraction of features that allow correct candidates (those that correctly represent defect locations) to be distinguished from incorrect ones [12]–[16]. Even though they are efficient, these techniques address volume diagnosis for yield improvement, which is a different problem than fault diagnosis of customer returns. Actually, numerous data gathered during manufacturing test and subsequent diagnosis phases are available during volume diagnosis, such as, e.g., hundreds of similar failed chips with candidates correctly labeled (good, bad) obtained in a previous stage. Hence, using these data for failure diagnosis of a new failed chip is possible. On the other side, only one failed chip is investigated during fault diagnosis of a customer return, with no information about the defective behaviour of similar chips used in the same conditions (environment, workload). For this reason, learning-guided approaches used for volume diagnosis cannot be reused for fault diagnosis of customer returns.

A learning-based solution for CA diagnosis of mission mode failures in customer returns was proposed in [17]. Several supervised learning algorithms were evaluated and compared to a traditional solution to diagnose CA defects. Results obtained on benchmark circuits and compared with those of a commercial CA diagnosis tool, showed the feasibility and accuracy of this approach. However, only static defects modeled by stuck-at faults were assumed in this work. So, we proposed a new CA diagnosis method in [18]. We assumed dynamic defects and used a Bayesian classification method for predicting the nature (likelihood to be a good candidate) of each new data instance (defect). Cell-aware delay test sequences generated by a cell-aware ATPG assuming a Launch-On-Capture (LOC) testing scheme were used in this work. Once again, the effectiveness of the proposed learning-based method for diagnosis of CA dynamic defects was established, through comparison with a commercial tool.

Despite the respective efficacy of the above two methods, it is not straightforward to combine them and deal with all types of defects, i.e., static and dynamic, indifferently. This is the consequence of two distinct processes initially employed in [17] and [18] for generating new data instances that are further used to identify suspected defects. So, in an attempt to deal with all types of defect that may occur in customer returns, we propose a new CA diagnosis flow in this paper. Constructing

such a comprehensive flow raised new problems and imposed setting up a new framework with specific rules to achieve the same level of effectiveness in terms of diagnosis accuracy and resolution. The proposed method is based on a Gaussian Naive Bayes (NB) trained model to predict good defect candidates. A generic description of this method was introduced in [19], with partial results obtained on benchmark circuits. In this paper, we propose a comprehensive description of our approach to show its superiority when compared to a commercial CA diagnosis tool. The proposed flow has been experimented and validated on industrial circuits (two test chips and one customer return), thus corroborating the results achieved on benchmark circuits.

The rest of this paper is organized as follows. Section II summarizes the works presented in [17] and [18]. Section III details the new cell-aware diagnosis framework. Section IV shows results obtained on benchmark circuits and compared to those of a commercial tool. Experiments and results on industrial circuits are also presented in this section. Section V concludes the paper and discusses some aspects of the work.

II. Previous Work

Figure 1 is a generic view of the learning and prediction processes utilized in [17] and [18]. Both approaches are based on supervised learning that takes a known set of input data and known responses (*labeled data*) used as training data, trains a model to classify those data, and then uses this model to predict (infer) the class of new data.

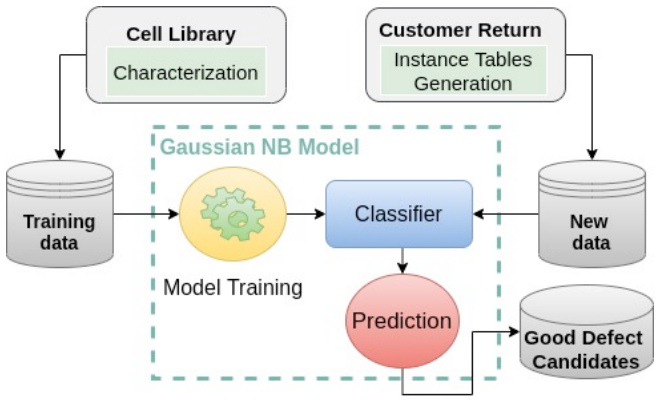


Figure 1: Generic view of the diagnosis flows in [17] and [18]

For each type of cell existing in the Circuit Under Diagnosis (CUD), training data are generated during an off-line characterization process done only once for a given cell library. Training data are extracted from cell-aware views provided by a commercial CAD tool that contains characterization results for a given cell type. These results are given in the form of a fault dictionary containing, for each defect within a cell, the cell input patterns that detect (or not) this defect. An example of training data as used in [17] and containing six instances for an arbitrary two-input cell is shown in Fig. 2. Each instance corresponds to a **static** defect D_i (last column), and a 1 (0) indicates that defect D_i is detectable (not detectable) at the output of the cell when cell test pattern P_j is applied on its inputs. Cell test patterns can be static (one input vector) or dynamic (two input vectors). There exists 2^n static test patterns and $2^n.(2^n–1)$ dynamic test patterns for an n-input cell. In Fig. 2, P1 to P4 denote static patterns (00, 01, 10, 11), and P5 to P16 denote dynamic patterns. This way of representing training data looks like a *Defect Detection Matrix* used in CA test pattern generation [20].

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	DEFECT
0	0	1	0	0	1	0	0	0	0	0	1	0	1	1	0	D11
0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	1	D12
0	1	0	0	1	0	0	0	0	0	1	0	1	0	1	1	D13
0	1	0	1	1	0	0	1	0	1	1	0	1	1	1	0	D14
1	1	1	0	1	1	1	0	1	0	1	1	1	0	0	0	D15
1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	0	D16

Figure 2: Example of training data for static defects in a two-input cell

New data are composed of various instances. Each of them is associated to one suspected cell in the CUD (customer return) and represents a features vector characterizing the real behavior of the cell during test application. From each features vector, one or more defect candidates can be further extracted and classified as good or bad candidate with a corresponding probability to be the root cause of failure. The format of a new data instance is quite similar to the format of a training data instance, but has a slightly different meaning. In each instance, the value '1' (resp. '0') is associated to a failing (resp. passing) cell test pattern P_i for a given defect candidate, meaning that the candidate is indeed detectable (resp. undetectable) by cell test pattern Pi at the output of the cell. The value '0.5' is associated to a cell test pattern for a given defect candidate when this pattern cannot appear at the inputs of a suspected cell during test application. This median value has been chosen to avoid missing information in new data instances while not biasing the features of these data.

```
-----------------------------------------------------------------
Nand Cell - ND2HVTX2
-----------------------------------------------------------------
Z   Output   L412/C1381A
A   Input    C2348/Z
B   Input    C2415/Z
-----------------------------------------------------------------
Pattern 1   FAILING   Transition on Z
Z   000000000000011 – 01
A   000000000000011 – 11
B   000000000000001 – 10
Pattern 2   PASSING   Transition on Z
Z   000000000000011 – 10
A   000000000000000 – 01
B   000000000000001 – 11
-----------------------------------------------------------------
```

Figure 3: Example of a dynamic instance table for a NAND cell

Regarding new data, they are generated after post-processing of so-called *instance tables*, which describe the behaviour (pass / fail) of each suspected cell in presence of an actual intra-cell defect (in one of the suspected cells) when a test pattern is applied to the cell. The format of a **dynamic** instance table looks like the one illustrated in Fig. 3 for a given two-input NAND cell and two dynamic cell-patterns [18]. In this example, the first part of the file gives information on how the cell is linked to other cells in the circuit, while the second part represents, respectively, the pattern number, the pattern status (failing, passing), and the cell output Z with the associated fault model for which exercising conditions are reported. These conditions shown right below each cell-pattern represent the stimulus arriving at the cell inputs during the shift phase (before '-') and applied during launch & capture cycles (after '-'). For example, cell-pattern 1 consists in applying a falling transition on input B, A being equal to static 1, and failing in detecting a rising transition on Z.

III. Proposed Cell-Aware Diagnosis Framework

In this section, we detail the various steps of the new CA diagnosis framework able to deal with all types of defect (i.e., static and dynamic) that may occur in customer returns.

A. Considered Test Protocol

With the advent of CMOS technologies, testing the scan-based logic blocks of a system-on-chip is done in several successive phases to target the various standard fault models such as stuck-at, transition, path delay, bridging, etc. Moreover, cell-aware testing is now used to increase the quality of manufacturing test by catching transistor-level defects within library cells (e.g. subtle shorts and opens intra-cell defects) that would have gone undetected using conventional fault models.

In our work, we have considered that the following tests have been applied after manufacturing. First, a static CA test sequence generated by a commercial cell-aware ATPG tool is applied to the circuit under test (CUT). This sequence targets all cell-level stuck-at faults plus cell-internal static defects, considering that these defects are not covered by a standard stuck-at fault ATPG. A standard (low speed) scan-based testing scheme is used to this purpose. Next, another option of the cell-aware ATPG is used to generate tests for cell-level transition faults plus cell-internal dynamic defects not covered by a standard transition fault ATPG. In this case, an at-speed LOC scheme has been used during test application. LOC requires two-vector test patterns, the first one is used for initialization, the second is used to generate transitions in the CUT.

As indicated, the first step of the diagnosis process is to re-use the test sequences initially used for manufacturing test. The goal is to mimic the process used during test for diagnosis preparation. So, we consider that two successive test sequences have been applied: a static CA test sequence and a dynamic CA test sequence. Note that in case additional test sequences or test schemes are used (e.g., a dynamic CA test sequence applied at low speed), the process described below can easily be adapted.

Here, dynamic defects are defects that require two-vector test patterns to be detected. These defects can be non-resistive defects modeled by stuck-open faults. More generally, these defects are mainly due to resistive opens or shorts that prevent signals to propagate within a circuit at the normal speed, and hence lead to IC failure. In this case, they are modeled by (quantitative) delay faults or (qualitative) transition faults.

B. Generation of Training Data

As indicated earlier, training data are extracted from cell-aware views provided by a commercial CAD tool containing characterization results for a given cell type. In the example shown in Fig. 2, dynamic patterns (from P5 to P16) appear in the training dataset, as it is well know that **static defects** modeled by stuck-at faults can be detected by both static **and dynamic patterns**. In this case, only the second vector of a dynamic test pattern is considered to determine whether or not a static defect is detectable by this pattern.

However, in the general case where both static and dynamic defects have to be considered, we need to take into account the fact **that dynamic defects can be detected not only by dynamic patterns, but also by static patterns** applied using a conventional scan testing scheme, provided that i) at least one transition has been generated at the cell inputs between the next-to-last scan shift cycle and the launch cycle, and ii) the delay induced by the defect is large enough to be detected (*this is always true, by definition, for a defect modeled by a transition fault, also referred to as gross delay fault*). For this reason, the training dataset in this work has a slightly different representation as shown in Fig. 4. In this case, the value '0.5' is assigned to each dynamic defect (D21 up to D23) for all related static patterns, meaning that such a defect is detectable or not depending on whether or not the above conditions are satisfied.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	DEFECT
1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	D11
1	0	0	1	0	0	0	1	0	1	0	0	1	1	0	0	D12
0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	D13
0.5	0.5	0.5	0.5	0	0	1	1	1	0	1	1	0	1	0	1	D21
0.5	0.5	0.5	0.5	1	1	1	0	1	0	1	1	1	0	0	0	D22
0.5	0.5	0.5	0.5	0	0	1	1	1	1	0	0	1	0	0	0	D23

Figure 4: Example of training data for all defect types in a two-input cell

Once training data have been generated, an important step before starting to train the Gaussian NB model is data preparation. It first consists in putting all data together and randomize the ordering. Few other manipulations are also done, such as grouping data by considering equivalent defects or removing data instances of undetectable defects. Then, it consists in splitting the data in two parts. The first part will be further used to train the model and is made of the majority of the dataset randomly selected (between 70-90%). The second part will be used for evaluating the performance of the trained model. All details about data preparation are given in [17].

C. Generation of Static and Dynamic Instance Tables

As indicated in Section II, new data are generated after post-processing of instance tables. In order to deal with both static and dynamic defects, we need to generate and use static as well as dynamic instance tables to further produce a new data instance for each suspected cell. The generation flow is illustrated in Fig. 5. First, static CA test patterns are applied to the failing CUD. We then obtain a "static" datalog containing information on the failing static CA test patterns with the corresponding failing primary outputs. From this information and the circuit netlist, we perform a logic diagnosis (by using the same commercial tool used for test generation) that gives the list of suspected cells. Finally, datalog information are used again to generate a static instance table for each suspected cell. Next, a similar process is carried out by applying dynamic CA test patterns to the failing CUD. We then obtain a "dynamic" datalog containing information on the failing dynamic CA test patterns. Datalog information are further used to generate a dynamic instance table for each suspected cell.

An important comment is the following: an instance table is generated for a given cell if and only if applying a test sequence (static or dynamic) to the CUD has led to at least one 'fail' at the circuit outputs. In other words, an instance table is generated for a given cell if the cell is a suspected cell after test application and logic diagnosis. In this context, it may happen that a cell is declared as suspect by:

- Case (1): a static CA test sequence only (in this case, the failure is due to a static defect inside the CUD *or a dynamic defect modeled by a stuck-open fault or a transition fault and not covered by the dynamic cell-aware test sequence*),

- Case (2): a dynamic CA test sequence only (in this case, the failure is due to a dynamic defect modeled by a delay fault *or a static defect not covered by the static CA test sequence*),
- Case (3): both static and dynamic CA test sequences (in this case, the failure is due to a static defect or a dynamic defect modeled either by a stuck-open fault or by a transition fault).

In the first two cases, only a static or a dynamic instance table will be generated for the suspected cell. In the last case, two instance tables (static and dynamic) will be generated. This will have an impact on the generation and utilization of new data instances, as detailed below in Subsections D and E.

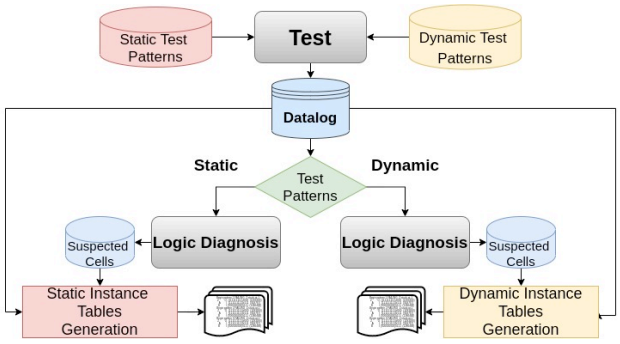


Figure 5: Generation flow of static and dynamic instance tables

Considering the above comment, we have made the following realistic assumptions (Ass_1 and Ass_2):

- Ass_1: as we consider an at-speed LOC scheme during application of the dynamic CA test sequence, it is unlikely to have a dynamic defect modeled by a transition or stuck-open fault that is detected by a static CA test sequence and not covered / detected by a dynamic CA sequence. So, in case (1), only static defects will be assumed.
- Ass_2: if a static defect is not covered / detectable by the static CA test sequence, it is unlikely that it will be detectable by a dynamic CA test sequence. So, in case (2), only dynamic defects will be assumed.

D. Generation of New Data – Cases (1) and (2)

The first step in generating a new data instance for each suspected cell consists in extracting information from each instance table associated to the cell (*cf.* Fig. 6). In case (1) discussed above, the new data will be directly extracted from the static instance table, and given as input to the NB classifier. Both static and dynamic parts of the new data will be filled with '0' and '1' values (where appropriate) since i) a static defect can be detected by a static or a dynamic pattern, and ii) these information are contained in the static instance table.

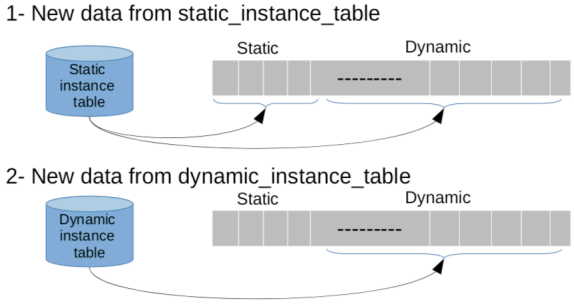


Figure 6: Extraction of new data instances

Similarly, in case (2), the new data will be extracted from the dynamic instance table. However, in this case, only the dynamic part of the new data will be filled with '0' and '1' values (where appropriate) since a dynamic defect can only be detected by a dynamic test pattern. This is in accordance with the format of a training data for a dynamic defect in which the static part is composed of '0.5' values only.

E. Generation of New Data – Case (3)

1) Observations and Conflicts

In case (3), two provisional new data are extracted, one from the static instance table (*Static New Data - SND*), one from the dynamic instance table (*Dynamic New Data - DND*). These static and dynamic new data have to be considered together to form the final new data for the suspected cell. Two strategies have been investigated initially to generate the final new data:

- Extract static (resp. dynamic) patterns information from the static (resp. dynamic) instance table to create the static (resp. dynamic) part of the static new data (resp. dynamic), and then combine both parts (*static part of the SND and dynamic part of the DND*) to create the final new data.
- Extract static and dynamic patterns information from both static and dynamic instance tables to create the full static and dynamic new data (*cf.* Fig. 6), and then mix these new data by using simple intersection rules ($0 \cap 1 = 1$, $0 \cap 0.5 = 0$, $1 \cap 0.5 = 1$, $0.5 \cap 0.5 = 0.5$) to create the final new data.

Unfortunately, these simple and straightforward strategies do not apply to our problem as several counterexamples have been found. With the first strategy, we may loose some information from the *static part of the DND* (stat_DND). For example, let us consider a NAND cell with two inputs A and B that contains a (static) short defect D_i between the gate and the drain of the NMOS transistor driven by A. The training data instance for this defect in such a NAND cell will be as follows:

00	01	10	11	0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	
1	1	0	1	1	1	0	1	0	1	1	1	1	1	1	0	D_i

Let us assume that this defect has been only sensitized and detected by the static pattern AB={01} during the static CA test sequence application. In this case, the SND will be:

0.5	1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1	0.5	0.5

Now, assume that during the dynamic CA test sequence application, the following dynamic patterns have appeared at the inputs of the cell AB={0R, 0F, R0, FR, F1, 1F}, two of them {R0, 1F} do not detecting the defect. This information will appear in the *dynamic part of the DND* (dyn_DND), but can also appear in its static part by applying an "inference" process that considers the second vector of each dynamic test pattern to re-construct (by implication) the static part. This process will be detailed subsection III.E.2. Considering this re-construction process, the DND will appear as follows:

1	1	0	0.5	1	1	0	0.5	0.5	0.5	0.5	0.5	1	1	0.5	0

By applying the first strategy, it can be seen that the static part of the final new data (0.5, 1, 0.5, 0.5), which will be equal to the static part of the SND, will be much farther away from to the training data of the defect (shown above) than the static part of the DND (1, 1, 0, 0.5), hence do not facilitating the task of the NB classifier in identifying the defect as the source failure.

The second strategy investigated to generate the final new data raises the same issue. In this case, mixing (intersecting) the static parts of the SND and DND will provide the static part of the SND (as the static part of the DND initially contains only '0.5' values), hence loosing information that could be obtained by re-construction as described above.

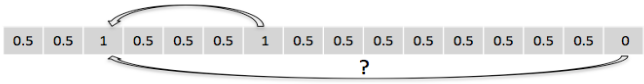
To solve the above issue, an alternate strategy has been investigated, which is based on the following key observations:

- In case of a static defect, exploiting information extracted from the dynamic instance table is recommended, as it is likely that this information will complement those extracted from the static instance table. This was illustrated in the example right above. In this case, the static part of the DND originally composed of '0.5' values must be always re-constructed to "enrich" the DND, and hence the final new data, once SND and DND are combined. A set of rules has to be defined to this purpose (see III.E.2).
- In case of a dynamic defect modeled by a stuck-open or transition (gross delay) fault, it is recommended to exploit and combine only the dynamic parts of both SND and DND, since additional information ('0' or '1') taken from the static part of the SND would deter the quality of the final new data that would be no longer so close to the training data of the defect (in which the static part only contains '0.5').
- Distinguishing between a static defect and a dynamic defect modeled by a stuck-open or transition (gross delay) fault, though not always possible, can easily be done. Identifying a static defect is possible by analyzing the static instance table (that looks like the one in Fig. 3) and looking for a 'fail' induced by the application of two identical successive vectors (no transition between them). Identifying a dynamic defect modeled by a stuck-open or transition (gross delay) fault can be done when there is occurrence of a *conflict* in SND or DND. This situation is illustrated below.

Let us consider a NOR cell with two inputs A and B that contains a full open defect between the source node of the NMOS transistor driven by A and Ground. The training data instance for this defect in such a NOR cell will be as follows:

00	01	10	11	0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	
0.5	0.5	0.5	0.5	0	0	1	0	0	0	0	0	0	0	0	0	D_i

Let us assume that this defect (a stuck-open) has been sensitized by its **unique** detection pattern P_7={R0} during the static CA test sequence application. This is possible provided that the scan shift process has brought the right initialization values {00} at the inputs of the cell. Detection will occur when applying V_2 = {10}, hence leading to a '1' on P_3 and P_7 in SND. Let us also assume that pattern P_{16}={1F} has appeared at the inputs of the cell during test, of course not sensitizing the defect (this is not possible as there is a '0' on P_{16} in the training data of the cell) and hence generating a "pass" ('0') when applying V_2 = {10} of P_{16}. In this case, a **conflict** will occur as application of V_2 = {10} during test will lead to a 'fail' when applying P_7 and a 'pass' when applying P_{16}. This conflict on the SND is illustrated below, and will be used to indicate that the defect is a dynamic defect modeled by a stuck-open or transition fault (it can not be a static defect as, otherwise, both P_7 and P_{16} would have let to a 'fail' during test application).



Such type of conflict can also appear when applying the dynamic CA test sequence. Considering the same previous (stuck-open) example, different test results when applying R0 and 1F would imply that the defect is dynamic. In the selected strategy, the occurrence of a conflict will be used to identify a dynamic defect (stuck-open or transition fault) and construct the final new data by combining only the dynamic parts of both SND and DND. However, it is important to note that a conflict does not necessarily occurs, because two conflicting patterns may not necessarily be generated by the same CA test sequence. In this case, the final new data will be generated by fully mixing SND and DND.

2) Rules for re-Constructing the Static Part of a DND

Let us consider a DND that contains values such as '0', '1' and/or '0.5'. Constructing stat_ DND that initially contains only '0.5' values cannot be done directly by simply considering '0' and '1' in dyn_DND, and applying direct implication rules (i.e., a '1' in dyn_DND for V_1V_2 implies a '1' in stat_DND for V_2, the same for '0'). The reason is that any value in dyn_DND is the consequence of the application of two successive vectors V_1 and V_2 (LOC scheme) in which V_1 may have sensitized the defect and propagated its effect up to the outputs (flip-flops), thus leading to a new V'_2 that differs from V_2. Instead, we need to consider information stored in stat_SND to infer values in stat_DND from values in dyn_DND. So, to re-construct stat_DND, the following implications rules will be applied:

- *Rule 1:* For any two-vector pattern giving a '1' in dyn_DND, if the corresponding vector V_1 is declared as fault-free during application of the static CA test sequence (i.e., V_1 has led to a '0' in stat_SND), then a '1' can be inferred in the stat_DND for vector V_2 since the defect detection can only be due to V_2.
- *Rule 2:* For any two-vector pattern giving a '1' in dyn_DND, if the corresponding vector V_2 is declared as fault-free during application of the static CA test sequence (i.e., V_2 has led to a '0' in stat_SND), then a '1' can be inferred in the stat_DND for vector V_1. In this case, as the defect is undetectable by V_2, the '1' in dyn_DND can only be due to V_1 (the defect has been propagated during the two clock cycles launched by V_1 and V_2 before being captured).
- *Rule 3:* For any two-vector pattern giving a '0' in dyn_DND, if the corresponding vector V_1 is declared as fault-free during application of the static CA test sequence (i.e., V_1 has led to a '0' in stat_SND), then a '0' can be inferred in the stat_DND for vector V_2.
- *Rule 4:* For any two-vector pattern giving a '0' in dyn_DND, if the corresponding vector V_2 does not appear during application of the static CA test sequence (i.e., V_2 has led to a '0.5' in stat_SND), then a '0' can be inferred in the stat_DND for vector V_2.

F. Overall Flow for Generating New Data

From the above assumptions, observations, and rules, we have finally set up a flow for generating a new data for a given suspected cell. Fig. 7 gives the pseudo code of this flow. It has been implemented and used in our CA diagnosis framework.

```
New_Data_Generation
1  if (#instance_table == 1) {
2     if (instance_table_type == static) {
3        New_data = SND;
4     }
6     else    // instance_table_type is dynamic
7        New_data = DND;   // without stat_DND re-construction
8  }
9  else {
10    // #instance_table = 2, at least a '1' in stat_SND and dyn_DND
11    // Research of conflict - cf. section III.E.1
12    conflict_variable = false;
13    for (dyn_SND and dyn_DND) {
14      if (conflict identified) {
15         // defect is dynamic - stuck-open or transition
16         conflict_variable = true;
17      }
18    }
19    // New data generation
20    if (conflict_variable == true) {
21       Static part of New_data = stat_DND;
22       Dynamic part of New_data = dyn_SND ⋂ dyn_DND;
23    }
24     else {   // defect can be static or dynamic
25       Re-contruct stat_DND from rules 1-4;
26       New_data = SND ⋂ DND;
27    }
28 } End
```

Figure 7: Generation of a new data for a given suspected cell

G. Cell-Aware Diagnosis Flow

The **two main steps** of the **supervised learning process** used for CA defect diagnosis are depicted in Figure 1. As mentioned, we use a Bayesian classification method for predicting the nature (likelihood to be a good candidate) of each new data instance. This choice was motivated by the result obtained in [17] after experimenting various learning algorithms and observing their inference accuracies. A cross-validation algorithm was also used to primarily calculate the prediction accuracy of these learning algorithms. So, the **first step** of our CA diagnosis flow is to generate a Naive Bayes (NB) model and to train it by using the training dataset. Training a model is done based on labeled training data and then can be used to assign a pre-defined class label to new objects. In this step, training data are used to incrementally improve the model's ability to make inference. The training data is divided into mutually exclusive and equal subsets. For each subset, the model is trained on the union of all the other subsets. Once training is complete, the performance (accuracy) of the model is evaluated by using a part of the dataset initially set aside [21]. The **second step** is to construct the NB classifier using a Gaussian distribution to model the *likelihood* probability functions, and use it to make probabilistic prediction (or inference) when a new data instance has to be evaluated. Naive Bayes classifiers work based on the Bayes' probability model that can be simply formulated as follows:

$$Posterior\ Probability = \frac{Conditional\ Probability\ .\ Prior\ Probability}{Evidence} \quad (1)$$

The *posterior probability*, in the context of our classification problem, can be interpreted as: "What is the probability that a new data instance D corresponds to a defect D_i in a suspected cell given its observed feature values?". It can be expressed as:

$$P(D=Di \mid features) \Rightarrow P(D=Di \mid T_1,\ldots,T_n) \quad (2)$$

where $T_1,\ldots,T_n$ represent the values of the cell-level test patterns associated to the new data instance D. The objective function in the Naive Bayes probability is to maximize the posterior probability given the training dataset in order to formulate the *decision rule* which is as follows:

$$D = Di \text{ if } P(D = Di \mid T_1,\ldots,T_n) \geq P(D \neq Di \mid T_1,\ldots,T_n) \quad (3)$$
Otherwise, $D \neq Di$

More details about the Naive Bayes's theorem and bayesian network classifiers, and about the way we implemented our NB classifier can be found in [22]-[24] and [18] respectively.

An important preliminary step before the above two ones is training data preparation, which is carried out in three phases: *Data Selection*, *Data Preprocessing*, and *Data Transformation*. It first consists in selecting the subset of all training data that will be used to build the model and classify new data. Then, it consists in putting all data together and randomize the ordering. In our selection process, 70% to 90% of the available data were randomly selected and this operation was repeated several times to obtain training data with good randomness. Few other manipulations are also done, such as grouping data by considering equivalent defects or removing data instances of undetectable defects. Then, it consists in splitting the data in two parts. The first part is further used to train the model and is made of the majority of the dataset randomly selected. The second part is used for evaluating the trained model's performance. More details can be found in [17].

IV. Experimental Results

Our framework has been implemented in a Python program. For validation purpose, we have experimented the proposed cell-aware diagnosis flow in four different ways:

- First, we conducted experiments on a set of ITC'99 benchmark circuits synthesized using a 28 nm FDSOI technology from STMicroelectronics.
- Next, we considered a test chip developed by STMicroelectronics and designed using a 28 nm FDSOI technology, and we performed a simulated case study with a defect injection campaign to corroborate the results achieved on the ITC'99 circuits.
- Finally, we considered two test chips (the previous one and another one designed using a 32 nm CMOS technology) and one automotive customer return from STMicroelectronics (designed with a 90 nm technology), and for each of them, we performed a silicon case study with a real defect subsequently analyzed and identified during PFA.

A. Experiments on ITC'99 Circuits

Experiments have been first conducted on benchmark circuits synthesized in a full scan manner using a 28 nm FDSOI technology from STMicroelectronics. A commercial CA ATPG tool was used to generate static and dynamic CA test sequences targeting maximum fault coverage for each circuit. For each circuit and their corresponding test set, the behavior of the tester was simulated by performing a defect injection campaign (about 2000 injections per circuit) into a number of randomly selected cells and collecting test information to build the tester data log. For the defect injection campaign, we considered each transistor of the selected cells

and we targeted all possible static and dynamic defects affecting that transistor. As several defects have the same impact on the logic behavior of the cell, and hence are logical-equivalent defects, they were grouped in **defect classes**. We used a commercial logic diagnosis tool to determine the list(s) of Suspected Cells (SC) after static and dynamic test sequences application. The average number (#aSC) for each circuit is listed in Table I, together with information about each circuit: number of cells, scan flip-flops, static and dynamic test patterns, stuck-at (including static CA) fault coverage and transition (including dynamic CA) fault coverage in %.

TABLE I. CIRCUIT FEATURES AND RESULTS OF LOGIC DIAGNOSIS

Circuit	#Cells	#SFF	#sTP	#dTP	SaFC	TrFC	#aSC
b15	2465	416	1607	2665	98.55	88.38	1.91
b17	7960	1314	2241	4423	98.83	91.10	2.39
b18	3238	215	3371	4436	99.61	93.46	2.51
b19	6337	430	3288	5583	99.57	93.79	1.64
b20	6733	430	3347	5460	99.52	93.76	1.60
b22	3218	215	3385	4670	99.64	93.78	1.64

For generating training data, we used characterization data provided by a commercial tool and ST libraries. For generating new data instances, we performed post-processing of instance tables obtained as shown in Fig. 5. From the training data and the Gaussian NB model, we make predictions on new data instances. Results obtained are a list of defect candidates with the highest probability to be the root cause of failure.

TABLE II. CELL-AWARE DIAGNOSIS RESULTS – B19

Defect	#SC	Proposed CA Diagnosis	Commercial Tool
D1	2 (A&B)	A=**D1** B=D7	A=**D1** B=D7
D2	1 (A)	A=**D2**	A=**D2**/D81
D3	2 (A&B)	A=**D3** B=D15	A=**D3**/D62 B=D9/D52
:	:	:	:
D43	2 (A&B)	A=**D43** B=D2/D44=0.5	A=**D43** B=D52/D55
D44	1 (A)	A=**D44**/D63=0.5	A=**D44**/D63
D62	2 (A&B)	A=**D62** B=D2	0
D63	2 (A&B)	A=**D63** B=D3/D48=0.5	**D63**
D64	2 (A&B)	A=**D64**/D43=0.5 B=D3/D48=0.5	A=**D64**/D63/D27/D72 B=D50
:	:	:	:
D140	2 (A&B)	A=**D140** B=D3	**A=D140**/D81=0.5 B=D50

Table II shows partial results obtained for a defect injection campaign in a three-input AndOr cell of circuit b19, with 145 potential defects including resistive and non-resistive opens and shorts. The first column lists the various injected static and dynamic defects. The second column shows the number of suspected cells obtained after logic diagnosis. Note that in this case study, the defect is always injected in the cell called A irrespective of the number of suspected cells. The next column lists the best defect candidates reported by the Gaussian NB classifier with the corresponding probability of being the root cause of failure, and provided after applying the proposed method successively on each suspected cell A and B (when two suspected cells exist). The last column reports candidates provided by a commercial CA diagnosis tool using **the same characterization data and test protocol**. This tool is non-probabilistic and provides the list of all suspects obtained after CA diagnosis with a ranking and a matching score. As this tool deals with static and dynamic patterns separately, two diagnostic reports are provided for each diagnosed failure file. Therefore, two options are possible to get the final set of suspects. The first one consists in making the **intersection** between the two diagnostic reports. This solution gives better resolution but can lead to wrong prediction (the actual defect is not reported). The second solution consists in considering the **union** between the two diagnostic reports. This solution is less accurate but is safer, especially in our case as the commercial tool behaves like a black box. In Table II, we have reported results obtained from the union of diagnostic reports.

These representative results first show that the real (injected) defect is **always** identified by the proposed flow. Sometimes, it is the only candidate and has a probability of 1 (e.g., D2) to be the best candidate. Sometimes, it is reported with another candidate in suspected cell A (e.g., D63), hence with a probability of 0.5. When two cells are suspected (e.g., D1), some defect candidates in suspected cell B are also reported, but the injected defect belongs to the whole set of candidates. Conversely, we can observe that the commercial CA diagnosis tool is **not always** able to report the injected defect as candidate (e.g., D62). This proves the superiority of our proposed framework in terms of accuracy (always 100%), which is not the case of the commercial CA diagnosis tool that sometimes can provide inaccurate results.

The reason of these cases of misdiagnosis with the commercial tool can be explained as follows. In such a tool, the logic diagnosis phase is embedded in the whole CA diagnosis flow, and no intermediate result about logic diagnosis can be observed (conversely to what is done in our learning-based flow). In such configuration, it may happen that the cell in which the real defect has been injected is found with a much lower probability to be the source of failure compared to the other identified suspected cells. In such a case, the tool may possibly ignore this cell in the next phases of the process, and finally arrive at a situation where either wrong defects or no defect will be identified as suspected candidates. Being unable to go deeper inside the functioning of the commercial tool, this is the most likely explanation about such cases of misdiagnosis.

The second comment about the results shown in Table II relates to resolution. In this example, we can observe that **in all cases**, our method provides results with a better resolution than what can be obtained with the commercial CA diagnosis tool.

TABLE III. OVERALL CELL-AWARE DIAGNOSIS RESULTS

Circuit	Accuracy		Resolution		
	Proposed	Com. Tool	Proposed	Com. Tool	
				Inter	Union
b15	100%	100%	2.047	3.79	4.68
b17	100%	100%	7.20	11.146	13.05
b18	100%	97.70%	4.129	5.733	7.56
b19	100%	98.90%	1.818	2.857	3.88
b20	100%	99.11%	2.299	2.947	3.90
b22	100%	99.12%	3.746	4.823	5.64

Results obtained on the biggest ITC'99 benchmark circuits are summarized in Table III. The second and third columns are about accuracy and give, for each circuit, the percentage of cases in which the injected defect was reported in the list of suspects provided by the proposed CA diagnosis and the commercial CA diagnosis tool respectively. As can be seen, the commercial tool is unable to achieve 100% (achieved with our technique) for 4 out of 6 circuits. The remaining columns are about resolution and give, for each circuit and considering all injection campaigns, the average number of suspects reported by the proposed method and the commercial tool respectively. The column labeled 'Inter' reports the resolution achieved

when performing the intersection between defect candidates obtained by applying static CA test sequences and those obtained by applying dynamic CA test sequences. The column labeled 'Union' reports the resolution achieved when performing the union between these two sets of candidates. Both types of resolution are reported since, again, we do not know how the commercial tool operates internally. As can be seen, the resolution achieved with our method is always better. So, overall, these results confirm the superiority of our approach in terms of accuracy and resolution. Note that the accuracy of the commercial tool was the same with either the union or intersection strategy, which explains why we have only one column in Table III for the commercial tool accuracy.

In our experiments, suspected defects were classified using a publicly available machine learning software package called Scikit-learn, which is an integrated development environment with a suite of ML tools [25]. The single defect assumption was considered, although the proposed framework is able to manage situations where multiple defects have occurred, provided that those defects are not in the same cell. This significant feature comes from the fact that our diagnosis flow considers all suspected cells one at a time, and then incrementally constructs a list of suspected defects identified in each of these cells. Finally, in-field failure mechanisms related to premature aging, such as NBTI or HCI, essentially lead to resistive opens and shorts. These mechanisms, that need to be considered in the context of customer returns, can be appropriately taken into account in our CA diagnosis flow.

The CPU time taken by the proposed flow to provide a list of defect candidates is always very low (few seconds) and does not depend on the circuit size. Only the number of suspected cells obtained after logic diagnosis may have an impact on the CPU time (for the generation of instances tables) but in a very light way (as this number is always very low as demonstrated in the results shown in Table I). The most time-consuming part of the flow (few hours) is the characterization phase, but it is done only once and is **not correlated** with the circuit size.

B. Simulated Test Case Study

We then conducted experiments on a test chip developed by STMicroelectronics and designed with a 28 nm FDSOI technology. The circuit is only composed of digital and memory blocks, except one PLL. The digital blocks are made of 1.385.864 cells. Other features are given in Table IV.

TABLE IV. MAIN FEATURES OF THE SILICON TEST CHIP #1

#cells	#PIs	#POs	#SFF	#sTP	SaFC	#dTP	TrFC
1.3M	91	33	88K	421	99.91	1238	99.93

We performed a first simulated case study with a static defect injection campaign to corroborate the results achieved on the ITC'99 circuits. We randomly and successively injected 2300 static defects in the description of the circuit. All defects were injected in a single full scan digital block composed of 203K cells, and tested with a static CA test sequence composed of 421 test patterns and achieving a stuck-at + static CA fault coverage of 99.91 %. The results obtained after executing our CA diagnosis flow and averaged over all defect injections have shown an accuracy of **100%** (the injected defect was always reported in the list of suspects) and a resolution of **1.91**. The resolution ranges between 1 and 7, and Fig. 8 shows the distribution of this resolution with respect to the total number of simulated cases (i.e., 2300 defect injections). As can be seen, in most of the cases, the number of suspects is less than 3.

We then performed a second simulated case study with a more extensive defect injection campaign. We randomly and successively injected 4800 defects (the same 2300 static defects plus 2500 new dynamic defects) in the description of the circuit. As previously, all defects were injected in a single full scan digital block composed of 203K cells. This time, as described in Section III.A, we successively applied the static and dynamic CA test sequences generated by the commercial CA ATPG tool. Details about the static CA test sequence are those given above. The dynamic CA test sequence was composed of 1238 test patterns achieving a transition + dynamic CA fault coverage of 99.93 %.

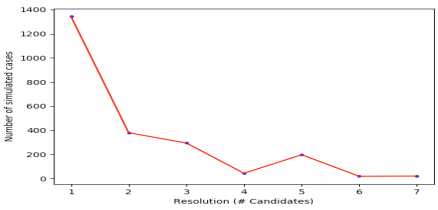


Figure 8: Distribution of the resolution wrt the simulated cases

Again, the results obtained after executing our learning-based CA diagnosis flow and averaged over all defect injections have shown an accuracy of **100%**. Regarding the resolution, we separated results obtained after injection of static defects and those obtained after injection of dynamic defects. Owing to the use of additional information provided by the dynamic CA test sequence, the average resolution obtained for the static defect injection experiments was slightly improved. The average resolution obtained for the dynamic defect injection experiments was of **1.97**. Again, the resolution ranges between 1 and 7, and the number of suspects is less than 3.

C. Silicon Test Case Studies

1) Silicon Test Chip #1

We performed a first silicon case study on the test chip considered and presented in the previous subsection, with the main features given in Table IV. The test conditions used to run the experiments were the following: a nominal supply voltage of 0.54 V, a scan test frequency of 10 MHz, a launch-to-capture clock speed (for the dynamic CA test sequence application) adjusted with respect to the nominal clock frequency of the circuit, and a temperature of 30°C. The process was considered as typical. We experimented our CA diagnosis flow, and we obtained the following results:

- The circuit failed on the tester after application of the static CA test sequence (only) when applied at the nominal voltage. This information was stored in a "static" datalog.
- A logic diagnosis gave a short list of suspected cells containing one cell of type NAND2. A static instance table was then generated for this suspected cell, and contained one failing and two passing cell level test patterns.
- From the new data generated after post-processing of this instance table, the Gaussian NB classifier identified defect D32: a short between the source and the gate of the B-input NMOS transistor of the NAND2 cell.

The above diagnosis results were provided to the Failure Analysis team and traditional techniques (laser voltage probing, static nanoprobing, EMMI, etc.) have been used to determine the correlation with the results obtained from our automated diagnosis flow. Figure 9 shows the layout view of the test chip and the incriminated transistor. As a final step, a physical defect was indeed found on the corresponding NMOS transistor after using the Planar Transmission Electron Microscope (PTEM) technique. The defect location along with the short connection observed on the failure analysis cross-sectional view is shown in Fig. 10.

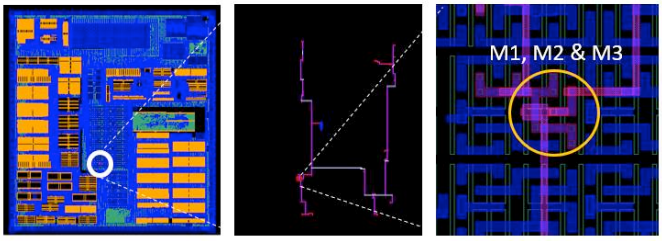


Figure 9: Layout view of the test chip and the incriminated transistor

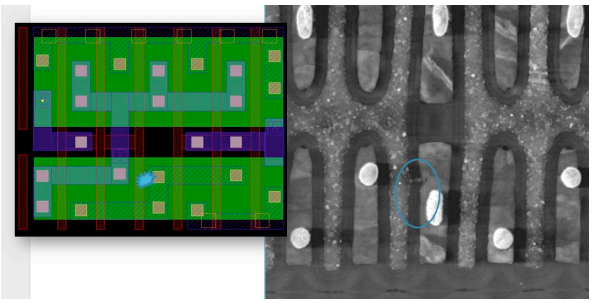


Figure 10: Physical defect found by the Failure Analysis cross-section

Note that we have also experimented the commercial CA diagnosis flow used in Section IV on this silicon case study. It was unable to identify the right defect candidate.

2) Silicon Test Chip #2

We performed a second silicon case study on another test chip developed by STMicroelectronics, designed with a 32 nm CMOS technology, and with a complexity similar to that of the first silicon test chip. We experimented our diagnosis flow and we obtained the following results:

- The circuit failed on the tester after application of the static CA test sequence (only) when applied at the nominal voltage. This information was stored in a "static" datalog.
- A logic diagnosis gave a short list of suspected cells containing a four-input AOI cell (made of 48 transistors) representing the two-level logic function AND-OR-Invert. The cell contains 431 potential short or open defects. A static instance table was then generated for this suspected cell, and contained 2 failing and 14 passing cell level test patterns.
- From the new data generated after post-processing of this instance table, the NB classifier identified defect D200: a full bridge between input A and internal net #89 of the cell.

The above diagnosis results were provided to the Failure Analysis team of STMicroelectronics, who already made a PFA on this silicon test case based on the results formerly found by their in-house intra-cell diagnosis tool. The result obtained with our CA diagnosis flow (a bridge between input A and internal net #89 of the cell) was validated at this stage, as it correlates with the former result found after analyzing the failure analysis cross-sectional view as shown in Fig. 11.

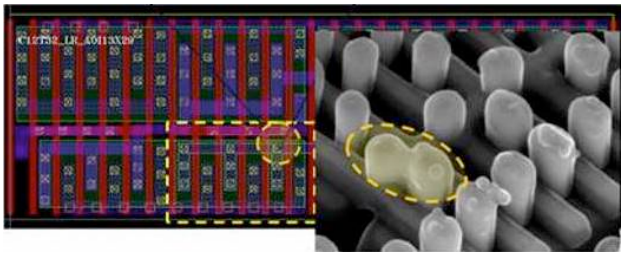


Figure 11: Physical defect found by the Failure Analysis cross-section

3) Automotive Customer Return

We performed a third silicon case study on a customer return coming from an automotive application and designed with a 90 nm CMOS technology from STMicroelectronics. The circuit contains 4.629.910 cells. Some features are given in Table V. The test conditions used to run the experiments were the following: a nominal supply voltage of 1.1 V, a scan test frequency of 10 MHz, a launch-to-capture clock speed of 1.79 GHz, adjusted with respect to the nominal clock frequency of the circuit, and a temperature of 105°C.

TABLE V. MAIN FEATURES OF THE CUSTOMER RETURN

#cells	#PIs	#POs	#SFF	#sTP	SaFC
4.6M	57	94	289K	7132	98.67

We experimented our CA diagnosis flow, and we obtained the following results:

- The circuit failed on the tester after application of the static CA test sequence (only) when applied at the nominal voltage. This information was stored in a "static" datalog.
- A logic diagnosis gave a short list of suspected cells containing a four-input AndOr cell (made of 10 transistors). The cell contains 523 potential short or open defects. A static instance table was then generated for this suspected cell, and contained 95 failing and 7 passing cell level test patterns.
- From the new data generated after post-processing of this instance table, the NB classifier identified defect D17: a short between input C and internal node #105 in the cell.

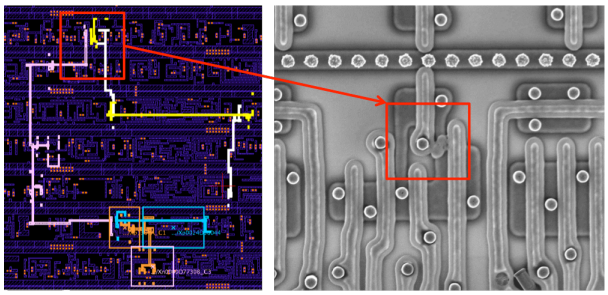


Figure 12: Physical defect found by the Failure Analysis cross-section

The above diagnosis results were provided to the Failure Analysis team of STMicroelectronics, who made a PFA in the past on this customer return based on the results found by their in-house diagnosis tool. The result obtained with our learning-based diagnosis flow was validated as it correlates with the

former result found after performing a polysilicon level inspection on the layout of the customer return and observing the failure analysis cross-sectional view as shown in Fig. 12.

V. Conclusion, Discussion and Future Work

In this paper, we have presented a new framework for cell-aware defect diagnosis of customer returns based on supervised learning. The proposed flow indistinctly deals with static and dynamic defects that may occur in real circuits. A Naive Bayes classifier was used to precisely identify defect candidates. A large set of experiments on both benchmark circuits and silicon test cases has been done to validate the proposed flow and demonstrate its efficacy in terms of accuracy and resolution.

Results of these experiments prove the appropriateness of a learning-based method to solve our problem, despite the small size of the training dataset used (only one sample for one defect class). When multiple defect sizes and test conditions will be used, this will be even truer. Indeed, multiple samples (one for each defect size or defect size range, one for each PVT test condition) will be associated to a given defect class, simply because the behavior of the defect will differ when applying the same set of test patterns. In terms of timing and complexity, this will just slightly impact our method, since training dataset is extracted from characterized cell libraries that are generated anyway for test and diagnosis purpose. So, even with large cell libraries with a huge number of defects to be simulated (e.g., 631 cells in a library, each with 4 to 6 inputs, 951 shorts and 749 opens on average – typical example of an ST library), our framework will still be easily and time-efficiently applicable.

It is worthnoting that among other factors, the effectiveness of our framework can be explained by the fact that an NB algorithm usually works well with small training dataset containing a lot of features, as in our case (e.g., for a 5-input cell, we have 1024 cell-level test patterns).

In our simulated case studies, all injected defects for evaluation purposes were present in the training dataset. Similarly, in our three silicon case studies, the actual defects were also present in the training dataset. However, in customer returns, actual defect behavior may not perfectly match the fault models that are used to train the NB classifier. Further work will be done to see how well the proposed method works in that scenario. Moreover, layout information will be used to consider only realistic defects during training data preparation, thus eliminating unrealistic defects during the learning process and possibly improving the performance of our inference engine. Finally, by exploiting the ranking of suspected cells usually provided after logic diagnosis by commercial tools, which was not done in the current work, our flow will be able to provide a similar ranking among defect candidates, thus giving additional useful information to be used during PFA.

Acknowledgements

This work has been funded by the French National Research Agency (ANR) under the framework of the ANR-17-CE24-0014-01 EDITSoC (Electrical Diagnosis for IoT SoCs in automotive) project.

References

[1] N. Sumikawa, D. Drmanac, Li-C. Wang, and M.S. Abadir, "Understanding Customer Returns From A Test Perspective," in *Proc. IEEE VLSI Test Symposium,* pp. 2-7, 2011.

[2] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and O. Riewer, "A Comprehensive System-on-Chip Logic Diagnosis," in *Proc. IEEE Asian Test Symposium,* 2010.

[3] Li-C. Wang, "Data Learning Based Diagnosis," in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conference,* pp. 247-254, 2010.

[4] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using the Single Location At a Time (SLAT)," *IEEE Transactions on Computer-Aided Design*, vol. 23, no. 1, pp. 91, 2004.

[5] S. Holst and H-J. Wunderlich, "Adaptative Debug and Diagnosis Without Fault Dictionaries," in *Proc. IEEE European Test Symposium,* pp. 7-12, 2007.

[6] D. Appello, V. Tancorre, P. Bernardi, M. Grosso, M. Rebaudengo, and M. Sonza Reorda, "Embedded Memory Diagnosis: An Industrial Workflow," in *Proc. IEEE International Test Conference*, pp.1-9, 2006.

[7] C. Zhang, Y. He, L. Yuan, and S. Xiang, "Analog Circuit Incipient Fault Diagnosis Method Using DBN Based Features Extraction," *IEEE Access,* vol. 6, April 2018.

[8] A. Ladhar, M. Masmoudi, and L. Bouzaida, "Efficient and Accurate Method for Intra-gate Defect Diagnoses in Nanometer Technology and Volume Data," in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2009.

[9] Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and E. Auvray, "Effect-Cause Intra-cell Diagnosis at Transistor Level," in *Proc. IEEE International Symposium on Quality Electronic* Design, 2013.

[10] T.P. Ho, E. Faehn, A. Virazel, A. Bosio, and P. Girard, "An Effective Intra-Cell Diagnosis Flow for Industrial SRAMs," in *Proc. IEEE International Test Conference*, 2018

[11] A. Bosio, P. Girard, S. Pravossoudovitch, and A. Virazel, "A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects," *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 289-300, 2010.

[12] Y. Xue, X. Li, R. D. Blanton, and C. Lim, "Diagnosis Resolution Improvement through Learning-Guided Physical Failure Analysis," in *Proc. IEEE International Test Conference*, 2016.

[13] X. Ren, M. Martin, and R. D. Blanton, "Improving Accuracy of On-Chip Diagnosis via Incremental Learning," in *Proc. IEEE VLSI Test Symposium*, pp. 1–6, 2015.

[14] Y. Huang, W. Yang, and W. Cheng, "Advancements in Diagnosis Driven Yield Analysis (DDYA): A Survey of State-of-the-Art Scan Diagnosis and Yield Analysis Technologies," in *Proc. IEEE European Test Symposium*, pp. 1–10, 2015.

[15] R.J. Tikkanen, S. Siatkowski, Li-C. Wang, and M.S. Abadir, "Yield Optimization Using Advanced Statistical Correlation Methods," in *Proc. IEEE International Test Conference*, 2014.

[16] Y. Xue, O. Poku, X. Li, and R. D. Blanton, "PADRE: Physically- Aware Diagnostic Resolution Enhancement," in *Proc. IEEE International Test Conference*, 2013.

[17] S. Mhandi, A. Virazel, P. Girard, A. Bosio, E. Auvray, E. Faehn, and A. Ladhar, "Towards Improvement of Mission Mode Failure Diagnosis for System-on-Chip," in *Proc. IEEE International On-Line Testing Symposium*, 2019.

[18] S. Mhandi, P. Girard, A. Virazel, A. Bosio, E. Faehn and A. Ladhar, "Cell-Aware Defect Diagnosis of Customer Returns Based on Supervised Learning," *IEEE Transactions on Device and Material Reliability*, vol. 20, no. 2, pp. 329-340, June 2020.

[19] S. Mhandi, P. Girard, A. Virazel, A. Bosio, E. Faehn and A. Ladhar, "Learning-Based Cell-Aware Defect Diagnosis of Customer Returns," in *Proc. IEEE European Test Symposium*, 2020.

[20] Z. Gao, S. Malagi, E.J. Marinissen, J. Swenton, J. Huisken, and K. Goossens, "Defect-Location Identification for Cell-Aware Test," in *Proc. IEEE Latin-American Test Symposium*, 2019.

[21] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica*, vol. 31, no. 3, pp. 249, 2007.

[22] I. Rish, "An Empirical Study of the Naive Bayes Classifier," in *Proc. International Joint Conference on Artificial Intelligence*, pp. 41–46, 2001.

[23] T. Mitchell, *Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression*, Chapter 3 - *Machine Learning*, McGraw-Hill Higher Education, 1997.

[24] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," in *Machine Learning*, vol. 29, pp.131–163, 1997.

[25] *http://scikit-learn.org/stable/user_guide.html*