Cell-Aware Defect Diagnosis of Customer Returns Based on Supervised Learning

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

Cell-Aware Defect Diagnosis of Customer Returns Based on Supervised Learning

Safa Mhamdi, *Student Member, IEEE*, Patrick Girard, *Fellow, IEEE*, Arnaud Virazel, *Member, IEEE*, Alberto Bosio, *Member, IEEE*, Eric Faehn, and Aymen Ladhar

***Abstract*— In this paper, we propose a new learning-guided approach for diagnosis of intra-cell defects that may occur in customer returns. In the first part of the paper, only static defects modeled by stuck-at faults have been assumed. Several supervised learning algorithms were considered, with various levels of efficiency. In the second part of the paper, we have extended the previous work by dealing with more sophisticated (i.e. dynamic) defects. This time, we concentrated on a Bayesian classification method used for predicting the nature (likelihood to be a good candidate) of each new data instance (defect) that has to be evaluated during the diagnosis process. Results obtained on benchmark circuits, and comparison with a commercial cell-aware diagnosis tool, demonstrate the efficiency of the proposed approach in terms of accuracy and resolution.**

***Index Terms*—Diagnosis, Machine Learning, Customer returns**

I. Introduction

Today's electronic systems are composed of complex Systems on a Chip (SoCs) that consist of heterogeneous blocks that comprise memories, digital circuits, analog and mixed-signal circuits, etc. To fit a critical application standard requirement, SoCs pass through a set of test phases at the end of the manufacturing process. The goal is to achieve near-zero Defective Parts Per Million (DPPM) so as to ensure the quality level required by the standard.

Despite the quality level (in percentage of fault coverage) of the test sequences generated by industrial or in-house tools and used during manufacturing test, SoCs may fail in mission mode due to occurrence of i) a defect not covered during the manufacturing test phase, or ii) early-life failures or failures due to various wear-out mechanisms. Early-life failures, also called infant mortality, are caused by defects that are not exposed during manufacturing tests, but that are degraded due to electrical and thermal stress during in-field use, and lead to a failure in functionality. Wear-out (or aging) manifesting as progressive performance degradation, is induced by various mechanisms such as Negative-Bias Temperature Instability (NBTI) or Hot-Carrier Injection (HCI).

Manuscript submitted on January 2020. Revision submitted on April 2020.

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.

S. Mhamdi, P. Girard and A. Virazel are with the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM), University of Montpellier / CNRS, Montpellier, 34095 France (e-mail: firstname.lastname@lirmm.fr).

A. Bosio is with the Lyon Institute of Nanotechnology (INL), Ecole Centrale de Lyon, Lyon 69000 France (e-mail: alberto.bosio@ec-lyon.fr)

E. Faehn and A. Ladhar are with STMicroelectronics, Crolles, 38920 France (e-mails: eric.faehn@st.com, aymen.ladhar@st.com).

Such failures that occur during the mission mode are the most critical as they may result in catastrophic consequences. Thus, in an attempt to identify the source of these failures and avoid their re-occurrence in next generation products, the defective SoC (referred to as "customer return") is always sent back to the manufacturer who is in charge of analyzing the device to determine the root cause of failures [1]. In this scenario, failures are not easy to reproduce in the company lab as the real mission conditions and executed workload are generally unknown and cannot be exhaustively modeled. Therefore, efficient diagnosis methods to locate and assess failures at different system levels are of vital importance.

Diagnosis is usually followed by Physical Failure Analysis (PFA), a time-consuming process for exposing the defect physically in order to characterize the failure mechanism. Due to the high cost and destructive nature of PFA, diagnosis resolution is of critical importance. In practice, it is very uncommon to perform PFA on any defect with more than five candidates [2]. Ideally, resolution is one, that is, a single location is identified when a defect is diagnosed. This ensures that the likelihood for uncovering the root-cause of failure is maximized when performing PFA. However, with the advent of very deep submicron technologies, such a resolution is not always reachable by today's intra-cell logic diagnosis tools based on conventional methods (effect-cause / cause-effect). In this context, machine learning can be viewed as an efficient mean to exploit data (logical or physical) other than that used by conventional methods to improve diagnosis resolution [3].

In this paper, we present a learning-guided approach for diagnosis of intra-cell defects that may occur in customer returns. In the first part of the paper, simple (static) defects modeled by stuck-at faults have been assumed. We have considered static cell-aware test sequences generated by a commercial cell-aware Automatic Test Pattern Generation (ATPG) tool assuming a standard (low speed) scan-based testing scheme. 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. Several supervised learning algorithms were considered, with various levels of efficacy. Results obtained on combinational benchmark circuits, and comparison with a commercial cell-aware diagnosis tool, show the feasibility and accuracy of this approach [4]. In the second part of the paper, the previous work has been extended by dealing with more sophisticated (i.e. dynamic) defects. We have considered cell-aware transition test sequences generated by a cell-aware ATPG assuming a Launch-Off-Capture scan-based testing scheme. A new cell-aware defect diagnosis method is thus proposed, this time concentrating on a Bayesian classification method for predicting the nature (likelihood to be a good candidate) of

each new data instance (defect) that has to be evaluated. As for diagnosis of static defects, the efficiency of the proposed Bayesian method for diagnosis of intra-cell dynamic defects is demonstrated through comparison with a commercial tool.

The rest of this paper is organized as follows. Section II presents a state-of-the-art on diagnosis methods and gives the motivations of this work. Section III presents the proposed learning-guided cell-aware static defect diagnosis approach. Section IV presents the cell-aware dynamic defect diagnosis method based on Bayesian classification. Section V presents results obtained on two sets of benchmark circuits, as well as a comparison with a commercial tool. Section VI concludes the paper and discusses missing aspects as well as future work.

II. State of the Art and Motivations

Diagnosis is the first analysis step for a defective SoC. This is a software-based method that analyzes the applied tests, the tester responses, and the netlist (possibly with layout information) to produce a list of candidates that represent the possible locations and types of defects (or faults) within the defective SoC [5]. The key metrics that characterize diagnosis performance are resolution, *i.e.*, the number of candidates reported by diagnosis for a given defective SoC, and accuracy, *i.e.*, the physical defect is indeed in the list of candidates.

In the case of a customer return, the first step is to re-use the original test program to check if the SoC fails again or not. If not, efforts have to be made to find new test patterns and test conditions (i.e. voltage and temperature) that will sensitize the defect and reveal the failure. Otherwise, if the SoC fails, a diagnosis program made of several routines is used to identify, step by step, the failing part and, finally, the suspected defects. Each routine corresponds to the application of a diagnosis algorithm at a given hierarchy level. *SoC level diagnosis* is the first routine used to identify the cores or interconnections in the system that can explain the failure. *Core level diagnosis* (inter-cell diagnosis) is then used to identify the possible failing cells within a core. *Intra-cell diagnosis* is finally used to pinpoint the possible defect candidates within a cell.

Except industrial in-house SoC diagnosis tools, very few comprehensive diagnosis approach able to deal with a full SoC and providing reliable information about fault localization exist. To the best of our knowledge, the only work targeting SoC-level diagnosis is reported in [6]. The key concept is that diagnosis consists in a comparison between a set of pre-computed SoC failures and the set of failures observed during test. This type of approach was formerly proposed in [7] and [8] but only for full-scan circuits. In [6], authors propose to extend it to the case of SoC. The main advantages of this approach w.r.t. the state-of-the-art are (i) the capability to manage both full-scan and sequential logic cores, (ii) to deal with several fault models at a time (both static and dynamic) and (iii) to address both single and multiple fault occurrences.

Regarding core-level diagnosis, a considerable amount of work can be found in the literature. Dedicated techniques have been proposed to target specific cores: logic cores (logic diagnosis) [7]–[9], memory cores (memory diagnosis) [10–11] and analog cores (analog diagnosis) [12–13]. Considering logic diagnosis, the result is either an interconnection between gates or a suspected gate. Faults can hence occur either in the interconnection between gates (inter-cell faults) or inside the gate (intra-cell faults). When the observed failure is inside the gate, cell-aware diagnosis is applied to locate the cause of this failure at the transistor level [14]. An intra-cell diagnosis method used for mission mode failures in customer returns has been proposed in [15]. It uses a Critical Path Tracing (CPT) algorithm applied at transistor level and works as follows. First, the test determines which are the failing and passing test patterns for a given Circuit Under Test (CUT). Then, *logic diagnosis* exploits this information to determine a list of suspected gates (candidates). Any available logic diagnosis tool can be used. For each suspected gate, we have to know the logical values applied to it when failing and passing test patterns are applied to the CUT. This step amounts to determine the actual set of failing/passing test patterns at the cell level. Finally, *intra-cell diagnosis* is executed for each suspected gate and its related failing /passing test patterns. The result is a list of suspected nets at transistor level with a set of fault models able to explain the observed failures. More details about this flow and results obtained on industrial circuits from STMicroelectronics (STM) are in [15].

Unfortunately, for various reasons, diagnostic resolution is typically far from ideal due to the SoC complexity. As a result, a lot of efforts have been dedicated for improving diagnosis resolution. Among several types of solutions, it has been demonstrated recently that diagnosis resolution can be improved with Machine Learning (ML) techniques, primarily through the derivation of characteristics that enables correct candidates (candidates that correctly represent defect locations) to be distinguished from incorrect ones (candidates that do not) [16]–[21]. In [16], authors describe an approach to identify bridge defects from a population of diagnosed defects by using a combination of effective rules and a decision-tree-based classifier. In [17], authors improve on-chip diagnosis resolution with a modified k-nearest neighbors classifier that is updated with real-time failure data. In [18], volume diagnosis resolution is improved with a Bayesian classifier that identifies the actual candidates based on their layout properties. In [19], authors present a novel yield optimization methodology based on establishing a strong correlation between a group of fails and an adjustable process parameter. The core of the methodology comprises three advanced statistical correlation methods. In [20], authors use statistical learning methods to predict the termination of tester-data collection to ensure good resolution. In [21], a learning-based resolution improvement approach called PADRE (Physically-Aware Diagnostic Resolution Enhancement) is proposed. PADRE uses a Support Vector Machine algorithm to analyze easily available tester and simulation characteristics about the candidates to identify those that correspond to the actual failure locations. The capabilities of this solution have been further extended with a novel Active Learning (AL) based PFA selection approach [2]. AL-PADRE selects the most useful defects for PFA so as to improve diagnostic resolution.

Despite their efficiency, a common feature of these techniques is that they all address volume diagnosis for yield improvement which is a different problem than fault diagnosis of customer returns. During volume diagnosis, numerous data collected during manufacturing test and subsequent diagnosis phases are available, such as, e.g. hundreds of similar failed

chips with candidates correctly labeled (good or bad) obtained in a previous stage. It is therefore possible to use these data for failure diagnosis of a new failed chip. Conversely, during fault diagnosis of customer returns, only one failed chip is investigated, with no information about the defective behavior of some other similar chips used in the same conditions (application, environment, workload, etc.). For this reason, learning-guided approaches used for volume diagnosis cannot be used for fault diagnosis of customer returns.

III. Cell-Aware Static Defect Diagnosis

Despite the good resolution achievable with conventional intra-cell diagnosis technique, in some cases (e.g. complex cells, complex failure mechanisms) the number of candidates is too high to allow an efficient PFA. This problem will be exacerbated with the advent of very deep submicron (i.e., 7 nm) technologies. Improving intra-cell diagnosis efficiency is therefore mandatory. A mean to achieve this goal is to use supervised learning algorithms to determine suspected defects. Supervised learning is now used in numerous classification problems where the knowledge on some data can be used to classify a new instance of such data. In this section, we present a new approach that uses supervised learning instead of traditional cause-effect and/or effect-cause analysis to identify static defect candidates within a cell with a high accuracy.

A. Overall Diagnosis Flow

Figure 1 shows the proposed diagnosis flow [4] based on supervised learning that takes a known set of input data and known responses (*labeled data*) used as training data, trains a model, and then implement a classifier based on this model to make predictions (*inferences*) for the response to new data.

Training Data are generated for each type of cell existing in the Circuit Under Diagnosis (CUD) during an off-line characterization process done only once for a given cell library. It takes as input i) the cell-level test patterns, i.e., all possible static and dynamic combinations of values at the inputs of a cell, ii) the list of all possible types of cell (NAND, NOR, etc.) in the CUD, and iii) the cell netlists at transistor level. From these inputs, intra-cell transistor-level defect simulations using Spice are performed by iteratively injecting all possible defects into each cell type and then simulate the behavior of the cell. The output is a set of instances associated to each type of cell, and representing the training data. An example of partial training data with six instances for an arbitrary two-input cell is shown in Fig. 2. Each instance is associated 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 the cell test pattern Pj is applied at the inputs of the cell. Cell test patterns are static (one input vector) or dynamic (two input vectors). For an n-input cell, there exist 2^n static test patterns and $2^n.(2^n - 1)$ dynamic test patterns. In Fig. 2, P1 to P4 denote static patterns (00, 01, 10, 11), while P5 to P16 denote dynamic patterns. Dynamic patterns appear in the training dataset, as it is well known that static defects modeled by stuck-at faults can be detected by both static and dynamic patterns. In this later 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. Note that this way of representing training data looks like a *Defect Detection Matrix* used in cell-aware test pattern generation [22].

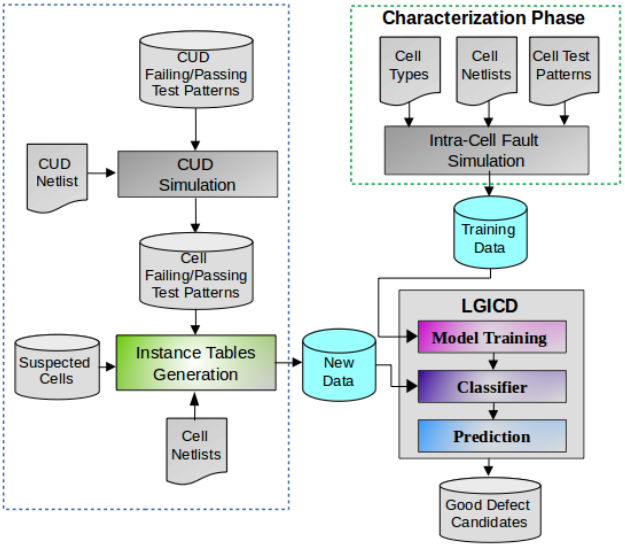

Fig. 1. Learning-guided intra-cell diagnosis flow

Besides training data, the Learning-Guided Intra-Cell Diagnosis (LGICD) module receives **New Data**. Each instance of the new dataset is associated to one suspected cell in the CUD (customer return) and represents a features vector that characterizes the real behavior of the cell during test application. From each features vector, we can further extract one or more defect candidates that have to be classified as good or bad candidate with a corresponding probability to be the root cause of failure. New data are generated after post-processing of so-called *instance tables* describing the behavior (pass / fail) of each suspected cell in presence of an intra-cell defect (in one of the suspected cells) when a test pattern is applied to the cell. An example of a dynamic instance table is given in the next section. As shown in Fig. 1, these instance tables are obtained from i) the list of suspected cells provided by a logic diagnosis tool and ranked according to their score to be the source of failure (to contain the real defect), ii) the netlist of each suspected cell, and iii) the cell failing/passing test patterns. These patterns are obtained by performing a simple logic simulation of the CUD with the failing/passing test patterns identified by the tester.

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

Fig. 2. Example of partial training data for static defects in a two-input cell

The format of a new data instance is quite similar to that of a training data instance, but has a different meaning. In each instance, the value 1 (respectively 0) is associated to a failing (respectively passing) cell test pattern Pi for a given defect candidate, meaning that the candidate is indeed detectable (respectively undetectable) by the cell test pattern Pi at the circuit output. In such instance, the value 0.5 is associated to a cell test pattern for a given defect candidate when this pattern does not exist in the list of cell failing/passing test patterns

(i.e., the cell test pattern cannot appear at the inputs of a suspected cell during test application). The median value 0.5 has been chosen to avoid missing information in new data instances while not biasing the features of these data. An example of a new data instance for a two-input cell is given in Fig. 3. R stands for rising (0 to 1) transition on a cell input, and F stands for falling (1 to 0) transition on a cell input.

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

Fig. 3. Example of a new data instance for a two-input cell

Finally, from the training and new dataset, LGICD provides a set of good transistor-level defect candidates with the corresponding probability to be the root cause of the failure.

B. Details of the LGICD Module

The first phase of the LGICD consists in data preparation. Since learning algorithms learn from data, it is essential to use data that perfectly match the problem to be solved. Data have to be in a useful format and include meaningful features. In our case, the process for getting data ready for machine learning algorithms can be summarized in three phases [23]:

1) *Data Selection*. It consists in selecting the subset of all available data that will be used to build a model and classify new data. In our case, available data are the training data obtained from the off-line characterization process. Each instance of the training data is associated to a defect, and corresponds to the behavior of the cell (fault-free or faulty) in presence of such defect and for all possible combinations (static and dynamic) of the cell inputs. Some defects lead to the same cell behavior. These defects are called *equivalent defects*. Some others are undetectable by any cell test pattern. 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.

2) *Data Preprocessing*. Once training data have been selected, we need to consider how they will be used. Training data are first stored in a CSV file. Then, they are sampled and grouped by considering equivalent defects. All equivalent defects are thus associated to a given Defect Class *i* (DCi). Training data instances of undetectable defects are removed.

3) *Data Transformation*. The final phase is to transform the preprocessed data ready for learning in a format manageable by the classifiers (or models). In this format, each instance of a training data contains $m+1$ columns, where $m = 2^n + 2^n.(2^n - 1)$ for an n-input cell (e.g. 16 for a 2-input cell). Columns 1 to m correspond to the exhaustive cell test patterns. Column $m+1$ corresponds to each defect class. The names of each column are specified when transforming data. This will help to explore these data in a later stage of the process.

In the second phase of the LGICD, we build models based on different classification algorithms (called classifiers). As we preliminary do not know which algorithm will be efficient for our problem, evaluating the performance of the selected algorithms is an important step. These evaluations are most often based on prediction accuracy (the percentage of correct prediction divided by the total number of predictions). There are many techniques used to calculate the accuracy of a classifier. The technique used in this work is known as cross-validation. The training data is divided into mutually exclusive and equal subsets. For each subset, the classifier is trained on the union of all the other subsets [24].

Finally, once we have selected the models, we make predictions on new data instances with all of the available data. In our case, the expected results correspond to a defect's class probability of being the root-cause of failure. Each model returns the best defect's class candidate and the probability for each class has a value between 0 and 1.

C. Selected Supervised Learning Algorithms

In this work, suspected defects were classified using a publicly available machine learning software package called Scikit-learn [25]. Scikit-learn is an integrated development environment with a suite of ML tools. Various tools of Scikit-learn with supervised learning algorithms for classification have been used in this work. The selected algorithms are the following: Logistic Regression (LR), K-Nearest-Neighbors (KNN), Naive Bayes (NB) Classifier, and Support Vector Machines (SVM). These algorithms represent a mixture of linear (LR) and nonlinear (KNN, NB, SVM) algorithms.

IV. Cell-Aware Dynamic Defect Diagnosis

As indicated at the beginning of the paper, we have extended the previous work by dealing with dynamic (open and short) defects. A new cell-aware dynamic defect diagnosis method is thus presented in this section. This time, we concentrated on a Bayesian classification method for predicting the nature (likelihood to be a good candidate) of each new data instance (defect) that has to be evaluated. This choice comes from the results obtained in [4] after experimenting several learning algorithms and observing their prediction accuracies. These results are partially reported again in Section VI.A.

Except those that may lead to stuck-open faults, dynamic defects are mainly due to resistive opens or shorts that prevent signals to propagate within a circuit at the normal speed. Dynamic defects induce delayed signals that may prevent good logic values to be captured in flip-flops during functional mode of operation, and hence lead to circuit failure. With the advance of deep submicron technologies, the occurrence of dynamic defects is constantly increasing, not only during the manufacturing process of ICs, but also during the lifetime of the circuit where latent or wear-out defects may appear due to various stress conditions (functional, environmental, etc.).

Dynamic defects can be modeled by delay or transition faults. Testing for delay faults is often done through at-speed scan testing for logic circuits. At-speed scan testing consists of using a rated system (nominal) clock period between launch and capture for each delay test pattern, while a longer clock period is normally used for scan shifting. In order to test for transition faults, two different testing schemes are used in practice during at-speed scan testing: Launch-off-Shift (LOS) and Launch-off-Capture (LOC). Although LOS and LOC have different but complementary delay fault coverage, LOC is the preferred scheme in industry due to its easier implementation. LOC requires two-pattern tests, where the first vector is used for initialization and the second is used to generate transitions.

Here, we assume that a LOC scheme (also called "Fast Sequential") has been used during test application. A cell-

aware ATPG has been used to generate tests for gate-level transition faults plus cell-internal dynamic defects not covered by a conventional transition fault ATPG. Though (cell-aware) transition test patterns are used for delay faults, and hence detect dynamic defects, they can detect static defects as well. For this reason, both static and dynamic defects could be considered by our diagnosis method, although only **dynamic** defects have been investigated in this part of our work.

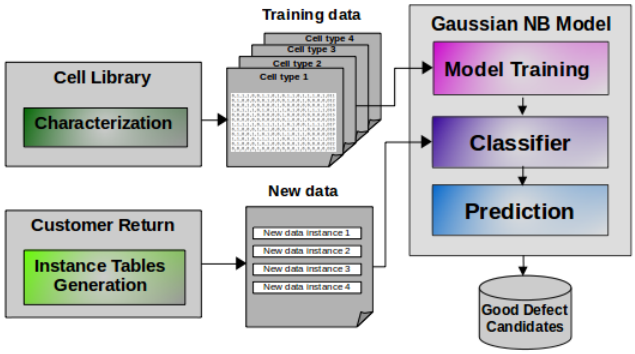

Fig. 4. Proposed intra-cell dynamic defect diagnosis flow

Figure 4 shows the flow of the proposed cell-aware dynamic defect diagnosis method. Training data are generated as described in Section III.A and have the format shown in Fig. 2. They are extracted from cell-aware views provided by a commercial Computer-Aided-Design (CAD) tool that contain all characterization results for a given cell type. These results are provided in the form of a fault dictionary containing, for each defect within a cell, the cell input patterns detecting (or not) this defect. New data are generated after post-processing of instance tables describing the behavior (pass / fail) of each suspected cell in presence of an intra-cell dynamic defect (in one of the suspected cells) when a transition test pattern is applied to the cell. The format of an instance table looks like the one illustrated in Fig. 5 for a given three-input AndOr cell and two test patterns. In this example, the first part of the file gives information on how the cell is linked to other gates in the circuit, while the second part represents, respectively, the pattern number, the pattern status (failing, passing), and the cell output with the associated fault model for which the exercising conditions are reported. Exercising conditions shown right below each pattern represent the stimulus arriving at the cell inputs during the shift phase (before '-') and applied during launch & capture cycles (after '-'). For example, Pattern 1 consists in applying a falling transition on input A ; B and C be equal to static 1 and 0 respectively ; and failing in detecting a falling transition on output Z.

The way to generate instance tables in our diagnosis flow is illustrated in Fig. 6. First, cell-aware transition test patterns are applied to the failing CUD (customer return). Remember that in our method, each test sequence is obtained from a commercial cell-aware test pattern generation tool that targets intra-cell defects. We then obtain a datalog containing information on the failing test patterns and 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. By using datalog information, we can finally generate an instance table for an each suspected cell.

```
-----------------------------------------------
              AndOr Cell - A0127
-----------------------------------------------
    Z    Output  L541/C1217
    A    Input   C1198/Z
    B    Input   C1335/Z
    C    Input   C1279/Z
-----------------------------------------------
  Pattern 1 FAILING Transition on Z
    Z   000000000000011 - 10
    A   000000000000011 - 10
    B   000000000000011 - 11
    C   000000000000000 - 00
  Pattern 2 PASSING Transition on Z
    Z   000000000000000 - 01
    A   000000000000000 - 00
    B   000000000000011 - 10
    C   000000000000000 - 00
-----------------------------------------------
```

Fig. 5. Example of a dynamic instance table for an AndOr cell

As mentioned earlier, all instance tables are post-processed to provide a number of new data instances, each representing one or more defect candidates that have to be classified according to their likelihood to be the root cause of failure. The format of a new data instance has been described in the previous section.

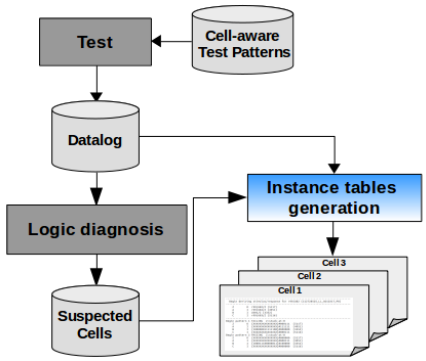

Fig. 6. Generation flow of instance tables

The core block in Fig. 4 depicts the two main steps of the supervised learning process used for intra-cell dynamic defect diagnosis. As indicated at the beginning of the paper, we use a Bayesian classification method for predicting the nature of each new data instance. So, the first main step consists in generating a NB model and to train it by using the training dataset. The second main step consists in constructing the NB classifier by using a Gaussian distribution to model the *likelihood* probability functions, and use this classifier to make probabilistic prediction (or inference) when a new data instance has to be evaluated. These two main steps are detailed in the next section. Note that an important preliminary step before the above two ones is training data preparation, already detailed in subsection III.B.

V. Prediction Model based on a Bayesian Classification Method

The goal of classification is to classify an instance of a class based on the value of several attributes (or features). Many classification approaches attempt to explicitly construct a function from the joint set of values of the attributes to make classification. Examples of such classifiers include decision trees and neural networks. Bayesian classification takes a quite different approach to this problem, by approximating the joint probability distribution of the class and the attributes [26]. Therefore, learning in Bayesian classification amounts to estimate this joint probability distribution. After constructing such an estimate, we classify a new instance of a class by

examining the conditional probability given the particular attribute values, and return the class that is the most probable. Bayesian inference (prediction) has been successfully used in analog-circuit diagnosis and board-level diagnosis [27-28]. Here, it is used for cell-aware dynamic defect diagnosis.

A. Naive Bayes Model Generation and Training

Bayesian classification is the general term defining a type of classification algorithm based on Bayes's theorem, which is an equation describing the relationship of conditional probabilities of statistical quantities. Bayesian classification aims at finding the probability of a class *C* given some observed *features*, which can be written as *P(C|features)*. Bayes's theorem expresses this in terms of quantities that can be computed more directly:

$$P(C|features) = P(features|C) \,.\, P(C) \,/\, P(features)$$

To do that, we need a model by which we can compute *P(features|C)* for a given class. Such a model is called a *generative model* because it specifies the hypothetical random process that generates the data. Specifying this generative model is the main piece of the *training* of such a Bayesian classifier. The general version of such a training step is a very difficult task, but we can make it simpler through the use of some simplifying assumptions about the form of this model. Clearly, by making naive assumptions about the generative model, we can find a rough approximation and then proceed with the Bayesian classification.

NB classifiers make the assumption of independence among predictors [29]. In simple terms, a NB classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Features are those characteristics (or attributes) that affect the results of a class label. In our training dataset, features are represented by the cell-level test patterns associated to a given class label, which itself represents a dynamic intra-cell defect. 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 the part of the dataset initially set aside.

B. Naive Bayes Classifier and Inference

Bayesian theory provides an efficient intuitive approach for drawing inferences from observations and *a priori* beliefs. NB 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:

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

where $T_1,\ldots,T_n$ represents the values of the cell-level test patterns associated to the new data instance *D*.

The objective function in the NB probability is to maximize the posterior probability given the training data in order to formulate the decision rule. This *decision rule* can be formulated based on the posterior probabilities as follows:

$D = Di$ if $P(D = Di \mid T_1,\ldots,T_n) \geq P(D \neq Di \mid T_1,\ldots,T_n)$
Otherwise, $D \neq Di$

An additional assumption of NB classifiers is the conditional independence of features. Under this *naive* assumption, the class *conditional probabilities* (or *likelihoods*) of the new data instances can be directly estimated from the training data instead of evaluating all possibilities of *T*. Thus, given a n-dimensional feature vector T, the class *conditional probability* *P(T|Di)* can be calculated as follows:

$$P(T|Di) = P(T1|Di) \bullet P(T2|Di) \bullet \ldots \bullet P(Tn|Di) = \prod_{k=1}^{n} P(Tk|Di) \quad (2)$$

Here, *P(T|Di)* simply means: "How likely is to observe this particular pattern *T* given that it belongs to class *Di*?" The individual likelihoods for every feature in the feature vector *T* can be estimated via the maximum-likelihood estimate [29].

The third element in Expression (1) is the *prior probability* that can be interpreted as the *prior belief* or *a priori* knowledge. In the context of pattern classification (a pattern corresponds to a new data instance in our case), the prior probabilities are also called *class priors*, and describe "the general probability of encountering a particular class". If the priors are following a uniform distribution, the posterior probabilities will be entirely determined by the class conditional probabilities and the evidence term. And since the evidence term is a constant, the decision rule will entirely depend on the class conditional probabilities.

After defining the class conditional probability and prior probability, there is only one term missing in order to compute posterior probability, that is the *evidence*. The evidence *P(T)* can be understood as the probability of encountering a particular pattern *T* independent from the class label. Although the evidence term is required to accurately calculate posterior probabilities, it can be removed from the *decision rule* since it is merely a scaling factor.

It is worthmentioning that in multi-class classification, each new data instance may be assigned multiple class labels (defect *Di* in our case). The NB algorithm is well known for multi class prediction feature. In this work, we can predict the probability of multiple classes of target variable.

To implement the classifier and then make inferences, we need to use a model representing the way features are distributed in feature vectors that correspond to new data instances. When the probability distributions of the features follow a normal (Gaussian) distribution, the Gaussian NB model can be used. In this work, we use a Gaussian kernel to calculate the class conditional probabilities.

In order to illustrate the distribution of features, and hence demonstrate the appropriateness of the Gaussian model used in our NB classifier, we plot in Fig. 7 the result obtained for an AND2 cell. Unimodal and bimodal Gaussian distributions were achieved depending on the type (static or dynamic) of

considered cell-level test patterns (P1 to P12).

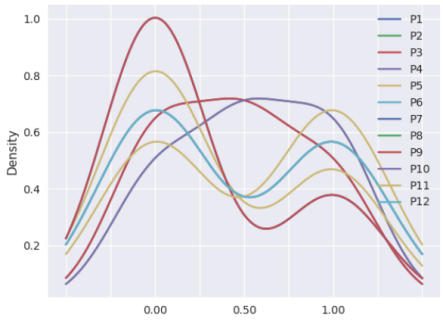

Fig. 7. Gaussian distribution of features in an AND2 cell

VI. Experimental Results

The two versions of the cell-aware defect diagnosis method presented in the previous sections, one for static defects, one for dynamic defects, have been implemented separately in a Python program. Results are presented and discussed below.

A. Results of Cell-Aware Static Defect Diagnosis

We conducted experiments on ISCAS'85 benchmarks circuits synthesized in a 28nm FDSOI technology from STM. A cell-aware commercial ATPG tool was used to generate test patterns for each circuit by targeting static cell-internal defects in addition to stuck-at faults. Test patterns were generated to achieve 100% stuck-at fault coverage. From each circuit and the corresponding test set, we simulated the behavior of the tester by performing a defect injection campaign (about 500 injections per circuit) into a number of randomly selected gates and collecting test information to build the tester data log. For the defect injection campaign, we considered each transistor of the selected gates and targeted all possible static defects affecting that transistor. These defects are shown in Fig. 8 and are as follows:

- RO_i: full open defect at node i (i = [Gate, Drain, Source, Bulk])
- RS_{ij}: full short defect between nodes i and j (i/j = [Gate, Drain, Source, Bulk])

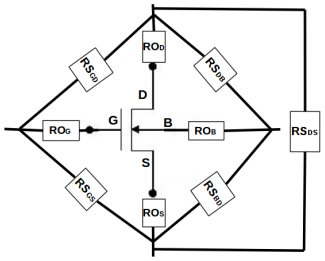

Fig. 8. Considered transistor defects

For example, the number of defects for a NAND2 gate is equal to 36 defects (9 defects per transistor). However, several defects have the same impact on the logic behavior of the gate. So, these defects are logical-equivalent defects and hence are grouped in defect classes. Table I shows the equivalent defects with the corresponding defect classes for such a gate. In this table, *Dtk* refers to a defect in transistor t (t ranges from 1 to 4), and k indicates the type (*RO* or *RS*) and source node (Gate, Drain, Source, Bulk) of the defect. Labels from 1 to 4 for k refer to open defects. Labels from 5 to 9 refer to short defects.

TABLE I. NAND2 Defect Classes

Defect Class	Equivalent Defects
DC1	D11, D12, D13, D16, D21, D22, D23, D26, D38, D39, D48, D49
DC2	D14, D24, D28, D34, D37, D44, D47
DC3	D15, D36
DC4	D17
DC5	D18, D27, D29
DC6	D19
DC7	D25
DC8	D31, D32, D33, D35
DC9	D41, D42, D43, D45
DC10	D46

From the list of failing/passing test patterns with the corresponding failing/passing CUD outputs (datalog), a logic diagnosis tool based on fault simulation was used to determine a list of suspected cells ranked according to their score to be the source of failure. We used a commercial diagnosis tool to this purpose. For most of experiments, the list of suspected cells contained the cell in which the defect was injected. In very few cases, the commercial tool was unable to identify the faulty cell as suspect. Learning-guided cell-aware diagnosis was not done in such cases. The average number of Suspected Cells (#aSC) for each circuit is listed in Table II, together with information about each circuit (number of primary inputs, primary outputs, cells, and test patterns).

TABLE II. Results of Inter-Cell Logic Diagnosis

Circuit	#PIs	#POs	#Cells	#TP	#aSC
c880	60	26	383	34	2
c1355	41	32	938	85	3
c2670	233	140	945	60	3
c3540	50	22	1504	131	2
c5315	178	123	2228	75	2
c7552	207	108	3417	83	4

For generating training data, we used the flow shown in Fig. 1. This characterization phase of the flow was done using a commercial tool and STM libraries. For generating new data instances, we performed post-processing of instance tables obtained as shown in Fig. 6. From the training data and each classifier, 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 III. Cell-Aware Static Diagnosis Results – c2670

Class	Det	#SC	LR	SVM	KNN	NB
DC1	Yes	2	DC1=0.11	DC1	DC1=0.5	DC1=1
DC2	Yes	3	DC6=0.11 / DC2=0.09	DC6	DC2=0.5	DC2=0.5
DC3	Yes	2	DC5=0.11 / DC3=0.10	DC3	DC3=0.5	DC3=1
DC4	Yes	7	DC9=0.14 / DC4=0.12	DC9	DC4=0.5	DC4=0.5
DC5	Yes	1	DC5=0.14	DC5	DC5=0.5	DC5=1
DC6	Yes	3	DC6=0.11	DC6	DC6=0.5	DC6=0.5
DC7	Yes	6	DC7=0.16	DC7	DC7=0.5	DC7=1
DC8	Yes	2	DC8=0.16	DC8	DC8=0.5	DC8=1
DC9	Yes	7	DC9=0.14	DC9	DC9=0.5	DC9=0.5

Table III illustrates the results obtained by the proposed approach for a defect injection campaign in a two-input AND gate of circuit c2670 (with 54 open and short defects grouped into 9 defect classes). The first column lists the various defect classes. The second column indicates if the defects of the corresponding class has been detected or not by the initial circuit-level test set. In the case such defects cannot be detected, this means that they have no impact on the gate output and hence cannot be the source of failure. So, they will no longer be considered in our diagnosis process. The third

column shows the number of suspected cells obtained after logic diagnosis. The next four columns list the best defect's class candidate for each learning algorithm with the corresponding probability of being the root cause of failure.

From these results, the first comment is that KNN and NB identify as best candidate the real (injected) defect. This is true for all defect classes. For LR, the real defect is always identified as a candidate, but sometimes (for DC2, DC3 and DC4) in the second or third position. The second comment refers to the probability given to each best candidate. For example, for DC1, the probability given by LR to DC1 to be the best candidate is 0.11. KNN gives a probability of 0.5 (with n-neighbors=2), which is even better. NB gives a probability of 1 to DC1 to be the best candidate, hence do not providing any other candidates with lower probabilities (unlike what is done by LR and KNN). SVM is a non-probabilistic algorithm and gives the right defect class in 7 (over 9) cases. It does not provide any other candidate (inherent property). All these results clearly demonstrate the feasibility, the effectiveness (in terms of resolution) and accuracy of the proposed diagnosis flow.

Using the same characterization data, a comparison with a commercial cell-aware diagnosis tool has been performed. This tool is non-probabilistic and provides the list of all suspects obtained after diagnosis with a ranking and a matching score. Results achieved with the same defect injection campaign in the same gate of circuit c2670 are reported in Table IV. The first three columns are identical to those in Table III. The fourth column gives the number of identified defect candidates. The fifth column shows the ranking of the injected defect (when it is in the list of candidates – NA otherwise) and the matching score. The last column reports the accuracy, i.e. the injected defect is or is not in the list of candidates. From these results, the first comment is that the commercial tool was often unable to provide a ranking among the candidates, thus complicating the decision before PFA. The second (more important) comment is that, in two out nine cases, the injected defect is not in the list of candidates provided by the commercial cell-aware tool (i.e. results are not accurate). Conversely, our technique with LR, KNN and NB **always** provides the right candidate. This proves the superiority of our approach.

TABLE IV. DIAGNOSIS RESULTS WITH A COMMERCIAL TOOL – C2670

Class	Det	#SC	#candidates	Ranking / Matching	Accuracy
DC1	Yes	2	4	No ranking / 100%	Yes
DC2	Yes	3	3	No ranking / 100%	Yes
DC3	Yes	2	4	No ranking / 100%	Yes
DC4	Yes	7	0	NA / 100%	No
DC5	Yes	1	3	No ranking / 100%	Yes
DC6	Yes	3	3	No ranking / 100%	Yes
DC7	Yes	6	1	1 / 100%	Yes
DC8	Yes	2	1	1 / 100%	Yes
DC9	Yes	7	0	NA / 100%	No

Tables V and VI summarize the results obtained on a set of ISCAS'85 benchmark circuits. Table V is about accuracy and reports, for each learning algorithm, the percentage of cases in which the injected defect was reported in the list of suspects provided by the algorithm. As can be seen, **the diagnosis accuracy achieved with our technique when using LR, KNN and NB algorithms is always 100%**. Results obtained with SVM are less accurate as this is a non-probabilistic algorithm that gives only one suspect, which is sometimes not the right one. The last column in Table V shows the accuracy obtained for each circuit with the commercial cell-aware diagnosis tool. As can be seen, for 4 out of 6 circuits, the commercial tool is unable to achieve 100% accuracy.

TABLE V. OVERALL DIAGNOSIS RESULTS - ACCURACY

Circuit	ACCURACY				
	LR	SVM	KNN	NB	Com. Tool
c880	100%	100%	100%	100%	100%
c1355	100%	77%	100%	100%	96%
c2670	100%	77%	100%	100%	84%
c3540	100%	62%	100%	100%	100%
c5315	100%	88.8%	100%	100%	97%
c7552	100%	77%	100%	100%	90%

Table VI is about resolution and gives, for each learning algorithm and for each circuit considering all injection campaigns, the average number of suspects reported by the proposed method and the commercial tool respectively. Note that in this case, only the cell in which the defect was injected has been considered, the objective being to select the best algorithm for further development of the proposed approach. The same assumption has been done for the resolution given by the commercial tool. We can see that LR always gives 9 candidates (with various probabilities), which corresponds to the number of defect classes of each suspected cell. Similarly, KNN always gives two candidates since *n-neighbors* is initially set to 2. Note that in these experiments, n-neighbors=2 was enough to get a 100% accuracy in all cases. Increasing the value of n-neighbors would just increase (uselessly) the number of defect candidates provided by the learning algorithm. Decreasing n-neighbors (to 1) would no longer lead to 100% accuracy in all experiments. Determining the value of *n-neighbors* a-priori is therefore the main difficult task when using KNN. Finally, only NB is able to provide the best resolution with the highest accuracy, thus definitively surpassing the commercial tool. NB was therefore selected for further development of the cell-aware diagnosis approach.

TABLE VI. OVERALL DIAGNOSIS RESULTS - RESOLUTION

Circuit	RESOLUTION				
	LR	SVM	KNN	NB	Com. Tool
c880	9	1	2	1.15	2.01
c1355	9	1	2	1.1	2.68
c2670	9	1	2	1.2	2.57
c3540	9	1	2	2.45	2.23
c5315	9	1	2	1.08	2.04
c7552	9	1	2	1.25	2.6

The CPU time taken by the proposed diagnosis flow to provide a list of good 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 slight manner. In fact, 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. Results of Cell-Aware Dynamic Defect Diagnosis

We conducted experiments on ITC'99 benchmarks circuits synthesized using a 28nm FDSOI technology from STM. Circuits were synthesized in a full scan manner by using a

commercial tool. A cell-aware commercial ATPG tool was used to generate transition test patterns for each circuit by targeting dynamic cell-internal defects in addition to gate-level transition faults. Test patterns were generated to reach maximum transition fault coverage. From each circuit and the corresponding test set, we simulated the behavior of the tester by performing a defect injection campaign (about 600 random injections per circuit) into a number of randomly selected cells and collecting test information to build the tester datalog. For the defect injection campaign, we considered each transistor of the selected cells and we targeted all possible dynamic (i.e. resistive open and short) 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 a list of suspected cells ranked according to their score to be the source of failure. For all experiments, the list of suspected cells contained the cell in which the defect was injected. The average number of suspected cells (#aSC) for each circuit is listed in Table VII, together with information about each circuit (number of primary inputs, primary outputs, cells, scan flip-flops, transition test patterns, and transition fault coverage in %).

TABLE VII. RESULTS OF LOGIC DIAGNOSIS

Circuit	#PIs	#POs	#Cells	#SFF	#TP	TFC	#aSC
b15	41	37	2465	416	2546	88.38	1.26
b17	42	38	7960	1314	840	91.10	1.83
b18	37	35	3238	215	4155	93.46	1.5
b19	37	23	6337	430	929	93.79	1.34
b20	37	23	6733	430	5133	93.76	1.55
b22	37	23	3218	215	4031	93.78	1.73

For generating training data, we used the flow shown in Fig. 1. This characterization phase of the flow was done using a commercial tool and STM libraries. For generating new data instances, we performed post-processing of instance tables obtained as shown in Fig. 6. From the training data and the Bayesian inference 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 VIII. CELL-AWARE DYNAMIC DIAGNOSIS RESULTS – B19

Defect	#SC	Proposed (SG=A)	Proposed (SG=B)	Cell-aware tool
D61	1 (A)	D64/**D61**=0.5		A=**D61**
D62	2 (A&B)	D81/**D62**=0.5	D55/D53=0.5	0
D63	1 (A)	**D63**/D82=0.5		A=**D63**
D64	2 (A&B)	D61/**D64**=0.5	D55	A=D82/**D64** B=D50
D66	2 (A&B)	**D66**	D51/D50=0.5	A=**D66** B=D52
D67	1 (A)	**D67**		A=**D67**
D69	2 (A&B)	**D69**	D55	A=**D69** B=D55
D81	1 (A)	**D81**/D63=0.5		0
D82	2 (A&B)	**D82**/D64=0.5	D55	A=**D82**/D64 B=D50

Table VIII illustrates results obtained for a defect injection campaign in an AndOr cell of circuit b19 that contains 90 potential defects including resistive and non-resistive opens and shorts. Equivalent defects are grouped into 24 defect classes, among which 9 of them are dynamic, with defect size (in Ω) set to default values provided by HSpice. The first column lists the various injected dynamic defects. The second column shows the number of suspected cells (#SC) obtained after logic diagnosis. Note that in this case study, the defect is always injected in the cell called A. The next two columns list the best defect candidates reported by the Bayesian classification with the corresponding probability of being the root cause of failure. Each column reports the defect candidates provided after applying the proposed method successively on each suspected cell A and B (when two suspected cells exist). The last column reports the defect candidates provided by a commercial cell-aware diagnosis tool using the same characterization data. This tool is non-probabilistic and provides the list of all suspects obtained after diagnosis with a ranking and a matching score.

From these results, the first comment is that the real (injected) defect is **always** identified by the proposed diagnosis approach. Sometimes, it is the only candidate and has a probability of 1 (e.g. D67) to be the best candidate. Sometimes, it is reported with another candidate in suspected cell A (e.g. D61), hence with a probability of 0.5. When two cells are suspected (e.g. D66), 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 cell-aware diagnosis tool is not always able to report the injected defect as candidate. This is the case for D62 and D81, for which the number of reported candidates is 0. **This is the most important observation from these results**, which demonstrates that in terms of accuracy, our proposed solution is 100% efficient, which is not the case of the commercial cell-aware diagnosis tool that sometimes can provide inaccurate results. The second comment is about resolution. In this example, not fully representative, we can observe that in some cases (e.g. D66, D67, D69, D82), our method provides results with the same resolution than what can be obtained with the cell-aware diagnosis tool. In some other (e.g. D61, D63), the resolution is a bit lower with our solution, but the difference is really small (3 instead of 2 candidates).

TABLE IX. OVERALL DIAGNOSIS RESULTS

Circuit	Accuracy		Resolution	
	Bayesian	Com. Tool	Bayesian	Com. Tool
b15	100%	100%	2.220	2.454
b17	100%	100%	5.636	5.242
b18	100%	97.89%	4.842	4.882
b19	100%	95.89%	2.405	1.807
b20	100%	95.83%	2.603	2.373
b22	100%	100%	5.280	5.114

Table IX summarizes the results obtained on a set of ITC'99 benchmark circuits. The first part of the table is about accuracy and gives, for each circuit, the percentage of cases in which the injected defect was reported in the list of suspects provided by the Bayesian diagnosis technique and the commercial cell-aware diagnosis tool respectively. As can be seen, for 3 out of 6 circuits, the commercial tool is unable to achieve 100% (achieved with our technique). The second part of the table is about resolution and gives, for each circuit and considering all injection campaigns, the average number of suspects reported by the proposed method and the commercial tool respectively. As can be seen, the resolutions achieved with both methods are very close. So, overall, these results confirm the superiority of our approach in terms of accuracy.

VII. Conclusion, Discussion and Future Work

In this paper, we have proposed a new learning-guided approach for diagnosis of intra-cell defects that may occur in customer returns. In the first part of the paper, we have considered several supervised learning algorithms and dealt with static defects modeled by stuck-at faults. In the second part of the paper, we have extended the previous work by dealing with more sophisticated (i.e. dynamic) defects. This time, we used a Gaussian NB classifier for predicting the nature (likelihood to be a good candidate) of each new data instance that has to be evaluated during the diagnosis process. In both case, we have compared our results with those obtained with a commercial cell-aware diagnosis tool to demonstrate the efficiency of the proposed approach in terms of accuracy and resolution.

The above results show 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). This will be even truer when multiple defect sizes and test conditions will be used. In these cases, 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 STM library), our framework will still be easily and time-efficiently applicable.

In this work, the single defect assumption has been considered in our experiments. However, this assumption is not necessary as the proposed approach is able to manage situations where multiple defects have occurred, provided that those defects are not in the same cell. This significant feature (also valid for commercial tools) comes from the fact that our diagnosis flow considers all suspected cells one at a time, and then incrementally constructs a list of suspects identified in each of these cells. Similarly, no ranking among the suspected cells provided after logic diagnosis has been considered in our experiments. As a consequence, our flow has reported all defect candidates coming from all suspected cells without any ranking. In case a ranking of suspected cells is done after logic diagnosis (usually the case with commercial tools), a similar ranking among defect candidates can be done in our flow. In-field failure mechanisms related to premature aging, such as NBTI or HCI, essentially lead to resistive opens and shorts. These mechanisms, that have to be considered in the context of customer returns, can now be appropriately taken into account in our cell-aware dynamic diagnosis flow.

The next step of this work will be to *fully combine the two proposed approaches described in this paper in order to get a comprehensive diagnosis method able to deal with all types of defects*, i.e. static and dynamic, that may occur in customer returns. Thought it may look simple, as just a combination of the two previous methods, proposing such a comprehensive method raised new problems and imposed setting up a new framework with specific rules to achieve the same level of efficiency in terms of diagnosis accuracy and resolution. Further developments have also to be done to address several missing aspects. In our study, all injected defects for evaluation purposes were present in the training dataset. In real silicon, especially for customer returns, actual defect behavior may not perfectly match the fault models that are used to train the classifier. Further work will be dedicated to see how well the proposed method works in that scenario. Moreover, layout information has to be used to refine the list of defects that are considered during training data preparation. By this way, only realistic defects will be assumed during the whole process, thus increasing diagnosis efficiency. Then, we need to consider multiple sensitization conditions that may occur due to i) the fact that most of industrial designs are not 100% full scan and hence require multiple capture cycles during test, ii) the presence of a mix of leading and trailing edge triggered flops in a design, and ii) the fact that the clock signal feeds into the system logic under test [30]. Another point is that unique test conditions have been assumed in our experiments. In the context of mission mode failure diagnosis, multiple test conditions with various PVT corners also need to be considered. Finally, we need to compare our results with those obtained with an industrial in-house tool [15], and perform experiments on customer returns provided by STM.

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, L. Winemberg, and M.S. Abadir, "Understanding Customer Returns From A Test Perspective," in *Proc. IEEE VLSI Test Symposium*, 2011, pp. 2-7.

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

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

[4] 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.

[5] 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, March 2010.

[6] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and O. Riewer, "A Comprehensive System-on-Chip Logic Diagnosis," in *Proc. IEEE Asian Test Symposium*, 2010, pp. 237-242.

[7] L. M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using the Single Location At a Time (SLAT)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 91-101, Jan. 2004.

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

[9] S. Venkataraman and S. B. Drummonds, "Poirot: Applications of a Logic Fault Diagnosis Tool," *IEEE Design & Test of Computers*, vol. 18, no. 1, pp. 19-30, Feb. 2001.

[10] 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*, 2006, pp.1-9.

[11] A. Ney, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, and M. Bastian, "A History-Based Diagnosis Technique for Static and Dynamic Faults in SRAMs," in *Proc. IEEE International Test Conference*, 2008, paper 3.2.
[12] K. Huang, H. Stratigopoulos, and S. Mir, "Fault Diagnosis of Analog Circuits Based on Machine Learning," in *Proc. IEEE/ACM Design, Automation & Test in Europe*, 2010, pp. 1761–1766.
[13] C. Zhang, Y. He, L. Yuan, and S. Xiang, "Analog Circuit Incipient Fault Diagnosis Method Using DBN Based Features Extraction," *IEEE Access*, vol. 6, pp. 23053-23064, April 2018.
[14] A. Ladhar and M. Masmoudi, "Efficient and Accurate Method for Intra-gate Defect Diagnoses in Nanometer Technology and Volume Data," in *Proc. IEEE/ACM Design, Automation & Test in Europe*, 2009, pp. 988-993.
[15] Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, and E. Auvray, "Effect-Cause Intra-cell Diagnosis at Transistor Level," in *Proc. IEEE International Symposium on Quality Electronic Design*, 2013, pp. 460-467.
[16] J. E. Nelson, W. C. Tam, and R. D. Blanton, "Automatic Classification of Bridge Defects," in *Proc. IEEE International Test C*onference, 2010, pp. 1–10.
[17] X. Ren, M. Martin, and R. D. Blanton, "Improving Accuracy of On-Chip Diagnosis via Incremental Learning," in *Proc. IEEE VLSI Test Symposium*, 2015, pp. 1–6.
[18] 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*, 2015, pp. 1–10.
[19] 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.
[20] H. Wang, O. Poku, X. Yu, S. Liu, I. Komara, and R. Blanton, "Test-Data Volume Optimization for Diagnosis," in *Proc. ACM/IEEE Design Automation Conference,* 2012, pp. 567.
[21] Y. Xue, O. Poku, X. Li, and R. D. Blanton, "PADRE: Physically- Aware Diagnostic Resolution Enhancement," in *Proc. IEEE International Test Conference*, 2013.
[22] Z. Gao, S. Malagi, E.J. Marinissen, J. Swenton, and J. Huisken, "Defect-Location Identification for Cell-Aware Test," in *Proc. IEEE Latin-American Test Symposium*, 2019.
[23] https://machinelearningmastery.com/
[24] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica*, vol. 31, no. 3, pp. 249-268, 2007.
[25] http://scikit-learn.org/stable/user_guide.html
[26] C. Bielza, G. Li, and P. Larranaga, "Multi-Dimensional Classification with Bayesian Networks," *International Journal of Approximate Reasoning*, vol. 52, no. 6, pp. 705-727, Sept. 2011.
[27] F. Liu, P. K. Nikolov, and S. Ozev, "Parametric Fault Diagnosis for Analog Circuits Using a Bayesian Framework," in *Proc. IEEE VLSI Test Symposium*, 2006, pp. 277-283.
[28] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty, "Board-Level Fault Diagnosis using Bayesian Inference," in *Proc. IEEE VLSI Test Symposium*, 2010.
[29] M. Martinez and L.E. Sucar, "Learning an Optimal Naïve Bayes Classifier," in *Proc. International Conference on Pattern Recognition*, vol. 3, pp. 1236–1239, 2006.
[30] M. Sharma, W.T. Cheng, T.P Tai, Y.S Cheng, W. Hsu, C. Liu, S.M. Reddy, and A. Mann, "Faster Defect Localization in Nanometer Technology based on Defective Cell Diagnosis," in *Proc. IEEE International Test Conference*, 2007, Paper 15.3.

Safa Mhamdi received her Master degree in Microelectronics from the University of Montpellier, France, in 2017. She is currently a PhD student at the University of Montpellier, and works in the Microelectronics Department of the LIRMM (Laboratory of Informatics, Robotics and Microelectronics of Montpellier – France). Her main focus of research lies in Test and Diagnosis of Digital circuits and systems and Reliability

Patrick Girard received a Ph.D. degree in Microelectronics from the University of Montpellier, France, in 1992. He is currently Research Director at CNRS (French National Center for Scientific Research) and works in the Microelectronics Department of the Laboratory of Computer Science, Robotics and Microelectronics of Montpellier (LIRMM) - France. He is Director of the International Associated Laboratory « LAFISI » (French-Italian Research Laboratory on Hardware-Software Integrated Systems). He is deputy director of the French scientific network dedicated to research in the fields of System-on-Chip, Embedded Systems and Connected Objects (SOC2). His research interests include all aspects of digital and memory testing, with emphasis on critical constraints such as timing and power. Robust design of neuromorphic circuits as well as machine learning for test and diagnosis are also part of his new research activities. He has supervised 37 PhD dissertations and has published 7 books or book chapters, 75 journal papers, and more than 250 conference and symposium papers on these fields. Patrick Girard is a Fellow of the IEEE.

Arnaud Virazel received the Ph.D. degree in Microelectronics from the University of Montpellier, France, in 2001. He is currently Associate Professor at the University of Montpellier, and works in the Microelectronics Department of the LIRMM (Laboratory of Informatics, Robotics and Microelectronics of Montpellier - France) where he is leading the TEST ("Test and dEpendability of microelectronic integrated SysTems") team. He has published 3 books or book chapters, 40 journal papers, and more than 140 conference and symposium papers spanning diverse disciplines, including DfT, BIST, diagnosis, reliability, delay testing, power-aware testing and memory testing. He is deputy head of the electrical engineering Master program (about 200 students) in charge of the first year and of the "Integrated Electronic Systems" specialization at the University of Montpellier.

Alberto Bosio received the PhD in Computer Engineering from the Politecnico di Torino, Italy in 2006. From 2007 to 2018, he was an Associate Professor at LIRMM - University of Montpellier in France. Hi is now a Full Professor at the INL – Ecole Centrale de Lyon, France. His research interests include Approximate Computing, In-Memory Computing, Test and Diagnosis of Digital circuits and systems and Reliability. He has co-authored 1 book, 3 patents, 35 journals, and over 120 conference papers. He is the chair of the ETTTC. He is a member of the IEEE.

Eric Faehn received an engineer degree in microelectronic and radio electricity from the "Ecole Nationale Supérieure d'Electronique et de Radioélectricité de Grenoble" in parallel with a Master of Research degree in integrated system design from the "Université Joseph Fourier de Grenoble" in 2004. Employed after his studies by STMicroelectronics as design engineer and customer support, he first developed eDRAM controllers. In 2008, he started to work on eSRAM and oversaw optimal test algorithms and BIST architecture definition. In 2010, he focused his activity on memory diagnosis and worked internally and with external tool suppliers to enhance and automate diagnosis capabilities and precision.

Aymen Ladhar received the PhD degree in Electrical Engineering from the University of Sfax, Tunisia, in 2010. He is currently a senior test & yield engineer at STMicroelectronics Crolles, France. His research interests include VLSI testing, fault diagnosis, layout analysis, defect extraction and simulation.