Machine Learning Support for Cell-aware Diagnosis

Aymen Ladhar, Arnaud Virazel

Chapter 5
Machine Learning Support for Cell-aware Diagnosis

Aymen Ladhar, Arnaud Virazel

Abstract

This chapter will provide an overview of the various machine learning approaches and techniques proposed to support cell-aware generation, test, and diagnosis. The chapter will focus on the generation of the cell-aware models and their usage for diagnosis. After some backgrounds on conventional approaches to generate and diagnose cell-aware defects, the chapter will present a learning-based solution to generate cell-aware models. Then, the chapter will present a ML-based cell-aware diagnosis technique. Effectiveness of existing techniques will be shown through industrial case studies and corresponding diagnosis results in terms of accuracy and resolution. The chapter will conclude by a discussion on the future directions in this field.

1 Introduction

The usage of Cell-Aware (CA) methodology becomes mandatory for semiconductor industry, especially for designs with the highest product quality. This is because fault models like stuck-at, transition, as well as layout-aware fault models are not accurate enough to achieve very low Defective Part Per Million (DPPM) rates and to resolve the underlying systematic yield detractors for a successful and fast yield ramp-up. Previous works on CA defect test and diagnosis can be classified into two categories. Techniques in the first category extend the application of logic algorithms to deal with transistor defects [1-2]. The main weakness of these techniques is the quality of the logic fault models that do not properly describe the behavior of potential transistor defects. These methods are limited to cell-level diagnosis and cannot be used during test pattern generation. The second category of cell-internal defect test and diagnosis techniques relies on the realistic assumption that the excitation of a defect inside a cell is highly correlated with the logic values at the input pins of the cell [3-4]. For this category, a cell-internal-fault dictionary or *CA model* (also referred to as *CA fault model* or *CA test model in the literature*), describing

Aymen Ladhar
STMicroelectronics, 38920 Crolles, FR, e-mail: aymen.ladhar@st.com
Arnaud Virazel
LIRMM, University of Montpellier / CNRS, 34095, Montpellier, FR
e-mail: arnaud.virazel@lirmm.fr

the detection conditions of each potential defect affecting a cell, is used [5-6]. These techniques are more efficient and can be used to guide the test pattern generation and CA diagnosis phases. However, the main limitation of these techniques is the generation effort needed to characterize all the standard cells per technology in terms of run time, number of SPICE simulator licenses, CPU requirements and disk usage. The second limitation of this technique is the usage of simple fault models to describe silicon failures, which is not always possible for actual silicon failures. In addition, the assumption that the excitation of a defect inside a cell is highly correlated with the logic values at the input pins of the cell is not always correct mainly for unmodeled cell internal defect.

With the fast development and vast application of Machine Learning (ML) in recent years, ML-based techniques have been shown to be quite valuable for diagnosis purpose, especially in the volume diagnosis scenario [5-6]. In this chapter, the the usage of machine-learning algorithms is extended to generate the cell-aware models in a first step, then to diagnose cell internal defects.

The remainder of this chapter is organized as follows. Section 2 gives some background on conventional cell-aware generation, test, and diagnosis. Section 3 explains the machine learning cell-aware flow. Section 4 summarizes a machine learning technique to diagnose cell-aware defect. Section 5 presents some industrial case studies performed with latest ML-based diagnosis techniques. Section 6 concludes the chapter and draws some conclusions.

2 Background on Conventional Cell-Aware Generation, Test and Diagnosis

To compete in the fast-growing market of automotive ICs as well as 3D transistor era, semiconductor industries need to address new challenges across the entire design flow. In addition, the requirement to be compliant with the highest standards like the ISO 26262 goal of zero Defective Parts Per Million (DPPM), has an impact on the test and diagnosis phase of nowadays silicon failures [21]. Cell internal defects are one of most critical defects that should be targeted since they are increasingly occurring in latest technology nodes. In addition, this defect type is hard to be targeted with conventional ATPG solution and fault models. Test and diagnosis of cell internal defects is possible thanks to the realistic assumption that the excitation of a defect inside a cell is highly correlated with the logic values at the input pins of the cell [3]. To perform CA test and diagnosis, a CA model is needed. It is obtained through the characterization of each standard cell library with regards to all possible cell-internal defects. These defects can be either transistor defects or inter-transistor defects like open and shorts [22-25]. Electrical simulations are performed to generate CA models for each cell in the library. These models include information about the behavior of each defect within the cell with regards to the stimuli applied at the cell inputs.

One bottleneck of the CA flow deployment in industry is the generation effort in terms of run time and flow complexity. Indeed, it requires extensive computational efforts to characterize all standard cells of a library [7]. Fig 5.1 represents a typical CA model generation flow. It starts with a SPICE netlist representation of a standard cell which is usually derived from a layout description, e.g., a GDSII file. The DSPF (Detailed Spice Parasitic Format) contains information about potential shorts and open defect, as well as their coordinates within the cell. This DSPF cell netlist is then used by an electrical simulator to simulate each potential cell internal defect against an exhaustive set of stimuli containing static and dynamic patterns. Once the simulation process is completed, all cell-internal defects are classified into defect equivalence classes with their detection information (required input values for each defect within each cell) and are synthetized into a CA model. Two tables or matrix exist in each CA model, the first one targets static defects whereas the second one targets dynamic defects. As standard cells may have several inputs, and thousands of cells with different complexities are used for a given technology, the generation time of CA models for complete standard cell libraries of a given technology may reach up to several months, thus drastically increasing the library characterization process cost.

Fig. 5.1 Conventional cell-aware model generation flow

As mentioned, cell-aware models can be either used for test purpose or for fault diagnosis of silicon failures:

- *ATPG usage*: the ATPG engine identifies for each cell in the Circuit Under Test (CUT) the minimum set of stimuli targeting the entire cell internal defects, then it generates test patterns exercising this test stimuli at the input pins of the cell under test and ensures the fault propagation to an observation point.
- *Fault diagnosis usage*: the diagnostic tool extracts the failing and passing logic values at the input pins of the defective cell in the Circuit Under Diagnosis (CUD). This information is then matched with the CA model of the defective cell to identify the suspect internal defect.

3 Learning-Based Cell-Aware Model Generation

The reason behind the ML usage for defective cell characterization is the result of several observations made while performing electrical simulation of several cell internal defects on different standard cell libraries and technology nodes:

- Several cell internal defects are independent of the technology and transistor size, and their defective behavior depends mainly on the location of the defect and test stimuli applied on the cell under characterization [8-9].
- For the same function, cell schematics are usually quite similar whatever is the technology node.
- Detection tables for static and dynamic defects, in the form of binary matrices describing the detection patterns for each cell-internal defect, are ML friendly.
- CA models may change with respect to test conditions and Process Voltage Temperature (PVT) corners. In fact, CA model generation for the same cell with different test conditions may exhibit slight differences. Few defects can be of different types (i.e., static, or dynamic) or may have different detection patterns. Since CA models are generated for specific test conditions and can be used with different ones, it may lead to inaccurate characterization. This inaccuracy is usually allowed in industry since it is marginal. This indicates that one can also tolerate few error percentages in our ML-based prediction.
- Very simple CA models are used to emulate short and open defects, for which resistance values are often identical for all technologies.
- A large database of CA models is usually available and can be used to train a ML algorithm.

All these above-mentioned reasons intuitively revealed that CA model generation through ML could be feasible. However, the first tricky task is to be able to describe cell transistor netlist as well as corresponding cell-internal defects in a uniform (standardized) manner, so that a ML algorithm can learn and infer from data irrespective of their incoming library and technology. Indeed, similar cells (e.g., cells with same logic function, same number of inputs and same number of transistors) may be described differently in transistor-level (SPICE) netlists of various libraries (e.g., a transistor label does not always correspond to the same transistor in two similar cells coming from two different libraries), and it is therefore mandatory to standardize the description of cells and corresponding defects for the ML-based defect characterization methodology. Heuristic solutions developed to this purpose are described in Sections 3.1 and 3.2. The second challenging task is to find a solution to represent all these information / input data so that they can be ML friendly. A matrix description of cells and corresponding defects can be chosen to this purpose. Section 3.3 describes how to adapt the ML method to deal with sequential cells.

3.1 Generation of Training and New Data

The learning-based defect characterization methodology proposed in [10] is used to predict the behavior of a cell affected by cell internal defects, hence avoiding costly and heavy electrical defect simulations. The proposed flow is presented in Fig. 5.2. It is 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.

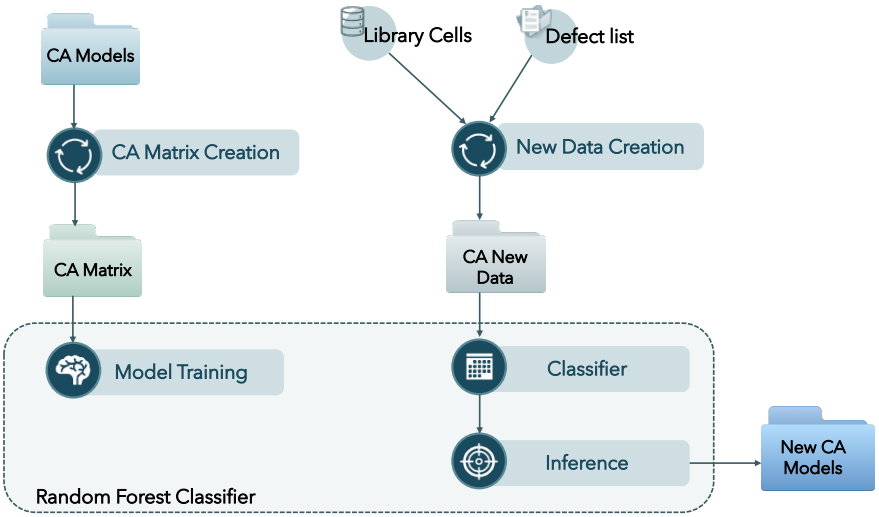


Fig. 5.2 Generic view of the ML-based defect characterization flow

Training data are made of various and numerous CA models formerly generated by relying to brute-force electrical defect simulations. For each cell in a library, the CA model is transformed into a so-called *CA-matrix* and filled in with meaningful information.

An overview of the CA-matrix flow is described in Fig. 5.3. It begins by rewriting the CA model in a ML friendly description. Then, it categorizes the activation conditions of each transistor with respect to input stimuli. Once the activation conditions for each transistor have been identified, transistors are renamed in a uniform way. This is a critical step in the proposed flow as it allows the usage of the training data across different libraries and technologies. Finally, the CA-matrix is created with the above information.

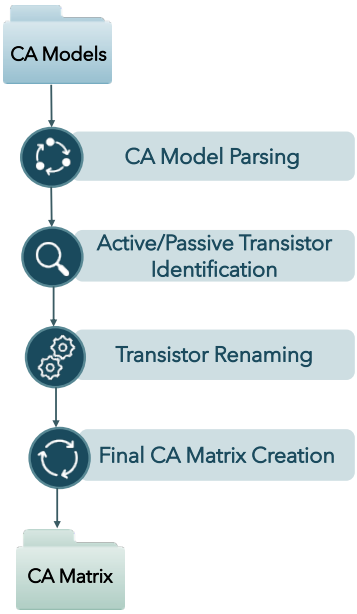


Fig. 5.3 Generic view of the CA-matrix creation flow

Table 5.1 shows an example of a training dataset for a NAND2 cell. It is composed of four types of information:

- *Cell patterns and responses*. This information represents an extended representation of a NAND2 truth table (A, B: inputs, Z: output). As it can be seen, the test pattern sequence provides all the possible input stimuli that can be applied to the cell. It takes into consideration all the possible transitions that can be applied on the inputs of this cell. These stimuli must also be efficient to detect sequence-depending defects like stuck-open defects. For this reason, a four-valued logic algebra made of 0, 1, R and F is used to represent input stimuli in the CA-matrix. R (resp. F) represents a Rising (resp. Falling) transition from 0 to 1 (resp. from 1 to 0).
- *Active / Passive Transistor Identification*. This indicates the activation conditions of each transistor in the cell schematic. Each transistor can be in the following state: active, passive, switching to active state, switching to passive state.
- *Defect description*. This gives information about defect locations in the cell transistor schematic. At least one logic value is used to indicates the name and the transistor port impacted by the defect.
- *Defect detection*. This is the class of the data sample (the output of ML classifier). A value '1' ('0') means that the defect is detected (undetected) by the input pattern.

The first three types of information constitute the inputs of the ML algorithm.

TABLE 5.1: EXAMPLE OF TRAINING DATASET FOR A NAND2 CELL

Cell inputs & responses			Transistor switching activity				Defect description				About defect		Defect detection
A	B	Z	N0	N1	P0	...	N1_D	N1_G	N1_S	...	name	type	fZ
0	0	1	0	0	1	...	0	0	0	...	free	free	0
0	1	1	0	1	1	...	0	0	0	...	free	free	0
0	F	1	0	F	1	...	0	0	0	...	free	free	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	0	1	1	...	1	0	1	...	D15	short	1
1	1	0	1	1	0	...	1	0	1	...	D15	short	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

New data represent the cells to be characterized and are obtained for each standard cell from the cell description, corresponding list of defects and cell patterns. The format of a new data instance is like that of the training data, except that the class (label) of the new data instance is missing. The ML classifier is used to predict that class. As for training data, new data are grouped together according to their number of cell inputs and transistors, so that inference can be done at the same time for cells with the same number of inputs and transistors.

As highlighted in Fig. 5.2, a Random Forest Classifier is used for predicting the class of each new data instance. This choice comes from the results obtained after testing several learning algorithms (k-NN, Support Vector Machine, Random Forest, Linear, Ridge, etc.) and observing their inference accuracies. So, the first main step of our CA diagnosis flow consists in generating a Random Forest model and to train it by using the training dataset. A Random Forest Classifier is composed of several Decision Tree Classifiers, which are models predicting class of samples by applying simple decision rules. During training, a Decision Tree tries to classify data samples and its decision rules are modified until it reaches a given quality criterion. Then, the Forest averages the responses of all Trees and outputs the class of the data sample. The second main step consists in using the Random Forest Classifier to make prediction (or inference) when a new data instance must be evaluated. Prediction for a new data instance amounts to answer to the question: “Does this stimulus detects this defect affecting this cell?”. Answering to this question allows obtaining a new CA model for a given standard cell

3.2 Cell and Defect Representation in the Cell-Aware Matrix

This section describes the used method to represent a transistor-level (SPICE) netlist in a CA-matrix, which is a standardized and ML friendly description. This representation must be accurate enough to clearly identify each transistor and each net of the cell transistor schematic. Furthermore, this description must also be able to correlate each transistor to its sensitization patterns and to report the output

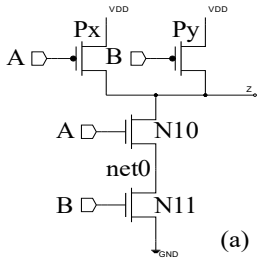
response for each pattern. Therefore, the cell description process needs several successive operations that are detailed below. Note that this process is applied to all cells in a library to be characterized.

3.2.1 Identification of Active and Passive Transistors

The first step consists in identifying active and passive transistors in the cell netlist with respect to an input stimulus. This information is used to represent the transistor netlist in the matrix format. To this end, a single defect-free (golden) electrical stimulation of each cell to be characterized is performed to identify active and passive transistors for each input stimulus (test pattern) and to measure the cell response on the output pin. A NMOS (resp. PMOS) transistor is considered active if a logic-1 (resp. logic-0) value is measured on its gate terminal. Similarly, a NMOS (resp. PMOS) transistor is considered passive if a logic-0 (resp. logic-1) value measured on its gate terminal. With this information, each cell pattern can be associated to the list of active transistors in the cell. After this step, the CA-matrix contains the columns:

- Cell inputs & responses columns. They contain all input stimuli (test patterns) that can be applied to the cell, and the corresponding responses.
- Transistor switching activity columns. They contain four possible values indicating if the transistor is active (1), passive (0), switching from an active state to a passive one (F) or switching from a passive state to an active one (R). Since PMOS and NMOS transistors are activated in opposite way, the '-' character is used before the PMOS values.

Fig. 5.4 represents an example of a NAND2 cell with its CA-matrix representation. In the partial representation of the CA-matrix of the cell shown in Fig. 5.4.b, columns A and B list all the possible input stimuli for this cell. These columns also define the length of the CA-matrix, which is equal to $2^n + 2^n.(2^n-1)$, n being the number of cell input pins (2^n is the number of static stimuli, $2^n.(2^n-1)$ is the number of dynamic stimuli). For each stimulus, active and passive information about each transistor of the cell is entered in the CA-matrix. For example, AB=00 leads to two active PMOS transistors (Px and Py) and two passive NMOS transistors N10 and N11).



A	B	Px	Py	N10	N11
0	0	-1	-1	0	0
0	1	-1	0	0	1
1	0	0	-1	1	0
...	...	...	...	...	...
R	R	-F	-F	R	R

(b)

Fig. 5.4 Example standard cell NAND2: (a) cell transistor schematic and (b) partial CA-matrix representation

This matrix description of the cell netlist is clearly dependent on the transistor names and the order they are specified in the SPICE netlist. In fact, two similar cell schematics may have different transistor naming and the order of transistors in the SPICE netlist may differ from cell to cell and from one technology to another. Without an accurate naming convention of each cell transistor in the CA-matrix, any ML algorithm will fail to predict the behavior of the cell in presence of a defect. To prevent this problem, a second step in the cell description process is needed. This step consists in renaming all cell transistors independently of their initial names and order in the input SPICE netlist. The algorithm built to this objective is explained in the next subsection.

3.2.2 Renaming of Transistors

In the CA-matrix, two cells with the same *transistor structure* will have the same transistor names irrespective of their incoming library and technology. A *transistor structure* is a virtual SPICE netlist without specification of the connections between transistor gates, i.e., only source and drain connections between transistors are listed. The transistor-renaming algorithm consists of the following two steps:

- *Determination of branch equations*: Since the transistor gate connections are not considered, the transistor structure is composed of one or more branches. A branch is a group of transistors connected by their drain and source terminals. The entry of each branch is the set of transistor gates, and the exit is the connection net between the NMOS and PMOS transistors. A branch is connected to a power and/or a ground net. A branch equation is a Boolean-like equation describing how the transistors of the branch are connected, using Boolean-and (symbolized by '&') for serial transistors or serial groups of transistors, and Boolean-or (symbolized by '|') for parallel transistors or parallel groups of transistors.

  In Fig. 5.5, the structure is composed of two branches. The two-transistors output-inverter is the simplest branch whose input is net Y and output is net Z. The inverter creates two paths between branch output and power nets, so its branch equation is ($N_{inv}|P_{inv}$). The equation of the second branch (NMOS branch

driving net Y) is ((N0&(N1|N2))|N3). To do not rely on any name present in the SPICE netlist, the branch equations are anonymized, i.e., a NMOS is described by '1n' and a PMOS by '1p'. The anonymized equation of the NMOS branch driving net Y in Fig. 5.5 is therefore ((1n&(1n|1n))|1n).

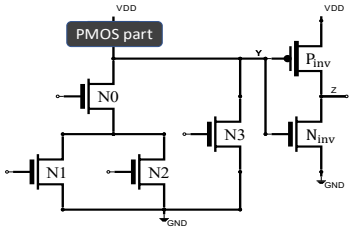


Fig. 5.5 Schematic example

- *Sorting of branch equations*: Once all the branch equations for the considered cell have been determined, they are sorted by using deterministic criteria:
    - *Level of each branch*. It is defined in ascending order with respect to the cell output (level-1 branches drive the cell output, level-2 branches drive the gates of transistors in level-1 branches, and so on and so forth),
    - *Number of transistors in each branch* - in ascending order,
    - *Anonymized branch equation* defined in alphabetical order.

3.2.3 Identification of Parallel Transistors

Identifying branch equations is not enough to rename all transistors of a given cell. The problem comes from parallel transistors. In fact, two or more parallel transistors share the same source and drain, which makes their identification quite difficult. For example, in Fig. 5.5, transistors N1 and N2 can be either presented as “N1|N2" or as "N2|N1". As the order of transistors in each branch will determine how transistors will be renamed, such confusing situation cannot by accepted. A solution consists in sorting transistors inside their branch according to their activity with respect to the input stimuli. The algorithm developed to this purpose proceeds as follows. For each transistor, an *activity value* is computed. For a cell with n inputs, the activity value is a 2^n-bit integer that represents the accumulative activation state of a transistor for all possible stimuli applied to the cell. The input stimuli range from (0,0…,0) to (1,1…,1). For each of these stimuli, the transistor is either active (1) or passive (0). The activity value is defined as a binary number, whose MSB is the activity of the transistor under input stimulus (0,0…,0) and LSB is the activity of the transistor under input stimulus (1,1…,1), with a bit significance decreasing for increasing binary value of input stimulus.

To compute the activity value, one need to know whether the transistor is active or passive for each input stimulus. This information is already presents in the CA-

matrix as described in Section III.A. To illustrate this process, activity values for the transistors of the NAND2 cell in Fig. 5.4.a are given in Table 5.2.

TABLE 5.2: ACTIVITY VALUES FOR A NAND2 CELL

			old names			
A	B	Comment	Px	Py	N10	N11
0	0	MSB	1	1	0	0
0	1		1	0	0	1
1	0		0	1	1	0
1	1	LSB	0	0	1	1
Activity value			12	10	3	5
			↓ Renaming ↓			
			P1	P0	N0	N1

Finally, transistors of each branch are sorted by increasing activity value to give the final description of the cell in the CA-matrix. For the example in Fig. 5.4, this leads to: first NMOS transistor of the first branch is named N0, second NMOS transistor of the first branch is named N1, first PMOS transistor of the second branch is named P0, and second PMOS transistor of the second branch is named P1.

3.2.4 Defect representation

This section details the cell-internal defects representation in the CA-matrix in a standardized and ML friendly manner. Cell internal defects are classified into:

- *Intra-transistor defects*. These defects affect transistor terminals (source, drain, gate and bulk) and can be either an open defect or a short. To describe these defects, all transistor terminals are listed as a column in the CA-matrix (cf. Table I). For an open defect, a value '1' indicates that this transistor terminal is affected by the defect, '0' otherwise. For a short, a value '1' on two transistor terminals indicates that a short exists between these two terminals, '0' otherwise.
- *Inter-transistor defects*. These defects affect a connection(s) between at least two different transistors. Though these defects are not considered in this work, the matrix representation is flexible enough to represent them. For these defects, the same representation mechanism is used as for intra-transistor defects.

Table 5.3 is an example of defect description in the CA-matrix of the NAND2 cell example presented in Fig. 5.4. Row with red cells describes the intra-transistor short defect between drain and source terminals of transistor P1 (formerly Px). Row with blue cells describes the inter-transistor short defect between P0-source and "net0" (net0 connects N0-source and N1-drain).

TABLE 5.3: DEFECT REPRESENTATION FOR A NAND2 CELL

P0_S	P0_D	P1_S	P1_D		N0_S	...	N1_D	Comment
0	0	1	1		0	...	0	source-drain short on P1
1	0	0	0		1	...	1	net0 & P0-source short

3.3 Support of Sequential Cells

This section describes the different modifications that must be considered during the creation of the CA matrix in case of a sequential cell:

- *Cell inputs and outputs:* For sequential cells it is important to know the stored value inside the cell before applying the test stimuli. To this end, a new column is included in the CA matrix. It includes the output value of the cell before applying the new test stimuli.
- I*dentification of Active, Passive, and pulsing transistors:* One characteristic of sequential cells is the presence of a clock signal, which has an impact on the value applied on each transistor. In fact, clock-signal-controlled transistors can be pulsing (resp. anti-pulsing), which means a 0-1-0 (resp. 1-0-1) sequence appears on the transistor gate terminal during application of the test pattern.
- *Determination of branch equations:* Sequential cells tend to integrate transmission gates to separate the latches from each other and from the inputs. A transmission gate is a transistor configuration acting as a relay that can conduct or block depending on the control signal. It is composed of one PMOS and one NMOS transistors in parallel (i.e., sharing drain and source), and the control signal applied to the gate of the NMOS transistor is the opposite (i.e., NOT-ed) of the signal applied to the gate of the PMOS transistor. A transmission gate directly connects the exit of a branch to the entry of another branch. As such, a transmission gate is considered as an autonomous branch of the transistor structure. The entry of such a branch is the set of transistors gates plus the exit of the previous branch, and the exit is the entry of the following branch. Each time a transmission gate is identified, the branch equation will be considered as a transmission gate and not as a PMOS with a NMOS transistors. The symbol *t* will be used into the branch equation.

4 Advanced Cell-Aware Diagnosis Based on Machine Learning

Although conventional diagnosis techniques such as “Cause-Effect” [11] and “Effect-Cause” [12] can achieve a good resolution, in some cases (e.g., complex cells, complex failure mechanisms) the number of candidates may be too high to allow an efficient PFA (Physical Failure Analysis). This problem will be exacerbated in the future with the advent of very deep submicron (i.e., 5 nm and beyond) technologies. Improving diagnosis efficiency at the transistor level (i.e., CA diagnosis) is therefore mandatory.

Previous works on Cell-Aware (CA) fault diagnosis focusing on logic cells can be classified into three approaches. The first approach converts a transistor-level netlist into an equivalent gate-level netlist by means of complex transformations rules [26]. Then, on the equivalent gate-level netlist, any classical fault diagnosis approach can be applied. The main drawback of this approach is that the set of transformation rules depends on the targeted defect and, thereby, the non-modeled defects may not be diagnosed. The second approach is based on the “Cause-Effect” paradigm [26-27]. The transistor-level netlist of a cell is exploited, in order to inject the targeted defects. Therefore, a defect dictionary is created by transistor-level simulations and the defect signatures of all the defects affecting the cells in the library are stored in this defect dictionary. Then, during fault diagnosis the defect signature of all defects affecting a suspected cell is compared with the observed failures to obtain a list of candidates inside the cell. These approaches can be further classified depending on the “accuracy” of the injected defects and the simulation “precision”. In [27], a large number of defects are simulated at transistor-level using SPICE. For a given defect, different resistance values are simulated, in order to be as accurate as possible. This approach leads to more precise results but it requires a huge simulation time. To reduce the simulation time and the fault dictionary size while keeping a high resolution, authors in [26] propose to exploit layout information, in order to consider only realistic defects. For example, for each cell, only the realistic, potential net bridging defects and via open defects are extracted and then simulated. Then, the identified set of realistic defects is simulated at transistor-level. The third intra-cell fault diagnosis approach is based on the “Effect-Cause” paradigm [28]. All the existing diagnosis techniques depend on the targeted fault models or defects. In [28], the main goal is to achieve a resolution close to the transistor-level. However, instead of explicitly considering defects at transistor-level, the idea is to exploit the knowledge of the faulty behavior induced by the defects.

Unfortunately, CA fault diagnosis resolution is typically far from the ideal due to circuit complexity. 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. This section summarizes the latest developments in the field of CA diagnosis based on supervised learning.

4.1 Preliminaries and Test Scenarios

Several learning-guided solutions for CA diagnosis have been proposed recently in [13-18]. All solutions are based on a Bayesian classification method for accurately identifying defect candidates in combinational standard cells of a customer return. Choosing one solution over another depends on the test scenario (test sequence, test scheme, test conditions) considered during the diagnosis phase and selected according to the types of targeted defects and failure mechanisms.

The test scenarios in [13-18] are sketched in Table 5.4. In [14-15], two **distinct** processes were developed to diagnose static and dynamic defects **separately**. In [14], a basic scan testing scheme used to apply static CA test sequences is considered, so that stuck-at faults plus static intra-cell defects are targeted during diagnosis. In [15], a fast sequential testing scheme used to apply dynamic CA test sequences is considered, so that transition faults plus dynamic intra-cell defects are targeted during diagnosis. *Note that [16] is just a combination of [14] and [15], i.e., two testing schemes, one static and one dynamic, and two CA diagnosis flows, one static and one dynamic, are considered independently*. The main limitation of the solutions in [14-16] is the required a priori knowledge of the type of targeted defects in the CUT. In other words, a test engineer needs to know what type of defects is screening before choosing between [14] or [15].

To deal **concurrently** with all types of defects that may occur and without any a priori knowledge of the targeted defect type, a new implementation of the CA diagnosis flow was proposed in [13-17]. *Note that [13] is a fully extended version of [17]*. Authors assume a test scenario in which **two** test sequences (static and dynamic) are used successively, each one considering a dedicated testing scheme, i.e., basic scan and fast sequential. First, a static CA test sequence generated by a commercial cell-aware ATPG tool is applied to the CUD. 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 a dynamic CA test sequence that targets cell-level transition faults plus intra-cell dynamic defects not covered by a standard transition fault ATPG. In this case, an at-speed *Launch-On-Capture* (LOC) scheme (also called *fast sequential*) is used during test application.

TABLE 5.4: TEST SCENARIOS CONSIDERED IN [13-18]

	Test Scenarios		
Fault & Defect	*Static (Low speed)*	*Dynamic (At-speed)*	*Static + Dynamic*
Stuck-at & Static CA	[14] [16]	[18]	[13] [17]
Transition & Dynamic CA		[15] [16] [18]	[13] [17]

To construct the comprehensive flow described in [13], a new framework was set up in which specific rules were defined to achieve a high level of effectiveness in terms of diagnosis accuracy and resolution. The proposed method was based on a Gaussian Naive Bayes trained model to predict good defect candidates. This method is summarized in the next subsection.

In [18], a new version of the CA diagnosis flow was proposed, assuming a test scenario in which **both** static and dynamic defects can be diagnosed owing to a **single** dynamic CA test sequence applied at-speed. This scenario may happen when such a test sequence has been generated to target transition faults plus cell-internal dynamic defects and appears to also cover the required percentage of stuck-at faults plus cell-internal static defects (or, more generally, satisfies the test coverage specifications). In this case, note that only one (dynamic) datalog is generated after test application and can further be used for diagnosis purpose. Nevertheless, both static and dynamic defects are considered in this scenario. As only dynamic instance tables are manipulated, the representation of training and new data is simplified, i.e., a single type of feature vector is used, without no loss of information and hence without decreasing the quality of the training and inference phases.

4.2 Learning-Based Cell-Aware Diagnosis Flow

Figure 5.6 is a generic view of the learning-based CA diagnosis flow utilized in [13]. It is 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.

After investigating several ML algorithms and observing their inference accuracies in [14], a Bayesian classification method has been chosen for the learning and inference phases in [14-18]. So, the first main step of the CA diagnosis flow consists in generating a Naive Bayes (NB) model and to train it by using a training dataset. In this step, training data are used to incrementally improve the model's ability to make inference. The training dataset is divided into mutually exclusive and equal subsets. For each subset, the model is trained on the union of all other subsets. Some manipulations, such as grouping data by considering equivalent defects or removing data instances of undetectable defects, are also done during this phase. Once training is complete, the performance (accuracy) of the model is evaluated by using a part of the dataset initially set aside. More details about performance evaluation as done in this framework can be found in [16].

The second main step consists in implementing the NB classifier by using a Gaussian distribution to model the *likelihood* probability functions and use this classifier to make prediction when a new data instance has to be evaluated. The next subsections detail the various steps of the CA diagnosis flow, which is able to deal with any type of cell-internal defect (i.e., static and dynamic) that may occur.

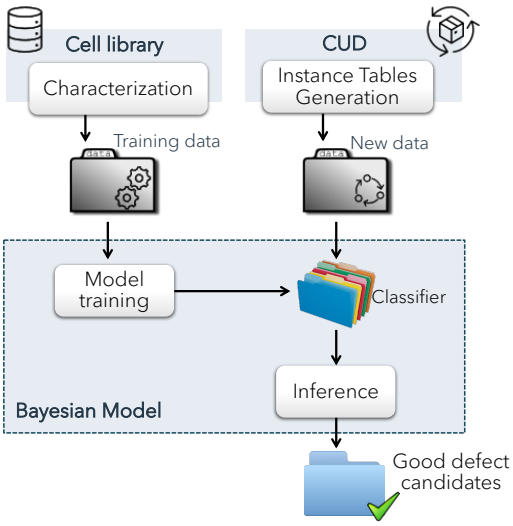


Fig. 5.6 Generic view of the cell-aware diagnosis flow used in [13]

4.2.1 Generation of Training Data

Training data are generated for each type of standard cell existing in the CUD during an off-line characterization process done only once for a given cell library. These data are extracted from CA views provided by a commercial 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. An example of training dataset, as used in [13-18] and containing six instances for an arbitrary two-input cell, is shown in Fig. 5.7. Each instance is associated to a static defect (D_1, D_2, D_3) or a dynamic defect (D_{11}, D_{12}, D_{13}). A 1 (0) indicates that defect D_i is detectable (not detectable) at the output of the cell when the cell-level test pattern P_j is applied at the inputs of the cell. Cell-level test patterns (called *cell-patterns* in the sequel) are static (one input vector - P_1 to P_4 in Fig. 5.7) or dynamic (two input vectors - P_5 to P_{16} in Fig. 5.7 in which R (F) indicates a rising (falling) transition at the cell input respectively). For an n-input cell, there exists 2^n static cell-patterns and $2^n.(2^n–1)$ dynamic cell-patterns.

Dynamic defects can be detected not only by dynamic patterns, but also by static patterns applied using a basic 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 (*these are the detection conditions of a dynamic defect modeled by a*

stuck-open or a gross delay fault). For this reason, the value '0.5' is assigned to each dynamic defect (D_{11}, D_{12}, D_{13}) for all related static cell-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	Pattern
00	01	10	11	0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	Defect
1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	D1
1	0	0	1	0	1	0	1	0	1	1	1	0	0	1	0	D2
0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	D3
0.5	0.5	0.5	0.5	0	0	0	0	0	0	1	0	0	0	0	0	D11
0.5	0.5	0.5	0.5	1	0	0	0	0	0	0	0	1	0	0	0	D12
0.5	0.5	0.5	0.5	0	0	0	0	1	0	0	0	0	0	0	1	D13

Fig. 5.7 Example of training dataset for all defect types in a two-input cell as used in [13-17]

As only dynamic test sequences are considered in [18], the representation of training data as used in [13-17] could be simplified without losing information and decreasing the quality of the training phase. This comes from the observation that a static defect is a particular case of dynamic defect (e.g., a full open is a resistive open with an infinite value of the resistance), and that all static cell-patterns for a given defect are embedded in its whole set of dynamic cell-patterns. Indeed, a dynamic defect requires a two-vector test pattern (V_1V_2) in which the values of V_1 and V_2 have to be properly defined for the defect to be detected. Conversely, only the value of V_2 is significant for a static defect to be detected by such pattern, irrespective of the value taken by V_1. When looking at Fig. 5.7, one can see that P_1={00} is embedded in P_6={0F}, P_{11}={F0} and P_{12}={FF}, and the same for P_2, P_3 and P_4. Similarly, one can see that static defect D_2 is detectable by P_1 and P_4, and hence by P_6, P_8, P_{10}, P_{11}, P_{12}, and P_{15}. So, by "compacting" a training dataset as shown in Fig. 5.8, in which only dynamic cell-patterns are considered, one can see that all meaningful information is still contained in this set, while redundant ('0' and '1' values in the first four columns of Fig. 5.7) or insignificant ('0.5' values in the same columns for dynamic defects) information is removed. More generally, such compact format for training data makes so that only one type of feature vector (dynamic) is used for both types of defects.

As the goal with training data is to provide a distinct feature vector for each data (defect), it is important to be able to distinguish between static and dynamic defects with such a new format of the training dataset. Let us consider two defects D_1 and D_{11} where D_1 is static and detectable by {00} and D_{11} is dynamic and detectable by {F0} (note that {00} is the second vector of {F0}). As can be seen in Fig. 5.8, these two defects can easily be distinguished since their training data instances (or *feature vectors*) are different. The consequence of using such a new format for training data (and hence for new data as will be shown later on) is not an improved accuracy or resolution, but rather a simplified manipulation of feature vectors.

P1′	P2′	P3′	P4′	P5′	P6′	P7′	P8′	P9′	P10′	P11′	P12′	Pattern
0R	0F	R0	RR	RF	R1	F0	FF	FR	F1	1R	1F	Defect
0	1	0	0	0	0	1	1	0	0	0	0	D1
0	1	0	1	0	1	1	1	0	0	1	0	D2
1	0	0	0	0	0	0	0	1	1	0	0	D3
0	0	0	0	0	0	1	0	0	0	0	0	D11
1	0	0	0	0	0	0	0	1	0	0	0	D12
0	0	0	0	1	0	0	0	0	0	0	1	D13

Fig. 5.8 Example of training dataset for all defect types in a two-input cell as used in [18]

4.2.2 Generation of Instance Tables

An instance table is a failure mapping file generated for each suspected cell by using information contained in the tester datalog. It describes the behavior (pass / fail) of the cell for each cell-pattern occurring on its inputs during test of the CUD. The generation process of instance tables is sketched in Fig. 5.9. First, CA test patterns are applied to the CUD. These test patterns are obtained from a commercial CA test pattern generation tool that targets intra-cell defects. Next, a datalog containing information on the failing test patterns and corresponding failing primary outputs is obtained. From this datalog and the circuit netlist, a logic diagnosis is carried out (still using a commercial tool) and gives the list of suspected cells. From this list and the datalog information, one can finally generate an instance table for each suspected cell. Note that in case several test sequences, e.g., one static and one dynamic, are used for diagnosis of the CUD, the generation process is repeated so as to produce static and dynamic instance tables for all suspected cells. This is the case in [13].

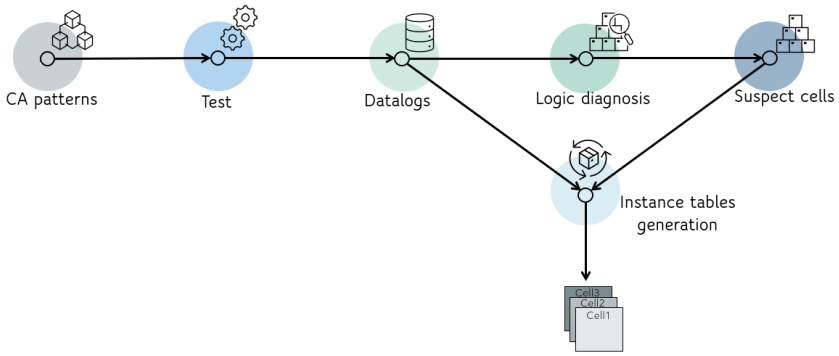


Fig. 5.9 Generation flow of instance tables

The format of a static instance table is illustrated in Fig. 5.10 for a given two-input NOR cell and two static cell-patterns. 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 in Fig. 5.10 represent the stimulus arriving at the cell inputs during the shift phase (before '-') and applied during the launch cycle (after '-'). For example, cell-pattern 2 consists in applying a 1 on input A and B, and failing in detecting a stuck-at 1 on Z.

```
-----------------------------------------------------------------
                NOR Cell - NR2NHVTX1
-----------------------------------------------------------------
          Z    Output    L412/C1381A
          A    Input     U59/Z
          B    Input     U28/Z
-----------------------------------------------------------------
      Pattern 1    PASSING    Z: stuck-at-0
          Z    000011111111111 - 1
          A    111100000000000 - 0
          B    000000000000000 - 0
      Pattern 2    FAILING    Z: stuck-at-1
          Z    011100000000000 - 0
          A    000011111111111 - 1
          B    100011111111111 - 1
-----------------------------------------------------------------
```

Fig. 5.10 Example of static and dynamic instance tables

4.2.3 Generation of New Data

New data are generated after post-processing of instance tables. They are composed of various instances, each of them being associated to one suspected cell in the CUD and represent a feature vector that characterizes the real behavior of the cell during test application. From each new data instance, one can extract one or more defect candidates that must be classified as good or bad candidate with a corresponding probability to be the root cause of failure. This classification is done by comparing the new data instance with the training data of the corresponding suspected cell and identify those training data instances that match (or not) with the new data instance.

The formats of a new data instance as used in [13-17] and [18] are illustrated in Fig. 5.11 and Fig. 5.12 respectively. This format is quite close to the format of a training data instance but has a different meaning. In each instance, the value '1'

(resp. '0') is associated to a failing (resp. passing) cell-pattern P_i for a given defect candidate, meaning that the candidate is **actually** detectable (resp. undetectable) by the cell-pattern P_i at the output of the cell during test of the CUD and hence can (cannot) be the real defect. In such instance, the value '0.5' is associated to a cell-pattern for a given defect candidate when this pattern cannot appear at the inputs of a suspected cell during real test application with an ATE. The median value '0.5' was chosen to avoid missing information in new data instances while not biasing the features of these data.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	
0.5	0.5	0.5	0.5	0	0	0	0.5	0	1	0	0	0	0	0	0	D_i

Fig. 5.11 Format of a new data instance for a two-input cell as used in [C-H]

P1'	P2'	P3'	P4'	P5'	P6'	P7'	P8'	P9'	P10'	P11'	P12'	
0	0	0.5	1	0	1	0	0	0.5	0	0	0	Di

Fig. 5.12 Format of a new data instance for a two-input cell as used in [G]

4.2.4 Diagnosis of Defects in Sequential Cells

All the work carried out in [13-18] was about diagnosis of defects occurring in combinational standard cells of a customer returns. However, defects in SoCs may also occur in sequential standard cells of logic blocks. This section shows how the previous diagnosis flow can handle sequential cells and related defects by adding new information to the training dataset [19].

The two main differences between a combinational cell and a sequential cell are that i) the latter has a clock input pin and ii) the fact that the previous logic value of a sequential cell output can affect the current output value of the cell. To take this difference into account, each cell-pattern for a sequential cell is considered as a tuple in which the first value represents the input clock signal (pulsing or not), the second value is associated to the main input of the cell (e.g., D), and the third value is associated to a virtual input pin representing the previous value of the output pin of the cell (e.g., Q). *Note that in case of sequential cells with multiple real inputs (e.g., D flip-flop with a D, Scan-In, Scan-Enable and Clock input signals), the cell-pattern representation is expanded accordingly*. In each tuple, the first value is either U (i.e., pUlse) or 0, depending on whether or not there is an active clock signal. The second value can be 0, 1, R or F. The third value can only be static (i.e., 0 or 1). An example of training dataset for all defect types (static and dynamic) that may occur in a sequential cell is shown in Fig. 5.13. Note that the CA views used during the generation of training data do not contain information about cell-patterns with

non-pulsing clock signals (i.e., none of the cell internal defects can be detected at the cell output without clock pulse). Consequently, the training data do not include such cell-patterns as can be observed in the example of Fig. 5.13. Note also that instance tables of sequential cells may contain cell-patterns with no transition on the main inputs of the cell. To allow the ML algorithm understanding this information, the solution consists in including static cell-patterns (e.g., P1 to P4 in Fig. 5.13) in the training data of sequential cells.

P1	P2	P3	P4	P5	P6	P7	P8	Pattern
U00	U01	U10	U11	UR0	UR1	UF0	UF1	Defect
0	0	1	0	0	0	1	0	D1
1	1	0	0	0	0	1	1	D2
0.5	0.5	0.5	0.5	1	1	0	0	D11
0.5	0.5	0.5	0.5	1	0	0	1	D12

Fig. 5.13 Example of training dataset for all defect types (static and dynamic) in a sequential cell. The pin order is clock-data-previous output.

With the above representation of training data for sequential cells, one can see that the diagnosis flow in Fig. 5.6 can be used in a straightforward manner without any change. The two main steps (model training by using a training dataset, implementation of the NB classifier to make inference) remain the same irrespective of the type of manipulated standard cells.

5 Applications on Industrial Cases

This section shows the experimental results of ML usage for CA model generation as well as CA diagnosis.

5.1 CA Model Generation Results

The CA model generation is performed with a python program. The ML algorithms were taken from the publicly available python module called scikit-learn [20]. The dataset was composed of 1712 standard cells coming from standard cell libraries developed using three technologies (C40 (446 cells), 28SOI (825 cells) and C28 (441 cells)). All these cells already had a CA model generated by a commercial tool. The CA-matrix is generated for each cell. The method was experimented in two different ways. First, the ML model was trained and evaluated using cells belonging

to one technology. Second, the model was trained on one technology and evaluated it on another one.

5.1.1 Predicting defect behavior on the same technology

The ML model is first trained on cells of 28SOI standard cell libraries. As mentioned earlier, cells were grouped according to their number of transistors and inputs. For *m* cells available in each group, the ML model is trained over *m-1* cells and evaluate its **prediction accuracy** on the *m*-th cell. A loop ensured that each cell is used as the *m*-th cell. On average, a group contains 8.6 cells. In the following, all possible open and short defects (static and dynamic) in each cell are considered. Results presented below report the prediction for open defects. Results achieved for short defects are similar.

TABLE 5.5: PREDICTION ACCURACY FOR CELLS IN THE SAME TECHNOLOGY

Prediction accuracy (%)		Number of inputs				
		2	*3*	*4*	*5*	*6*
Number of transistors	*6*	99.98	99.99			
	8	99.91	99.96	99.91		
	9		100.0			
	10	99.98	99.81	99.96		
	12	99.72	99.73	100.0	99.91	99.93
	14	99.70	99.56	99.83	99.92	99.96
	16	99.99	100.0	99.94		99.98
	18	99.99	99.94			
	20	100.0	99.98	100.0	99.73	
	22		99.84	99.98	99.62	
	24	100.0	99.84	**99.97**		99.85
	26	100.0	99.70	100.0		99.89
	28	99.49	99.98	100.0	99.88	99.81
	30	99.75	100.0	100.0		
	32	100.0	100.0			99.98
	42		100.0			
	44		100.0			
	46		99.81			
	47		99.98	99.95		

Table 5.5 presents the prediction accuracy achieved for open defects. Non-empty boxes report the **average** prediction accuracy obtained **for a group of cells**. Empty boxes mean that there is zero or one cell available and that the group cannot be evaluated. A grey background indicates that the maximum prediction accuracy in this group is 100%, i.e., the ML model can perfectly predict the defective behavior of at least one cell. In contrast, white background indicates that no cell was perfectly

predicted in that group (all prediction accuracies are less than 100%). For example, let us consider the bold value in Table 5.5. One has 24 cells having 4 inputs and 24 transistors: i) 15 cells are perfectly predicted (100% accuracy), which leads to a green background, ii) the prediction accuracy for the 9 remaining cells ranges from 99.82% to 99.99% and iii) the average prediction accuracy over all 24 cells is 99.97%.

These results show that the ML model can accurately predict the behavior of a cell affected by a given defect and that our method could be used to generate CA models. The goal of the next subsection is to leverage on existing CA models to generate CA models for a new technology.

5.1.2 Predicting defect behavior on another technology

Experiments are also conducted on cells belonging to two different technologies. Evaluation was slightly different compared to the previous one. Here, the ML model was trained over all available cells of a given technology and the evaluation was done on one cell of another technology. A loop was used to allow all cells of the second technology to be evaluated. Cells were grouped according to their number of inputs and transistors.

TABLE 5.6: AVERAGE PREDICTION ACCURACY FOR CELLS IN DIFFERENT TECHNOLOGIES

Prediction accuracy (%)		Number of inputs				
		2	*3*	*4*	*5*	*6*
Number of transistors	*6*	98.21	99.47			
	8	94.56	96.86	99.00		
	9					
	10	94.69	96.01	99.27		
	12	87.73	98.05	99.10		99.76
	14	85.69	97.35	98.75		
	16	91.74		99.20		
	18	88.18	96.28			
	20	90.29	94.37			
	22	78.73		98.37		
	24	87.91	96.88	99.37		99.79
	26	87.24	98.92			
	28	88.18	98.68			
	30			97.52		
	32	88.73	95.6			
	42					
	44					
	46					
	47					

Table 5.6 shows the prediction accuracy achieved on open defects of the C28 cells after training on the 28SOI cells. Results are averaged over all cells in each group (same number of inputs and number of transistors). The average prediction accuracies are globally lower compared to those of Table 1. After investigating on this point, one can noticed that the behavior of most of the cells (68% of cells) is accurately predicted (accuracy > 97%), while accuracy for few cells is quite low. This phenomenon is discussed later in this section.

To verify the efficiency of the method when different transistor sizes are considered, the ML model was trained over the 28SOI standard cells and used to predict the behavior of C40 cells. Table 5.7 shows the prediction accuracy achieved on open defects of the C40 cells after training on the 28SOI cells. Results are averaged over all cells in each group (same number of inputs and transistors). This time, 80% of cells are accurately predicted (accuracy > 97%), proving that our ML-based characterization methodology could be used to generate CA models for a (large) part of cells of a new technology.

TABLE 5.7: AVERAGE PREDICTION ACCURACY FOR CELLS WITH DIFFERENT TRANSISTOR SIZE

Prediction accuracy (%)		Number of inputs				
		2	*3*	*4*	*5*	*6*
Number of transistors	*6*	100.0	99.80			
	8	87.39	99.14	99.03		
	9		97.19			
	10	92.07	95.49	99.32	98.46	
	12	91.71	98.07	99.24	98.47	99.46
	14	90.1	95.84	98.63	98.79	99.52
	16	91.17	93.59	99.23		99.59
	18	88.5	97.15	97.14	97.74	
	20	83.87	97.73	97.15	98.94	
	22	87.26		98.98	98.44	
	24	93.96	99.34	98.58	98.84	99.63
	26	87.52	97.55	99.04	99.02	99.92
	28	98.19		98.79	99.31	99.44
	30			99.13	99.37	99.58
	32	92.91			98.92	99.78
	42					
	44	92.03	98.82			
	46		99.23			
	47		98.29	99.76		

5.1.3 Analysis and Discussion

The cells for which the defect characterization methodology gives excellent prediction accuracy as well as those for which the prediction accuracy was quite low are firstly analyzed. Then, the limitation of the detailed method for CA model generation is investigated. After running several experiments on different configurations using one fault model at a time, one noticed the following behaviors:

- Accuracy for most of the cells is excellent, i.e., more than 97% prediction accuracy for 70% of cells. In this case, **the CA model generated by ML fit the real behavior achieved with electrical simulation**.
- Accuracy for few cells (30%) is quite low and the ML prediction is not accurate.

For the first cell category with good prediction score, cells have been analyzed manually to identify why they led to good results. The analysis showed that all these cells had at least one cell in the training dataset with the same transistor structure or a similar one.

For the second cell category – cells leading to poor prediction accuracy – the manual analysis showed that they have: i) new logic functions that do not appear in the cells of the training dataset, or ii) a transistor configuration which is completely new when compared to cells in the training dataset.

5.1.4 Hybrid flow for CA model generation

Considering the above analysis, it appears that the ML-based CA model generation flow cannot be used for all cells in a standard cell library to be characterized. A mixed solution, which consists in combining ML-based CA model generation and conventional (simulation-based) CA model generation, should be preferably used. This is illustrated in the following.

The flow sketched in Fig. 5.14 is proposed for accelerating the CA model generation. Typically, when the CA model for a new cell is needed, one first check if the ML-based generation will lead to high-quality CA models. This is done by analyzing the structure of the new cell and check whether the training dataset contains a cell with identical or similar structure (as presented in V.B). If the ML algorithm is expected to give good results, the new cell is prepared (representation in a CA-matrix) and submitted to the trained ML algorithm.

The output information is then parsed to the desired file format. Conversely, if the ML algorithm is expected to give poor prediction results, the standard generation flow presented in Fig. 5.14 is used to obtain the CA model. A feedback loop uses this new simulated CA model to supplement the training datasets and improve the ML algorithm for further prediction.

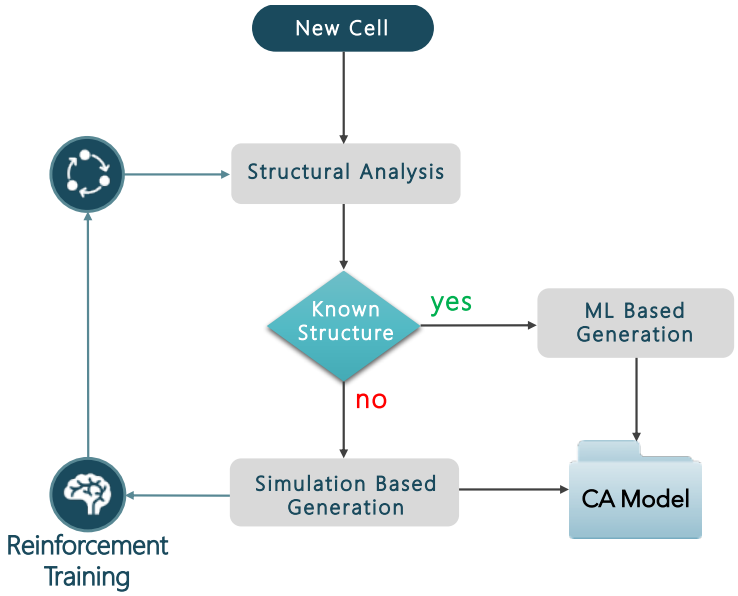


Fig. 5.14 Hybrid flow for CA model generation.

To estimate the improvement in CA model generation time achieved with the flow in Fig. 5.2, the following experiments are performed. The Random Forest model is trained on 28SOI standard cells and generated CA models for a subgroup of the C40 standard cell libraries. A subgroup is composed of cells representing all the cell functions available in C40 libraries. In our experiments, this subgroup contained 409 cells: 118 (29%) have a cell with an identical structure in the training dataset, 87 (21%) have a cell with an equivalent structure (as explained in Section V.B) in the training dataset, and 204 (50%) have no identical or equivalent structure in the training dataset (a simulation-based generation is thus needed). For these 204 cells, the generation time was calculated and found to be equal to ~172 days (~ 5.7 months) considering a single SPICE license. Using the ML-based CA model generation for the 118 + 87 = 205 (50%) remaining cells requires 21947 seconds (~ 6 hours), again considering a single SPICE license. Considering that a simulation-based generation for these 205 cells would require ~78 days, one can estimate the reduction in generation time to **99.7%**. Now, if one considers the whole C40 subgroup composed of 409 cells, the hybrid generation flow would require ~172 days + ~6 hours, to be compared with ~172 days + ~78 days = ~250 days by using only the simulation-based generation. This represents a reduction in generation time of about **38%**. After investigating results of these experiments, one can observed that the ML-based CA model generation works well for about 80% of cells of the C40 subgroup. Surprisingly, the structural analysis revealed that only 50% (205 cells) could be evaluated using the ML-based generation part of the flow. This shows that there is still room for further improvement of the structural analysis in this flow, and hence get better performance of the ML-based CA model generation process.

To conclude, experiments have been carried out on a reasonable size (1712) of standard cell population. Considering that more than 10000 cells have usually to be characterized for a given technology, the hybrid flow in Fig. 5.14 is expected to provide even better results, especially owing to the reinforcement training that uses simulation generated models for supplementing the training datasets, and hence reduce the number of electrical simulations.

5.2 CA diagnosis results

The CA diagnosis flow described in Section 4.2 and targeting defects in both combinational and sequential cells of CUD has been implemented in a Python program. For validation purpose, authors in [13-19] have experimented the proposed flow in three different ways:

- First, they conducted experiments on ITC'99 benchmark circuits with defect injection campaigns targeting **combinational cells** in each circuit. Various results are reported in [13-19] to show the superiority of the framework when compared to commercial diagnosis solutions.
- Next, they considered a test chip developed by STMicroelectronics and designed using a 28 nm FDSOI technology, and they conducted two defect injection campaigns targeting **sequential cells** [19]. Results are reported in subsection 5.2.1 and demonstrate the effectiveness of the diagnosis framework.
- Finally, they considered **a customer return** from STMicroelectronics and performed a silicon case study with a real defect subsequently analyzed and identified during PFA. Results are reported in subsection 5.2.2.

5.2.1 Simulated Test Case Studies

Authors in [19] conducted experiments on a silicon test chip developed by STMicroelectronics and designed with a 28 nm FDSOI technology. The test chip is only composed of digital and memory blocks, and one PLL. The digital blocks are made of 3.8 million cells. Other features (number of primary inputs, primary outputs, and scan flip-flops) are given in Table 5.8.

A first simulated case study was done with a **static** defect injection campaign. All possible static defects were successively injected into three scan flip-flops (SFF) of a single full-scan digital block. This block was tested with a static CA test sequence achieving a stuck-at + static CA fault coverage of 100%. The average numbers of passing and failing test patterns are given in Table 5.9. Results obtained after executing the 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 **1.25.** The resolution ranges between 1 and 3, and Fig. 5.15 shows

the distribution of this resolution with respect to the total number of simulated cases. As can be seen, in most of the cases, the number of suspects is equal to 1 (perfect resolution).

TABLE 5.8: MAIN FEATURES OF THE SILICON TEST CHIP

#cells	#PIs	#POs	#SFF
3.8M	97	32	17.5k

TABLE 5.9 AVERAGE PATTERN COUNT IN INSTANCE TABLES OF THE FIRST SIMULATED CASE STUDY

#passing patterns	#unique passing patterns	#failing patterns	#unique failing patterns
43.4	24.0	15.5	8.6

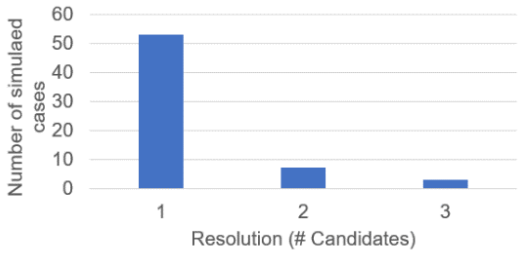


Fig. 5.15 Distribution of the resolution with respect to the simulated cases

A second simulated case study with another defect injection campaign was performed on the same test chip. All possible dynamic defects were successively injected into three scan flip-flops of a single full-scan digital block. This time, a dynamic CA test sequence was applied and achieved a transition + dynamic CA fault coverage of 89.8%. The average number of failing test patterns was 7.9. Again, the results obtained after executing the CA diagnosis flow and averaged over all defect injections have shown an accuracy of 100%. The average resolution obtained for dynamic defect injection experiments was 1.37. Again, the resolution ranged between 1 and 3, and in most of the case, the number of suspects was equal to 1.

5.2.2 Silicon Test Case Studies

Next, a silicon case study was performed on a customer return designed with a 28 nm FDSOI technology from STMicroelectronics [19]. The test conditions used to run the experiments were as follows: a nominal supply voltage of 0.83 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 25°C. The process was considered as typical. The CA diagnosis flow was experimented and the following results were obtained. Initially, the circuit failed on the tester after application of the static CA test sequence when applied at the nominal voltage. This information was stored in a "static" datalog. Then, a logic diagnosis gave a short list of suspected cells among which a six-input SFF cell made of 56 transistors and having a Reset, an Enable, a Test-Input and Test-Enable input pins. The cell contains 758 potential short or open defects. A static instance table was then generated for this suspected cell and contained 5 failing and 75 passing cell-level test patterns. From the new data generated after post-processing of this instance table, the NB classifier identified four suspected defects among which defect D62 (a short between the gate and source of NMOS 19).

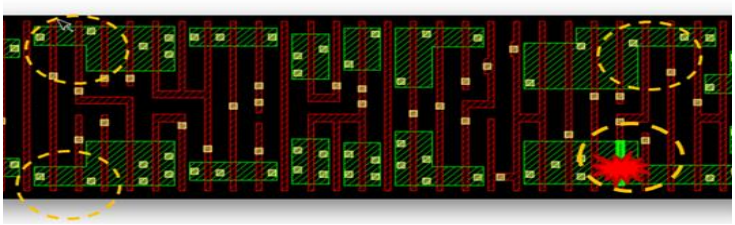


Fig. 5.16 Layout view of the suspected cell and the incriminated transistor. Yellow circles indicate defect candidates and red mark indicates actual observed defect

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 intra-cell diagnosis tool. The result obtained with the CA diagnosis flow was validated as defect D62 was found to be the real defect. This was done after performing a polysilicon level inspection on the layout of the cell (c.f. Fig. 5.16) and observing the failure analysis cross-sectional view.

6 Conclusion and Discussion

This chapter has provided an overview of the various machine learning approaches and techniques proposed to support cell-aware generation, test, and diagnosis from leading-edge research in this domain. In a comprehensive form, it proposes a compendium of solutions existing in this field. More in detail, the chapter has presented, after some backgrounds on conventional approaches to generate and diagnose cell-aware defects, learning-based solutions to generate CA models and CA diagnosis. Experiments on silicon test cases have been done to validate those solutions and demonstrate their efficacy in terms of accuracy and resolution.

Results of this chapter prove the appropriateness of learning-based methods to solve the problem of CA models generation and CA diagnosis. The learning-based solutions to generate CA models was evaluated using two scenarios: i) the model was trained and evaluated using cells belonging to one technology, and ii) the model was trained on one technology and evaluated it on another one. Those evaluations have shown that the ML-based CA model generation flow cannot be used for all cells in a standard cell library to be characterized. A mixed solution, *Hybrid flow for CA model generation*, which consists in combining ML-based CA model generation and conventional (simulation-based) CA model generation, should be preferably used. The learning-based solution to generate CA diagnosis has been experimented in three different ways: i) on ITC'99 benchmark circuits with defect injection campaigns targeting combinational cells, ii) using a test chip developed by STMicroelectronics and designed using a 28 nm FDSOI technology with defect injection campaigns targeting sequential cells, and iii) using a customer return from STMicroelectronics with a real defect subsequently analyzed and identified during PFA. In all cases, the learning-based solution to generate CA diagnosis succeeds in identifying the good defect candidate compared to commercial diagnosis solutions.

References

1. A. Ladhar, M. Masmoudi, and L. Bouzaida, "Efficient and Accurate Method for Intra-Gate Defect Diagnoses in Nanometer Technology," in Proc. IEEE/ACM Design Automation and Test in Europe, 2009.
2. 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 Symp. on Quality Electronic Design, 2013.
3. F. Hapke, et al., "Cell-Aware Test," IEEE Transactions on Computer-Aided Design, vol. 33, no. 9, pp. 1396 - 1409, 2014.
4. P. Maxwell, F. Hapke, and H. Tang, "Cell-Aware Diagnosis: Defective Inmates Exposed in their Cells," in IEEE European Test Symp., 2016.
5. 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.
6. S. Wang and W. Wei, "Machine Learning-Based Volume Diagnosis," in Proc. IEEE/ACM Design Automation and Test in Europe, 2009.
7. F. Lorenzelli, Z. Gao, J. Swenton, S. Magali, and E.J. Marinissen, "Speeding up Cell-Aware Library Characterization by Preceding Simulation with Structural Analysis," in *Proc. IEEE European Test Symp.*, 2021.
8. S. Venkataraman and S.D. Drummonds, "A Technique for Logic Fault Diagnosis of Interconnect Open Defect", in *IEEE VLSI Test Symp*, 2000.
9. C.-M. Li and E.J. McCluskey, "Diagnosis of Resistive-Open and Struck-Open Defects in Digital CMOS ICs", *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 24, no 11, pp. 1748 – 1759, 2005.

10. P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, "A Learning-Based Methodology for Accelerating Cell-Aware Model Generation", IEEE /ACM Design Automation and Test in Europe, 2021.
11. J.A. Waicukauski and E. Lindbloom, "Failure Diagnosis of Structured VLSI," IEEE Design & Test of Computers, vol. 6, no. 4, pp. 49-60, July 1989.
12. M. Abramovici and M. A. Breuer, "Fault Diagnosis Based on Effect-Cause Analysis: An Introduction," in Proc. ACM Design Automation Conference, pp. 69–76, 1980.
13. S. Mhamdi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "A Learning-Based Cell-Aware Diagnosis Flow for Industrial Customer Returns," in Proc. IEEE International Test Conference, 2020. DOI: 10.1109/ITC44778.2020.9325246
14. 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.
15. S. Mhandi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "Cell-Aware Diagnosis of Automotive Customer Returns Based on Supervised Learning," presented at IEEE Automotive Reliability and Test Workshop, 2019.
16. S. Mhamdi, 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 Material and Reliability, vol. 20, no. 2, 2020.
17. S. Mhandi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "Learning-Based Cell-Aware Defect Diagnosis of Customer Returns," in Proc. IEEE European Test Symposium, 2020.
18. S. Mhandi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "Cell-Aware Diagnosis of Customer Returns Using Bayesian Inference," in Proc. IEEE International Symposium on Quality Electronic Design, 2021.
19. P. d'Hondt, S. Mhamdi, P. Girard, A. Virazel, and A. Ladhar, "A Comprehensive Framework for Cell-Aware Diagnosis of Customer Returns," to appear in Microelectronics Reliability Journal, October 2021.
20. F. Pedregosa et al, "Scikit-learn: Machine learning in Python," Journal of machine learning research, 12(Oct), pp.2825–2830. 2011
21. N. Mukherjee et al, "Digital Testing of ICs for Automotive Applications" in Proc of IEEE International Conference on VLSI Design
22. F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, S. Eichenberger, et al, "Defect-Oriented Cell-Aware ATPG and Fault Simulation for Industrial Cell Libraries and Designs", Proc. IEEE International Test Conference, Nov. 2009.
23. F.M. Goncalves, I.C. Teixeira, and J.P. Teixeira, "Integrated Approach for Circuit and Fault Extraction of VLSI Circuits," Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Nov. 1996.
24. F.M. Goncalves, I.C. Teixeira, and J.P. Teixeira, "Realistic Fault Extraction for High-Quality Design and Test of VLSI Systems," in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Oct. 1997.
25. Z. Stanojevic, and D.M. Walker, "Fed–x - A Fast Bridging Fault Extractor," Proc. of IEEE International Test Conference, Nov. 2001.
26. X. Fan, W. Moore, C. Hora, and G. Gronthood, ''A Novel Stuck-At Based Method for Transistor Stuck-Open Fault Diagnosis,'' in Proc. IEEE International Test Conference, pp 386-395, 2005.
27. F. Hapke, M. Reese, J. Rivers, A. Over, V. Ravikumar, W. Redemund, A. Glowatz, J. Schloeffel, and J. Rajski, "Cell-Aware Production Test Results from a 32-nm Notebook Processor," in Proc. IEEE International Test Conference, 2012. DOI: 10.1109/TEST.2012.6401533
28. A. Sun, A. Bosio, L. Dillilo, P. Girard, A. Virazel, S. Pravossoudovitch, and E. Auvray, "Intra-Cell Defects Diagnosis," Journal of Electronic Testing: Theory and Applications, vol. 30, no. 5, pp. 541-555, 2014.