THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En SYAM - Systèmes Automatiques et Micro-Electroniques

École doctorale - Information, Structures et Systèmes

Unité de recherche - Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier

# TECHNIQUES D'APPRENTISSAGE POUR LA CARACTÉRISATION DE BIBLIOTHÈQUES DE CELLULES EN VUE DU TEST ET DU DIAGNOSTIC DE CIRCUITS INTÉGRÉS

Présentée par Pierre D'HONDT

le 23 novembre 2022

Sous la direction de Patrick GIRARD
et Arnaud VIRAZEL

Devant le jury composé de

| | |
|---|---|
| François BERRY, Professeur à l'Université Clermont-Auvergne, ISPR, Aubière | Président |
| Lorena ANGHEL, Professeure à INP Grenoble / PHELMA, SPINTEC, Grenoble | Rapporteur |
| Alberto BOSIO, Professeur à l'École Centrale de Lyon, INL, Lyon | Rapporteur |
| Aymen LADHAR, Ingénieur Test et Diagnostic, STMicroelectronics, Crolles | Examinateur |
| Patrick GIRARD, Directeur de Recherche au CNRS, LIRMM, Montpellier | Directeur de thèse |
| Arnaud VIRAZEL, Professeur à l'Université de Montpellier, LIRMM, Montpellier | Co-directeur |

UNIVERSITÉ DE MONTPELLIER

# Abstract

The rapid growth of worldwide semiconductors demand leads to numerous innovations and improvements in cost, speed, and power consumption. The constant shrinking of the transistors feature size allows the development of new applications and capabilities but reveals new types of manufacturing defects. Efficient test and precise diagnosis are crucial to guarantee the creation of quality products and to improve the production yield. To keep test and diagnosis in phase with the new types of defects, new methodologies and faults models have been invented and deployed. The Cell-Aware (CA) fault models abstract the subtle defects found inside the standard cells used to design digital ICs, at the transistor level. Cell-aware test uses these fault models with ATPG to create test patterns explicitly targeting cells internal defects. Cell-aware diagnosis tools use the CA data to identify the location and type of intra-cell defects, providing valuable insights in the context of large and complex standard cells.

While becoming an industry standard, the CA methodology has a large and costly development overhead, involving numerous electrical simulations to characterize standard cells. Due to the high number of standard cells to be characterized, and, for each cell, the high numbers of potential intra-cell defects and cell-level patterns to consider, the CA characterization phase represents a heavy usage and cost of simulator licenses and computational power.

This thesis presents an innovative flow using Machine-Learning (ML) to reduce the CA test method runtime and ease its adoption for industrial usage. Experiments using different technology nodes demonstrated an over 99% runtime reduction for 80% of combinational cells.

To ensure the generation of a quality CA model for all cells, while decreasing the CA characterization time, a hybrid flow is proposed, mixing ML-based CA models prediction method with the conventional method using electrical simulations. This hybrid-flow includes a decision algorithm, which leverage ML techniques to decide whether the CA characterization of a new standard cell should be ML-based or simulation-based, thus allowing to decrease the CA characterization runtime while maintaining high quality CA models for all cells. Experimental results demonstrate the high performance of the new decision algorithm and the quality of the obtained CA models.

The coverage of real cell-internal defects of ATPG patterns using ML-predicted CA data proves that our predicted CA data can accurately replace those obtained by running extensive analog simulations, thus proving the effectiveness and pertinence of the proposed methodology.

# Résumé

La croissance rapide de la demande mondiale de semi-conducteurs conduit à de nombreuses innovations et améliorations en matière de coût, de vitesse et de consommation d'énergie. La réduction continue de la taille des transistors permet le développement de nouvelles applications et fonctionnalités, mais fait apparaître de nouveaux types de défauts de fabrication. Pour garantir la fabrication de produits de qualité et améliorer le rendement de production, il est nécessaire de disposer d'outils de test efficaces et de diagnostic précis. Pour permettre la manipulation des nouveaux types de défauts par les outils de test et de diagnostic, de nouvelles méthodes et de nouveaux modèles de fautes ont été inventés et développés. Les modèles de fautes Cell-Aware (CA) représentent les défauts subtils situés à l'intérieur des cellules standards utilisées pour concevoir des circuits intégrés numériques, au niveau transistor. Le test CA fournit ces modèles de fautes à un ATPG pour créer des vecteurs de test ciblant explicitement les défauts internes des cellules. Les outils de diagnostic CA utilisent les données CA pour identifier l'emplacement et le type des défauts intra-cellules, fournissant de précieuses informations pour les cellules standards les plus larges et complexes

Bien que devenant un standard dans l'industrie, la méthodologie CA a un surcoût de développement important et coûteux, impliquant de nombreuses simulations électriques pour caractériser les cellules standards. En raison du nombre élevé de cellules standards à caractériser et, pour chaque cellule, des nombres élevés de défauts intra-cellules potentiels et de stimuli possibles à appliquer aux entrées de la cellule, la phase de caractérisation CA représente un coût important en matière de licences de simulateur à utiliser et de puissance de calcul nécessaire.

Ce manuscrit présente une méthode innovante qui utilise l'apprentissage automatique (Machine-Learning — ML) pour réduire le temps d'exécution du test CA et faciliter son adoption par l'industrie. Des expériences utilisant différents nœuds technologiques ont démontré une réduction de plus de 99% du temps d'exécution pour 80% des cellules combinatoires.

Pour assurer la génération d'un modèle CA de qualité pour toutes les cellules standards, tout en diminuant le temps de caractérisation CA, un flot hybride est proposé, mélangeant la méthode de prédiction des modèles CA basés sur l'apprentissage

automatique avec la méthode conventionnelle utilisant des simulations électriques. Ce flot hybride comprend un algorithme de décision, qui tire également parti des techniques d'apprentissage automatique pour décider si la caractérisation CA d'une nouvelle cellule standard doit être basée sur des méthodes ML ou bien basée sur des simulations analogiques, permettant ainsi de réduire le temps d'exécution de la caractérisation CA tout en obtenant des modèles CA de haute qualité pour toutes les cellules. Les résultats expérimentaux ont démontré les bonnes performances du nouvel algorithme de décision.

Sur des défauts intra-cellules réels, le taux de couverture de fautes de vecteurs de test obtenus par un ATPG utilisant des modèles CA générés par les méthodes d'apprentissage automatique démontre le fait que nos données CA obtenues par prédiction peuvent efficacement remplacer celles obtenues par simulations. Ce point prouve l'efficacité et la pertinence de la méthodologie proposée.

# Table of Contents

**Abstract** i

**Résumé** iii

**Table of Contents** v

**List of Figures** ix

**List of Tables** xi

**Introduction** 1

**1 Context - Test and Diagnosis of digital ICs** 7
- 1.1 Defects and fault models . . . 8
- 1.2 Test of digital ICs . . . 9
- 1.3 Diagnostic of digital ICs . . . 11

**2 State of the art of CA methodologies** 15
- 2.1 Targeting intra-cell defects . . . 16
- 2.2 Cell-aware methodology . . . 18
- 2.3 Limitations and improvement trials . . . 21
- 2.4 PhD Objectives . . . 23

**3 Machine learning for CA models generation** 25
- 3.1 Learning-Based CA Defect Characterization . . . 26
  - 3.1.1 Motivation . . . 26
  - 3.1.2 Proposed Defect Characterization Methodology . . . 27
- 3.2 Standard cell representation in the CA-matrix . . . 29
  - 3.2.1 Identification of Active and Passive Transistors . . . 30
  - 3.2.2 Renaming of Transistors . . . 31
  - 3.2.3 Identification of Parallel Transistors . . . 32
- 3.3 Defect Representation in the CA-Matrix . . . 33
- 3.4 Modifications for sequential and complex cells . . . 34
  - 3.4.1 Sequential cell representation in the CA-matrix . . . 36

3.4.1.1 Identification of Active, Passive and Pulsing Transistors . 36
3.4.1.2 Renaming of Transistors . . . 37
3.4.1.3 Identification of Parallel Transistors . . . 39
3.5 Experimental results for combinational cells . . . 40
3.5.1 Predicting defects' behavior on the same technology . . . 40
3.5.2 Predicting defects' behavior on another technology . . . 41
3.5.3 Analysis - Discussion . . . 41
3.6 Experimental results for sequential cells . . . 44
3.6.1 Predicting defects' behavior on the same technology . . . 44
3.6.2 Predicting defects' behavior on another technology . . . 45
3.6.3 Controlled experiments . . . 45
3.7 Comparison of results . . . 47

**4 Hybrid flow for CA models generation 51**
4.1 Hybrid flow presentation . . . 51
4.1.1 Runtime reduction results . . . 52
4.2 Improvement using ML-based decision block . . . 53
4.2.1 Fine-graining . . . 53
4.2.2 Presentation of ML-based decision block . . . 54
4.2.3 Generating a dataset for ML-based decision block . . . 55
4.3 Experimental results . . . 57
4.3.1 Machine-Learning decision block evaluation . . . 57
4.3.2 Industrial validation: ATPG with ML-based CA models . . . 61
4.3.2.1 Methodology . . . 61
4.3.2.2 Experiments . . . 62

**5 Machine-learning for cell-aware diagnosis 65**
5.1 Cell-Aware Diagnosis . . . 66
5.2 Previous works . . . 67
5.2.1 Summary of our Previous Work . . . 68
5.2.2 Improvement of Diagnosis Resolution . . . 69
5.3 Learning-based cell-aware diagnosis flow . . . 71
5.3.1 Generation of Training Data . . . 72
5.3.2 Generation of Instance Tables . . . 74
5.3.3 Generation of New Data . . . 74
5.4 Diagnosis of defects in sequential cells . . . 76
5.5 Experimental results . . . 77
5.5.1 Summary of Results Reported in [16] . . . 77
5.5.2 Simulated Test Case Studies . . . 79
5.5.3 Silicon Test Case Study . . . 80

5.6 Conclusion, discussion, and future work . . . . . 81

**6 Conclusions and future work** **83**

**Publications of the author** **85**

**Glossary** **87**

**Bibliography** **89**

# List of Figures


1.1 Generic schematic of an ATPG tool . . . . . 10

2.1 Potential transistor-level faults affecting a standard cell' transistor . . . . . 20
2.2 Simulation-based cell-aware model generation flow . . . . . 21

3.1 Generic view of the ML-based defect characterization flow . . . . . 27
3.2 Generic view of the CA-matrix creation flow . . . . . 28
3.3 Example standard cell NAND2 . . . . . 31
3.4 Example schematic . . . . . 32
3.5 Block-level representation of an example scan flip-flop . . . . . 35
3.6 MUX of the scan flip-flop in Fig. 3.5 . . . . . 37
3.7 MASTER latch of the scan flip-flop in Fig. 3.5 . . . . . 38
3.8 Example schematic . . . . . 39
3.9 Typical transistor configurations leading to good prediction . . . . . 44

4.1 Hybrid flow for CA model generation . . . . . 52
4.2 Updated hybrid flow with the new decision block based on machine learning 54
4.3 Process to create a dataset for the ML-decision algorithm. . . . . 56
4.4 Method to evaluate the ML-decision algorithm. . . . . 58
4.5 Histogram of the prediction accuracy . . . . . 60
4.6 ATPG flow to assess quality of the CA models obtained by the hybrid flow 62

5.1 Example of a multi-run industrial ATPG flow . . . . . 68
5.2 Generic view of the cell-aware diagnosis flow used in [16], [20]–[24] . . . . . 71
5.3 Generation flow of instance tables . . . . . 74
5.4 Example of static and dynamic instance tables . . . . . 75
5.5 Overall Cell-Aware Diagnosis Results [16] . . . . . 78
5.6 Distribution of the resolution w.r.t. the simulated cases . . . . . 80
5.7 Layout view of the suspected cell . . . . . 81

# List of Tables


2.1 Example of CA model dictionary as a detection table . . . . . 19
2.2 Average percentage of common values in CA models for different PVTs . . 22

3.1 Example of training dataset for a NAND2 cell . . . . . 29
3.2 Activity values for the NAND2 cell in Fig. 3.3.a . . . . . 33
3.3 NAND2 cell in Fig. 3.3: Example of defect columns . . . . . 34
3.4 Example of training dataset for a scan flip-flop with a non-inverting output 35
3.5 Branch equations for the schematic of Fig.3.8 . . . . . 39
3.6 Activity values for the mux in Fig. 3.5 . . . . . 40
3.7 Average prediction accuracy for cells in the same technology . . . . . 42
3.8 Average prediction accuracy for cells in a different technology . . . . . 42
3.9 Average prediction accuracy for cells with different transistor sizes . . . . . 43
3.10 Average prediction accuracy for (a) cells in the same technology; (b) cells in different technologies; (c) cells with different transistor sizes . . . . . 46
3.11 Results of the controlled experiments . . . . . 47
3.12 Comparative results – average prediction accuracy - combinational vs sequential cells . . . . . 47

4.1 Average prediction accuracy per defect type for a population of C40 cells . 53
4.2 Decision error (%) of the ML-decision blocks . . . . . 59
4.3 Number of patterns and associated faults coverages - static defects . . . . . 63
4.4 Number of patterns and associated faults coverages - dynamic defects . . . 63

5.1 Test scenarios considered in [16], [20]–[24] . . . . . 68
5.2 Test scenarios and corresponding analysis . . . . . 70
5.3 Example of training dataset for all defect types . . . . . 72
5.4 Example of training dataset for all defect types . . . . . 73
5.5 Format of a new data instance for a two-input cell . . . . . 75
5.6 Format of a new data instance as used in [24] . . . . . 75
5.7 Example of training dataset for all defect types in sequential cells . . . . . 76
5.8 Main features of the silicon test chip . . . . . 79
5.9 Average pattern count in instance tables of the first simulated case study . 80

# Introduction

## Background

To meet the ever-increasing worldwide demand for affordable and efficient semiconductor products, numerous improvements in cost, speed, and power consumption of Integrated Circuits (*ICs*) have been developed by the industry. Simultaneously, such improvements have permitted new applications for ICs, sustaining an increase in demand. In the world of MOS ICs, the demand for enhancing performance and functionality at reduced cost has driven the rapid improvement of the technology, and notably the constant shrinking of the transistor feature size following the famous Moore's law. The Moore's Law was formulated as an empirical observation and is now a challenging objective of the persistent periodic increase in the level of miniaturization. The Moore's Law states that the "the number of components per chips doubles every 18 months". The reduction of the transistors size results in the introduction of a new technology generation or technology node. Every new technology node leads to higher integration, lower energy consumption, better performances and increased operating frequencies. Thus, the semiconductor technology scaling optimizes circuit performance and power consumption with every new technology generation and allows realization of more and more complex systems [1].

CMOS-based digital ICs are manufactured on silicon wafer by doping the silicon with different donors and forming transistors. The transistors are connected using polysilicon and metal wires to form cells. The cells are then connected together through several metal wires to achieve the circuit desired behavior. The manufacturing process of ICs is prone to errors. Excesses or lacks of deposited material cause the circuits to deviate from intended specifications. Manufacturing tools may also introduce chip-to-chip variations (e.g., depending on the chip location on the wafer), leading chips that implement the same design to exhibit diverging timing behaviors. As the semiconductor industry moves to increasingly smaller geometries, new types of manufacturing defects appear. To detect defective or out-of-specification ICs, and avoid selling them to customers, all manufactured chips go through manufacturing test.

The objective of manufacturing test is to thoroughly exercise the Circuit Under Test (*CUT*) in order to detect if it is affected by manufacturing defects or process variations

that make it inoperative or fallen out of specifications. Manufacturing tests apply sequences of electrical stimuli (known as *test vectors* or *patterns*) to the CUT and compare the responses of the CUT with the expected results in terms of logic behavior, frequency, power consumption, etc. Applying every possible test vector to the CUT would be ideal to detect all potential defects. However, the number of possible test vectors grows exponentially with the number of input pins of the ICs. Longer sets of test vectors imply longer testing time and cost. As every manufactured ICs go through manufacturing test, it is important to keep the testing time and cost at minimum to maintain competitive ICs pricing. Therefore, exhaustive testing of ICs is inapplicable on commercial products. In order to decrease the number of test patterns, and therefore decrease the test time, for a given IC, Automatic Test Pattern Generation (*ATPG*) algorithms have been developed to select test patterns with the highest added value. ATPG algorithms use fault models to this end. A *fault model* is a representation of the effect of defects on IC behaviors. A fault model may be described at logic, circuit, or physical level of abstraction [2].

Circuits failing the manufacturing test are withdrawn from the production lines. Circuits that passed the manufacturing test and were sent to customers but failed in the field are often sent back to the manufacturer. Defective circuits in both categories can undergo further analysis to determine the cause of failure. Determining the cause of a failure may lead to manufacturing process adjustments to avoid producing more defective chips and increase the production yield. Physical Failure Analysis (*PFA*) is the process of collecting and analyzing data to determine the cause of a failure. It uses a wide array of methods, especially microscopy and spectroscopy. However, locating a sub-micron defect on a chip is a time-consuming and costly and often destructive process. To help pinpoint the location of the defect, logic diagnosis uses a model of the circuit and the defective chip responses to test patterns to determine parts, called *candidates*, of the circuit that can explain the erroneous chip responses. Logic diagnosis can provide candidates at circuit, core, gate, or transistor levels, depending on the provided model of the circuit. Diagnosis candidates must explain the behavior of the defective chip for both failing and passing test patterns. As the defective chip has been withdrawn from production line, the diagnosis time is not as critical as testing time, and extra sets of patterns can be created using ATPG to further help improve diagnosis accuracy and resolution.

With modern technologies, conventional fault models like stuck-at, transition, as well as layout-aware (e.g., bridging) fault models are becoming less effective for ensuring desired test and diagnosis quality levels. Indeed, digital ICs are commonly synthesized with pre-defined libraries of standard cells of various nature and complexity and conventional fault models only consider faults at the boundary of library cells. However, an increasing number of defects in circuits fabricated with the most recent manufacturing technologies occur at the transistor level, within the logic cell structures. They are called

*intra-cell* or *cell-internal* defects [3]–[5]. These defects are only covered fortuitously with conventional fault models, and hence not surprisingly, these defects are found to be the root cause of a significant fraction of test escape [6].

Cell-Aware (CA) test and diagnosis have been proposed recently to target those subtle defects in ICs with the highest product quality requirement [7]–[11]. The realistic assumption under this concept is that the excitation of a defect inside a cell is highly correlated with the logic values at the input pins of the cell [12], [13].

Cell-aware diagnosis methods, which can identify the location and type of cell-internal defects at the transistor level, are quality assessment solutions widely adopted today in industry [14]. Cell-aware diagnosis is valuable in the context of large, complex cells such as adders, multipliers, and multi-bit sequential elements. Even when a defect is known to be within a cell, finding defects in such a complex cell during PFA can be challenging and time-consuming, especially if the defect is exhibited for a specific test condition and undetected for some others. CA diagnosis shortens the lengthy investigation process by pinpointing a small subsection of the suspected cell. It improves results for diagnosis scenarios such as customer return analysis as well as volume diagnosis applications like yield analysis [13].

A preliminary step when performing CA test and diagnosis is to characterize each standard cell of a given library with respect to all possible cell-internal defects. Analog (SPICE) simulations are performed to identify which cell-internal defects are detected by which cell patterns. The simulation results are encoded in a cell-internal-fault dictionary or CA model (also referred to as CA fault model or CA test model in the literature) [15], [16]. One bottleneck of CA model generation is that it requires extensive computational efforts to characterize all standard cells of a library [17], [18]. Typically, the generation time of cell-aware models for few hundreds of cells may reach up to several months considering a single SPICE license. Reducing the generation run time of CA models and easing the characterization process is therefore mandatory to faster deploy the CA methodology on industrial IC designs and make it a standard in the qualification process of silicon products [19]. To this end, Machine Learning (ML) can be used to drastically accelerate the CA model generation flow.

# Contribution

The work of this thesis is focused on finding a new way to perform defect characterization of library standard cells and form their associated CA models. This new way should decrease the reliance on costly SPICE simulator licenses and decrease the characterization time of standard cells, which in turn decreases the cost of the required computational

power. This time reduction will help spreading the use of CA methodology across the industry, ultimately increasing the quality of test and diagnosis of digital ICs, thus helping the development of quality products.

In this work, we propose a method to characterize the defective behavior of standard cells using machine-learning techniques. The ML algorithms are used to replace the SPICE simulator to determine how a potential defect can be detected inside a standard cell.

To this end, we propose a machine-learning compatible representation of a standard cell, together with the cell-patterns to be applied to its input pins and the potential defects affecting it. The proposed matrix can represent any type of standard cells, combinational or sequential, as well as defects affecting them, transistor defects and layout-based defects.

Using the proposed matrix format, several machine-learning algorithms are used in a supervised training manner. Leveraging on existing CA models, ML algorithms were trained to determine if a given cell-pattern can detect a given cell-internal defect affecting a given standard cell type. Once trained, the ML algorithms are used to predict the defective behaviors of new standard cells, regardless of the technology or the transistor features size.

Machine-learning algorithms trained in a supervised manner are better for predicting the behavior of objects similar to the ones they were trained on. To obtain CA models for all standard cells in a new library of interest, we present a so-called hybrid flow, in which the new standard cell is analyzed and compared to the standard cells in the ML training database. Depending on the similarity between the new cell and the ones in the training database, the new cell will have its CA model generated using either ML prediction or analog simulation.

We present two ways of analyzing a new standard cell and comparing it to the cells in ML training database. The first one uses algorithmic rules and suffers from limitations and mis-decision rates. The second one is also based on a machine-learning algorithm and is much more accurate.

Following previously published works [16], [20]–[24] in which the research team proposed several learning-guided solutions for CA diagnosis of mission mode failures in customer returns, this thesis extends the previous works by dealing with sequential cells and diagnosis of related defects in customer returns. All solutions are based on a Bayesian classification method for accurately identifying defect candidates in standard cells of a defective IC. The selection of one solution over another depends on the considered test scenario (test sequence, test scheme, test conditions), thus leading to different types of targeted defects and failure mechanisms. New experiments done on a silicon test chip and a customer return have proven the efficacy of the flow in terms of diagnosis accuracy and

resolution for a given test scenario.

The research presented in this dissertation is validated by experiments on industrial circuits and libraries in production at STMicroelectronics.

To summarize, this thesis proposes the following contributions:

- We present a matrix format representation of any library standard cell, compatible with machine-learning techniques.
- We propose a method to use machine-learning techniques and presented matrix format representation to predict the defective behavior of a standard cell and obtain its CA model.
- We propose a hybrid-flow to deal with the intrinsic limitations of supervised learning techniques and obtain a CA model for every cell in a new cell library.
- We propose several analysis methods to orient new standard cells in the hybrid-flow. The most efficient method is based on a machine-learning algorithm.
- We propose extensions and modifications in the previously proposed flow for ML-based CA diagnosis of mission mode failures in customer returns.

# Outline

The work in this thesis is organized as follow:

Chapter 1 presents the context of test and diagnosis of digital ICs. The chapter introduces the notions of defects and fault models and present some of the tools used for ICs test and diagnosis.

Chapter 2 includes the state of the art of intra-cell defects test and diagnosis. The CA methodology is presented, along with its limitations and costs. The objectives of this thesis are then precisely described.

Chapter 3 presents the proposed ML-based CA defect characterization methodology and associated representation of standard cells and cell-internal defects along with experimental results.

Chapter 4 discusses the limitations of the proposed methodology and offers a solution, by introducing the so-called hybrid-flow and the ML-based decision block orienting the new standard cells through it.

Chapter 5 extends previous work on the use of machine-learning for CA diagnosis.

This method uses CA data to improve the precision of logic diagnosis and pinpoint the location of a defect inside a standard cell, which orients and speed up the Physical Failure Analysis phase.

Finally, Chapter 6 presents the conclusion and gives some perspectives for future research.

Chapter 1

# Context - Test and Diagnosis of digital ICs

A wide range of mechanisms can be the source of a circuit malfunction. Imperfections in the manufacturing process can alter the physical structure of transistors or wires in an IC. Depending on the nature of the alteration, it may cause the manufactured circuit to experience unexpected behaviors, in terms of logic behavior, speed or power consumption, inducing risks for the system in which the IC is embedded. Defects can also appear during the lifetime of the chip, due to wear-out mechanisms.

Manufacturing defects impact yield levels negatively. To assess the quality against manufacturing defects, all fabricated ICs are tested at the end of the manufacturing line. *Manufacturing test* exercise fabricated ICs, compare their responses with the expected ones. In case of a mismatch, the IC has failed the test, otherwise, it has passed it. As *manufacturing test* is applied in an industrial environment, it needs to be as quick as possible to satisfy economical constrains, while targeting the most potential defects to ensure products quality. Fault models are representation of the impact of potential defects on the circuit behavior and are used to determine the optimal sets of test patterns.

Once a chip has been marked as failing the manufacturing test, it is withdrawn from production line and analyzed to determine the cause of failure. Determining the cause of a failure may lead to modifications of the manufacturing process to avoid the production of more defective chips and increase the production yield. To help pinpoint the location of the failure, *logic diagnosis* uses an abstract model of the circuit and the defective chip responses to test patterns to determine *candidates* explaining the erroneous chip responses.

This chapter first presents the most common defect mechanisms that may appear during manufacturing and the fault models that abstract them. Section 1.2 then presents ATPG algorithms and the manufacturing test of digital ICs. Finally, section 1.3 introduces the diagnosis of digital ICs.

## 1.1 Defects and fault models

Due to the feature size of nanometric devices, their manufacturing is subject to imperfections. In particular, spot defects such as excesses or voids in the material, and parametric variations, such as variations of the transistor channel length or the wells doping, cause the fabricated circuit to deviate from the layout specification [25]. Such deviations from the planned layout may cause unexpected or erroneous behaviors of the circuit.

A *bridge or short defect* is an unwanted connection of two or more parts of the IC. A short can be caused by the deviation of the chip components leading to a contact, or by the deposition of superfluous material by the manufacturing equipment, or by the deposition of an outside particle during the chip manufacturing time. A bridge defect connects two nets that should have been independent. The effect of the bridge defect on the circuit behavior depends on the electrical property of the connection. A low-resistance bridge defect will force the affected nets to the same voltage value, whether it is the value of the strongest net or an intermediate value due to conflict between the nets' drivers. Low-resistance bridges may flip the logical value of the affected nets and affect the IC logic behavior. A high-resistance may not flip the value of nets but may cause delay in the propagation of logical values on affected nets, which may in turn affect the IC logic behavior and performances.

An *open defect* is a void or crack in the material which alters its physical properties [25]. Depending on its location and size, it may affect the IC behavior in several ways. A *full open* will disconnect its two parts entirely, leaving one of the node floating. The value of the floating node depends on its capacitance coupling with adjacent lines. This floating value may propagate down the rest of the circuit and affect the IC logic behavior. A *resistive open* may reduce the amount of current going through the affected element, which may introduce an additional delay in information propagation, ultimately affecting the IC logic behavior and performances.

Defects originate from many sources and can appear in any location of the IC layout. As their physical detection and observation is a costly and destructive process, the manufacturing test rather focuses on detecting the effect defects have on the behavior of ICs. A *fault model* is a representation of the effect of defects on IC behaviors. A fault model may be described at logic, circuit, or physical level of abstraction [2].

The *stuck-at model* assigns a fixed logical value to a victim line (sa0 ; sa1). A short between a net and power or ground can be represented by a stuck-at fault. However, it is not flexible enough to represent defects with sophisticated activation conditions. For many years, the stuck-at has been the state-of-the-art representation for test pattern generation or diagnosis.

More short defects can be described using one of the *bridge models* [26]. All bridge faults are activated by two lines being driven to opposite logic values. The resulting value of the lines depends on the bridge model assumptions. One of the lines may keep its value while forcing it to the other (*dominant bridge model*) or forcing a combination of the two values to the other (*dominant and bridge*). Both lines can also have their value changed by the bridge, matching a combination of their original values (*wired-and* and *wired-or bridges*). More sophisticated bridge models were proposed to represent the variety of defects' behaviors [27]–[30].

*Delay fault models*, such as *path delay fault*, *transition fault* and *small delay fault* are meant to represent the deviation of parts of an IC from their intended timing specification.

Defects affecting the transistors are represented by *transistor fault models*. A low-resistance short defect between drain and source terminal of a MOS transistor will lead the transistor to conduct no matter the value applied to its gate terminal. This behavior is represented by the *stuck-on fault model*. The *stuck-off model* abstracts a defective transistor that will never conduct any current. A stuck-off transistor may lead to a floating intra-cell net, which value depends on the surrounding nets and previous value of the net, creating a memory effect in the cell.

The presented fault models are some of the most used fault models to test and diagnose digital ICs. The next section will present how the ATPG uses fault models to obtain sets of test patterns.

## 1.2 Test of digital ICs

Test is the procedure of applying sequences of electrical stimuli (known as *test vectors* or *patterns*) to the Circuit Under Test (*CUT*) and compare the responses of the CUT with the expected results in terms of logic behavior, frequency, power consumption, etc. In case the chip responses do not match the expected one, the chip is marked as faulty. Test aims at detecting faulty devices as early as possible. The first step of test is the generation of sets of test patterns which will allow efficient detection of the defects in the chip. This is done by choosing one target fault model to represent the behaviors of the defects to detect. An Automatic Test Pattern Generation (*ATPG*) algorithm, such as the one illustrated in Fig. 1.1, will use an abstract model of the design and will assemble a set of test patterns that aims at activating and propagating as many of the considered faults as possible.

Applying a fault model to a design results in a list of faults to be detected. As digital ICs are commonly synthesized with pre-defined libraries of standard cells of various nature and complexity, the design is often abstracted at the gate level. Conventional fault models

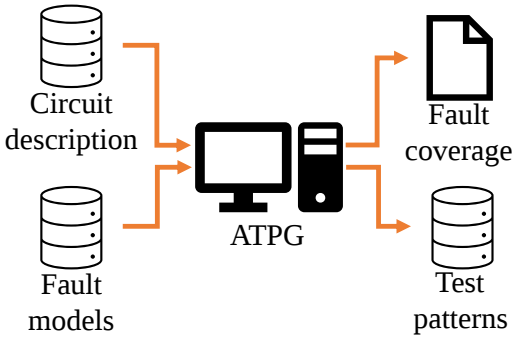



Figure 1.1: Generic schematic of an ATPG tool

only consider faults at the boundary of library cells to establish the fault list. ATPG algorithms use logic simulation of the design to choose input combinations that activate the faults and propagate their effects to a primary (or pseudo-primary) output. The *activation* of a fault implies assigning the component affected by the fault the opposite value than expected. The justification of the lines allows to set the correct values at the primary (or pseudo-primary) inputs so that the desired activation values is set at the fault location. *Propagation* of the fault effect ensures that the faulty value is transmitted to at least one primary (or pseudo-primary) output, which causes the test output to differ from the fault-free response. The ATPG aims at providing test patterns targeting every fault in the fault list. Once a test pattern has been determined by the ATPG algorithm, it is fault simulated against all remaining fault in the fault list. Faults detected by the newly formed test pattern are marked as detected and dropped from the fault list. This allows to decrease ATPG run-time and increase the added value of the test pattern. Numerous flavors of the ATPG algorithms exist. For example, N-detect ATPG aims at having at least N test patterns targeting each fault of the fault list [31], [32]

Several sets of test patterns can be assembled for a single design, for targeting different groups of faults and defects. Once the set(s) of test patterns is established, it is applied to the CUT. If the obtained responses deviate from the expected output patterns, the chip is identified as faulty and discarded.

Circuits are tested several times during their lifetime, to ensure quality and reliability of both the IC itself and the system it may be a part of. The first test is the manufacturing test, which sorts out failing devices affected by a manufacturing defect. Failing chips are discarded and sent to diagnosis and PFA teams for further analysis of the failure mechanism. A chip passing the manufacturing test is deemed valid and shipped to customer. Once integrated in a system, the chip undergoes *in-the-field* or *online* tests to detect any latent defect or degradation due to aging and wear-out mechanisms. Should an IC fail in the field, it is often sent back to the manufacturer for further analysis.

To maintain competitive manufacturing costs, the manufacturing test should be as quick as possible while uncovering a maximum of potential manufacturing defects and variations. The duration of a test is correlated with the number of test patterns in the test set. Hence test patterns should be carefully chosen and allow the detection of several faults. ATPG algorithms are in charge of determining the test patterns having the most added value. The test patterns selection is performed based on a fault model.

*Fault coverage* is one of the most used metrics to assess the quality of a test set. A test metric is a measure of the completeness of a test and is also used to guide test pattern generation [2]. Fault coverage measures the percentage of the fault list which is detected by a given test set. Fault coverage is estimated using fault simulation: the test set is simulated against all faults in the fault list. A test set coverage can be assessed on a fault list different from the one it was originally created on. This is particularly useful if one plans to apply several sets of test patterns on a given design. By dropping faults of the second fault list detected by the first patterns set, one eases the work of the second ATPG run and may decrease the number of patterns in the second set. As fault models are merely an abstraction of actual defects' behavior, even a patterns set achieving 100% fault coverage on a given model may not detect every potential defect on a chip. To guide test pattern generation and account for the wide variety of potential defects, numerous flavors of test metric have been proposed, such as the Single Stuck-at Fault (SSF) test metric, the N-detect test metric [31], [32], the transition test metric [33] and the TARO test metric [34].

## 1.3 Diagnostic of digital ICs

The objective of logic diagnosis is the identification of the location and nature of the underlying defects explaining the faulty behavior observed during test. The goal is to improve yield or performance. To help pinpointing the location of the defect, logic diagnosis uses the model of the circuit and the defective chip responses to test patterns to determine parts, called *candidates*, of the circuit that can explain the erroneous chip responses. Logic diagnosis can provide candidates at circuit, core, gate, or transistor levels, depending on the provided model of the circuit. Diagnosis candidates must explain the behavior of the defective chip for both failing and passing test patterns. As the defective chip has been withdrawn from production line, the diagnosis time is not as critical as testing time, and extra sets of patterns can be created using ATPG to help improve diagnosis accuracy and resolution.

Logic diagnosis can be performed following a cause-effect or an effect-cause approach [35].

Methods using the cause-effect approach rely on intensive fault simulation process. It records the test response of every fault under the applied test set and record them in a dictionary [36]. Depending on the design complexity, the number of faults and patterns may grow very large and it may be impossible to gather the dictionary for all faults. When a defective circuit is to be diagnosed, its failing test response is matched against the ones in the dictionary, which gives fault location candidates.

Methods using the effect-cause approach search in the design for fault location that can explain all failing patterns. Starting for the output pins that mismatch the defect-free circuit response, the structure of the circuit is back traced to mark as candidate every element that could have affected the output pins. By performing this back tracing on every test pattern, the number of candidates eventually decreases [37], [38]. The simplest approach consider that a single fault is responsible for all failing patterns [39]. This help reducing the number of candidates as this single fault has to be located in a place where it can affect all mismatched output pins. If several fault locations explain all failing patterns, the diagnosis tools may use the passing test patterns to differentiate between the candidates.

Two metrics usually evaluate the quality of a diagnosis results: accuracy and resolution. If the actual defect is reported as a candidate, then the diagnosis is accurate. The number of reported candidates defines the resolution of the diagnosis. The ideal case is then an accurate diagnosis with perfect resolution (i.e., one).

The set of candidates returned by the diagnosis step benefits to the Physical Failure Analysis (*PFA*). Knowing the location of the faulty component reduces the suspect area, which in turn decreases the time needed to locate the physical defect. Determining the nature of the defect may help to detect a systematic defect caused by a malfunction in the manufacturing process. Solving this malfunction will help increase the manufacturing yield.

# Chapter conclusion

Manufacturing ICs is a complex and error prone process. To ensure the production of quality and defect free products, every manufactured chip undergoes manufacturing test. The manufacturing test differentiates between passing ICs, that are then shipped to customers, and failing ICs that are discarded and sent to diagnosis and PFA teams for further analysis.

Automatic Test Pattern Generation (ATPG) algorithms employ logic simulation and abstract models of circuits' design and defects' behavior to determine the optimal sets of test patterns to detect as many potential defects as possible while keeping low the

test cost and time.

Diagnosis tools receive as input an abstract model of the circuit's design and the test responses of a failing IC. The diagnosis tools then return a list of fault candidates explaining the test fails. Physical Failure Analysis (PFA) uses this list of potential defect location to determine the cause of the failure and proposes adjustments to the manufacturing process to improve the yield.

Chapter 2

# State of the art of CA methodologies

As increasingly smaller geometries are used by the semiconductor industry, the traditional fault models, such as stuck-at and transition fault models, are no longer sufficient to efficiently detect potential defects that can appear in digital Integrated Circuits (ICs) during manufacturing. These 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

Indeed, these fault models only consider faults on interconnections, at the boundaries of standard cells used to synthesize digital ICs [3], [4]. However, with recent advanced technologies, more and more defects are found to be located inside the standard cells and are referred to as intra-cell or cell-aware defects. These manufacturing defects may not behave as conventional faults. Therefore, these defects are only fortuitously covered by traditional fault models and are often found to be the root cause of a significant fraction of test escape [6].

Previous works on test and diagnosis of intra-cell defects can be classified into two categories. Techniques in the first category extend the application of logic algorithms to deal with transistor defects [3], [4]. The main weakness of these techniques is the quality of the logic fault models that do not accurately describe the behavior of 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 to the logic values at the input pins of the cell [12], [13]. This assumption allows to use common gate-level ATPG algorithms to deal with intra-cell defects. Several methods have been proposed to guide ATPG work toward detection of intra-cell defects, such as the gate-exhaustive test metric [40], or the cell-aware fault model [12].

In the remaining, we call *cell-pattern* the logic values applied at the input pins of a given standard cell in a design, as opposed to test patterns which are applied to the primary (or pseudo-primary) inputs of the design. Both cell-patterns and test patterns are composed of one or several vectors of logic values applied consecutively.

To improve the test coverage of these subtle intra-cell defects, fault models and test metrics have been proposed. With *gate exhaustive* methods [2], the ATPG aims at applying every possible input combination at the input of every cell in a design, forcing activation of defects inside. *N-detect* methods use a classic fault model but force the ATPG algorithm to provide N different test patterns targeting each fault, which may increase the fortuitously coverage of non-targeted defects. With *embedded multi-detect* test [41], the ATPG algorithm objective is to target several times each fault without adding new test patterns to the set, by exploiting the test patterns don't care bits. These techniques suffer from limitations, the main one being the increased number of test patterns, leading to an increase of test time, making them inapplicable on real-world designs.

In [42] the authors present a new fault model to significantly increase the defect coverage of the test patterns generated by ATPG algorithms, called Cell-Aware (CA) fault model. This new fault model is meant to directly abstract transistor-level and layout-based intra-cell faults, allowing the targeting of the actual root causes of intra-cell defects. Using these fault models, CA test and diagnosis techniques have been developed to target these subtle defects [12].

Cell-aware methodologies also suffer from limitations, mainly the long step of CA characterization of standard cells. This is caused by an extensive computational effort in terms of run time, SPICE simulator licenses and CPU requirements, required to generate the CA fault models for every cell of a cell library or a design.

This thesis proposes an innovative method to characterize standard cells and obtains their CA models without relying on costly and time-consuming analog simulations. Instead, the methodology uses Machine Learning (ML) techniques for predicting the behavior of cell-internal defects. More widely, the aim is to use existing CA models from various standard cell libraries developed using different technologies to predict CA models for new standard cells independently of the technology.

The rest of this chapter is organized as follow. Section 2.1 presents several approaches to cover intra-cell defects with ATPG test patterns. Section 2.2 details the CA methodology and its heavy deployment process. Section 2.3 presents some of the improvements proposed in the literature to decrease CA methodology deployment time. Finally, Section 2.4 presents the goals and purposes of this thesis.

## 2.1 Targeting intra-cell defects

Increasing defect coverage of IC test is a never-ending task. This section presents some of the methods proposed in the literature to create sets of test patterns targeting defects inside library standard cells. These subtle defects may not behave like a stuck-at fault at

one of the standard cell input pins.

Techniques extending the application of logic algorithms to deal with transistor defects have been proposed for cell-level diagnosis [3], [4]. To diagnose a defect inside a standard cell, its transistor structure is abstracted as a netlist of logic gates, and logic diagnosis is performed on that netlist. The main weakness of these techniques is the quality of the logic models that do not accurately describe the behavior of transistor defects.

Extending ATPG algorithms to manipulate transistors and activate and propagate transistor-level fault has recently been discussed in [43]. The authors have proposed a method to model a standard cell's transistors netlist using ATPG transistor primitives. ATPG is then used to create sets of test patterns detecting transistor-level defects inside the standard cell. Due to the limitations of ATPG primitives for transistors, many adjustments of the cell's netlist are necessary. Moreover, this technique does not work properly for all defects, as some of them have an intrinsic analog behavior. Finally, the authors apply their method to test individual standard cells composed of a few to a few hundred transistors. Applying this method to a billion-transistor design will significantly increase the work of the ATPG algorithms.

Methods to obtain sets of test patterns targeting intra-cell defects while using a gate-level ATPG algorithm have been proposed. *Gate exhaustive testing* aims at creating a test set that applies all possible input combinations to each gate and observes the gate responses at an observation point such as a primary output or a scan cell [2]. As such, the gate exhaustive test does not use fault models to generate test patterns. Such test sets maximize the gate exhaustive test metric proposed in [40]. Assuming a single gate presents defect(s), all logical faults would be detected by such a test set. A gate exhaustive test set also effectively detects various kinds of faults such as bridging faults and transistor stuck-on faults [44]. However, gate exhaustive testing causes an important increase in the number of test patterns.

*Region exhaustive testing* is a pseudo-exhaustive testing technique [2], [45]. As such, it partitions the circuit into segments or regions, in a way that the number of inputs of every region is small enough (and significantly smaller than the number of primary inputs of the circuit). These reduced numbers of inputs allow to exhaustively test each segment. In [45], the authors consider static patterns only, leading to $2^N$ patterns for a region with $N$ input pins. Finding the optimum partitioning of the circuit is a NP-complete problem, and as such is computationally intensive. In [45], the authors propose a heuristic to partition the circuit into groups of gates having a single output. The goal of the algorithm is to create $N$ regions such that the total number of region-exhaustive faults is less-than-or-equal-to the total number of gate-exhaustive faults, while minimizing

the total number of region-exhaustive faults. Although not optimal, this heuristic makes the approach practical. The authors then show that exhaustive testing of a region cover a larger proportion of un-modeled defects, compared to patterns obtained using gate exhaustive or traditional stuck-at fault models.

None of region- and gate-exhaustive testing techniques use a fault model to guide the ATPG algorithms, they simply apply all possible pattern to every regions/cells. As such, they do not account for the specificity of the real standard cells, resulting in potentially redundant test patterns, leading to an increase of the length of pattern sets and test time. Cell-Aware fault models have been proposed to efficiently guide ATPG algorithms.

## 2.2 Cell-aware methodology

Cell-Aware (CA) test and diagnosis techniques have been introduced to target intra-cell defects [12]. These techniques rely on the realistic assumption that the excitation of a defect inside a standard cell is correlated with the logic values applied to its inputs [12], [13]. CA test is a subset of gate-exhaustive test, as it guides the ATPG to apply only cell-patterns known for targeting intra-cell defects, instead of all possible cell-patterns.

One of the first steps when deploying a CA methodology for a new technology is therefore to characterize each standard cell in the library with respect to all possible cell-internal defects. Non-resistive short and open defects are usually considered, but resistive ones can also be characterized if required by the application, and such for several resistance values. This CA characterization process is traditionally performed using analog (SPICE) simulations, which identify which cell-internal defect is detected by which cell-pattern. The simulation results are then organized into a cell-internal-fault dictionary called CA model (also known as CA test model) [15], [16].

The CA model dictionary maps cell-patterns to the intra-cell faults it detects. Faults having equivalent detection conditions are grouped into equivalence classes. Table 2.1 show a representation of a CA model dictionary as a detection table, for a 2 inputs (A&B) and 1 output (Z) AND2 cell. Each row of the table accounts for a possible cell-pattern. Each column of the table accounts for a class of intra-cell faults. At a given row-column intersection, a '1' means that the cell-pattern detects the fault (i.e., if the fault is present in the standard cell, then applying the cell-pattern will results in a defective output value). A '0' indicates non detection of the fault by the cell-pattern. For cells with several outputs, other non-zero values are used to indicate which output pin(s) will be affected by the fault, allowing gate-level ATPG to propagate the fault effect to one circuit's primary (or pseudo-primary) output.

As can be seen in Table 2.1.b, CA dictionaries can also contain and describe intra-

Table 2.1: Example of CA model dictionary as a detection table for (a) statically detected faults and (b) dynamically detected faults

(a)

| A | B | Z | D1 | D2 | D3 | D13 | D15 | D25 | D26 | D29 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

(b)

| A | B | Z | D4 | D5 | D6 | D8 | D10 | D35 | D40 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| R | 1 | R | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | R | R | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| F | 1 | F | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | F | F | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

cell faults detected by cell-patterns composed of several vectors of logic values, applied at-speed at the input pins of the cell. Such cell-patterns are called dynamic cell-patterns. In Table 2.1.b, dynamic cell-patterns are composed of two vectors. An 'R' represents the *Rising* transition of a pin going from logic state 0 to logic state 1. Similarly, an 'F' is a *Falling* transition from 1 to 0. Faults detected by dynamic cell-patterns are called dynamically detected faults. The binary representation of a CA model dictionary points toward its compatibility with machine-learning and is one of the motivations for this work. This point will be detailed in Chapter 3.

In CA models, intra-cell defects are abstracted with transistor-level faults or layout-based faults. There are two types of transistor-level faults: shorts and opens. A short on a transistor connects two of the transistor terminals. This is modeled by adding a low-value resistor between the two terminals nets. Note that a transistor stuck-on fault can be modeled by a short between the transistor drain and source terminals. Transistors with three terminals (drain, source, gate) can be affected by 3 different shorts. If the transistor bulk connection is considered, 6 potential shorts exist. An open on a transistor terminal disconnect the terminal from its driving net. This is modeled by adding a high-value serial resistor between the terminal and its driving net. Note that a transistor stuck-off fault can be modeled by an open at drain (or source) terminal. Transistors with three terminals (drain, source, gate) can be affected by 3 different opens. Figure 2.1 illustrates transistor-level faults.

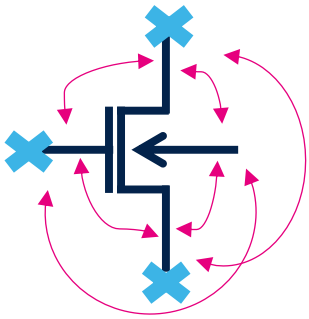


Figure 2.1: Potential transistor-level faults affecting a standard cell' transistor

Layout-based intra-cell faults are based on the actual layout implementation of a standard cell. For example, two or more nets could be connected if their polygons are neighbors on the layout and a defect or a variation, exceeding DRC safety rules, appears at manufacturing time. Such a defect would be abstracted by a short and modelled by adding a low value resistor between the incriminated nets. Methods to algorithmically locate potential layout-based defects have been proposed in [3]. By modifying the DRC rules, one can use existing DRC tools to locate adjacent polygons that could be connected by a defect.

Cell-aware models are used by ATPG to generate test and diagnosis patterns. Gate-level ATPG algorithm do not have to be heavily modified to work with CA models. Activation of fault is done by setting the nets connected to the cell input pins to the desired values, as indicated in the CA model dictionary. The fault effect must be propagated from the net connected to the cell affected output(s) to a primary (or pseudo-primary) output. Setting a net to a value and propagating a fault effect to output are basic ATPG features, so few modifications are needed to enable CA ATPG.

Cell-aware models describe the defective behavior of standard cells as autonomous object. They do not take into account the actual placement and surroundings of the cells inside ICs. For this reason, the CA models are generated during the creation of the standard cell library and can be used for every instance of the standard cell in a design. So, from the design of circuits point of view, the CA characterization of standard cells is an offline process which do not affect the steps to create a new product.

The typical CA model generation flow is represented in Fig. 2.2. Using the SPICE netlist representation of a standard cell, which is usually derived from a layout description, e.g., a GDSII file written in DSPF (Detailed Spice Parasitic Format), an electrical simulator simulates each potential defect against an exhaustive set of stimuli containing static and dynamic cell-patterns. From these simulation results, all cell-internal defects are classified into defect equivalent classes with their detection information (which cell-pattern detects which defect).

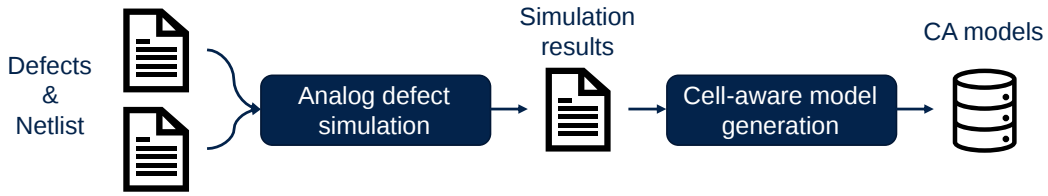



Figure 2.2: Simulation-based cell-aware model generation flow

As standard cells may have more than 10 inputs and thousands of standard 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.

## 2.3 Limitations and improvement trials

A main limitation while deploying CA methodologies for a new technology or a new cell library is the extensive computational effort required to generate the CA models for every cell of the library [17], [18]. All possible cell-patterns have to be simulated against all potential cell-internal faults. For a cell with $n$ input pins, it exists $2^n$ different static cell-patterns, $n * 2^n$ different dynamic cell-patterns if we limit ourselves to cell-patterns with only transition on one input pins, $4^n - 2^n$ if we allow multi-transitions dynamic cell-patterns. For a cell with $m$ transistors, it exists $9m$ transistor-level faults (6 shorts and 3 opens per transistor). This characterization effort needs to be done for every cell type in the technology or design of interest, often in the thousands. This leads to a substantial number of analog simulations to be performed.

During CA characterization, cells are independent from one another, so it is possible to run simulations in parallel. Regardless of the chosen run strategy, CA models generation by analog simulations has a cost in terms of run time, SPICE simulator licenses, CPU requirements and disk usage.

Due to this cost, a CA methodology can be partially deployed (e.g., applied for a restrained selection of cells), or deployed with inaccuracies. One tolerated inaccuracy is to obtain CA models for cells at one given PVT (e.g., the most typical) and to use it for cells operating at all PVTs. CA models for a given standard cell at different PVTs exhibit a few percentages of difference. Table 2.2 presents the percentage of common values in CA models of a same cell for different PVTs, averaged over 20 cells. 'SS' stands for the most pessimistic PVT corner (Slowest NMOS, Slowest PMOS, lowest Voltage, lowest Temperature), 'TT' for the most typical and 'FF' for the most optimistic. Note that the matrix is slightly non-symmetric due to faults being detected at one PVT but not at the other. As can be seen from Table 2.2, up to 5% of inaccuracy are introduced

by using CA model from the most typical PVT corner.

Table 2.2: Average percentage of common values in CA models for different PVTs

| Other \ Reference | SS | TT | FF |
|---|---|---|---|
| SS | | 95% | 93% |
| TT | 94% | | 97% |
| FF | 93% | 97% | |

This inaccuracy however is tolerated by the industry as it does not affect much, if at all, the quality of test patterns obtained using such CA models. This tolerance of the CA ATPG flow to such levels of inaccuracy points toward its compatibility with machine-learning and is one of the motivations for this work. This point will be detailed in Chapter 3.

Solutions to reduce the CA models generation effort has been proposed in the literature. In [17], the authors aim to decrease CA library characterization run-time by decreasing the number of analog simulations performed during characterization. In each cell, a structural-analysis algorithm determines defect-pattern $(d; p)$ pairs that are logically undetectable. A cell-internal defect $d$ is logically undetectable if it is located in a part of the cell that is inactivated by cell-pattern $p$. Those pairs are excluded from characterization, decreasing the number of simulations. In their implementation, the authors generate a graph representing the cell's active path, marking nets as nodes and transistors as edges. The defect is injected inside the graph: (i) short defects add an edge and (ii) open defects remove an edge. Then, the un-detectability criteria is checked: (i) short defects are un-detectable if there is no path between VDD and VSS nodes and (ii) open defects are un-detectable if the output node is connected to VDD or VSS. Results on GPDK045's 476 cells (sequential and combinational) have shown a 47% reduction in characterization time (~6h of analyze + 1033h of simulation for proposed method versus 1955h for simulation only)

The authors of [43] offer to decrease the reliance on analog simulations to obtain CA models. Their proposal is to model the standard cells at the transistor-level using ATPG primitives. Then, ATPG tools can be used to determine the cell-patterns detecting intra-cell defects. By working on a standard cells basis, one keeps the ATPG run time constrained. Standard cells may have internal structures that ATPG primitives do not model. To handle this limitation, the authors of [43] establish a list of special structures and propose models for them by combining existing ATPG primitives. However, intra-cell defects may exhibit subtle analog behaviors. For this reason, the authors classify the defects as *hard-* and *soft-detected*. Hard-detected defects are detected at the cell output

regardless on the physical parameters of the cell. On the other hand, soft-detected defects detection depends on the physical parameters of the cell, such as the relative driving strength of two transistors driving the output at opposite values. For the soft-detected defects, the authors of [43] rely on analog simulations to settle the matter.

In [6], the authors modify the CA models to improve CA test in term of pattern count. This decreases the time and cost of IC CA testing. In [6], the authors modify the CA model detection table to allow *don't care bits*. Indeed, CA model detection tables, such as the ones presented in Table 2.1, tend to indicate the values to apply on every input pin of the standard cell, although some may not be of concern. For example, in Table 2.1.a, defect 'D29', can be detected by applying $AB = 01$ or $AB = 11$, so only the value of input B matters and the value of input A 'don't care'. By allowing the 'don't care' $X$ value, the authors decrease the constrains on the ATPG tool, which can be exploited during cell-to-chip expansion to increase fault coverage and reduce test-pattern count.

## 2.4 PhD Objectives

Cell-aware test and diagnosis is a powerful tool to improve ICs quality and reliability. However, the high deployment cost of the method slows down its deployment in the industry.

The goal of this work is to proposes an innovative method to characterize standard cells and obtains their CA models without relying on costly and time-consuming analog simulations. Instead, we propose to use Machine Learning (ML) techniques for predicting the behavior of cell-internal defects. More widely, the aim is to use existing CA models from various standard cell libraries developed using different technologies to predict CA models for new standard cells independently of the technology. This new way should decrease the reliance on costly analog simulator licenses and decrease the characterization time of standard cells. This time improvement will help spreading the use of CA methodology across the industry, ultimately increasing the quality of test and diagnosis of digital ICs, helping the development of quality products.

Machine learning is a type of Artificial Intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so [46]. A ML learning algorithm is fed with input data, called *features*, and outputs values, which interpretation depends on context. In supervised learning, a ML algorithm is trained to produce an expected answer from the provided features. To this end, a set of $(features; expected\,output)$ values is necessary, and called the *training dataset*. The training phase consists in a loop that feeds the training features to the ML algorithm, observes its output values and modify the ML algorithm until the provided

output values match with the expected ones, to a certain quality criteria. After completion of training, the ML algorithm can be used to *predict* the output values for different features. Machine-learning algorithms trained in a supervised manner offer abstraction and generalization capabilities. Many types of ML algorithms exists such as Linear models [47], Support Vector Machines (SVM) [48], Nearest Neighbors models (NN) [49], Gaussian Processes [50], Naive Bayes (NB) [51], Decision Trees [52], and Neural network models [53].

In this work, we propose a method to characterize the defective behavior of standard cells using machine-learning techniques. The ML algorithms are used to replace the analog simulator to determine how a potential defect can be detected inside a standard cell. To this end, we propose a ML-compatible representation of a standard cell, the cell-patterns applied to its input pins and potential defects affecting it. The proposed matrix can represent any type of standard cells, combinational or sequential, as well as defects affecting them, both transistor defects and layout-based defects.

Using the proposed matrix format, several ML algorithms are used in a supervised training manner. Leveraging on existing CA models, ML algorithms are trained to determine if a given cell-pattern can detect a given cell-internal defect affecting a given standard cell type. Once trained, the ML algorithms are used to predict the defective behavior of new standard cells, regardless of the technology or the transistor features size.

# Chapter conclusion

This chapter has presented the state of the art of intra-cell defects test. In order for ATPG patterns to target explicitly the subtle defects hidden inside the standard cells composing digital ICs, several methods have been proposed. Some methods extend logic simulation methods to allow ATPG to work on transistor-level netlists. Other methods keep the ATPG working at the gate-level and guide the ATPG to apply specific values at the input pins of a standard cell to detect its intra-cell defects.

Cell-aware methodology is part of the second category. For each standard cell in a cell library or a design, a CA fault model is generated. This CA fault model contains a dictionary indicating which cell-pattern detects which cell-internal defect. The ATPG uses this information to create sets of test patterns targeting the cells internal defects.

Although efficient, the CA methodology comes at a high computational cost. To lower that cost, this thesis proposes a method based on machine-learning techniques to replace the costly analog simulations by prediction of standard cells defective behaviors.

Chapter 3

# Machine learning for CA models generation

Based on the aforementioned, improving the generation run time of CA models and ease the characterization is required to faster deploy the CA methodology on industry designs and make it a standard in the qualification process of silicon products [19]. To this purpose, the objective of this work is to develop a methodology for predicting the behavior of cell-internal defects using Machine Learning (ML). More widely, the aim is to use existing CA models from different standard cells and technology nodes to predict CA models for new libraries independently of the technology node. To the best of our knowledge, **this is the first work to address this problem** since previous works on ML focused on cell library characterization without defect injection [54]–[56].

In this chapter, we present an efficient methodology to obtain CA models using ML techniques. More specifically, the ML algorithms are used to predict if a given pattern stimulus applied to the input pins of a given type of standard cell affected by a given cell-internal defect can or cannot detect this defect (i.e., flip the standard cell's output pin digital value). In this sense, the ML algorithms replace the SPICE simulators.

The major objective of this chapter is to describe standard cells, their intra-cell defects and cell-pattern input stimuli applied to their input pins, in a ML compatible format. To this end, a matrix representation of a standard cell is proposed. Each line of this matrix contains a description a standard cell, of a potential cell-internal defect affecting it and an input stimulus applied to the cell input pins. Each line represents the question "Does this stimulus detects this defect inside this cell?" asked to the ML algorithms. The ML algorithms are to provide a Yes/No (Boolean) answer to those questions.

Leveraging on the existing CA models data available at STMicroelectronics, the ML algorithms are trained in a supervised manner. In supervised learning, the training dataset contains both questions and expected answers. The ML algorithms are initialized randomly and answer the questions. The training routine compares the provided answers

with the expected one, then modify the ML algorithms in a way to increase the accuracy of the ML algorithms. Once the models reach the desired accuracy on the training dataset, the models can be used to predict the answers for new questions.

The rest of the chapter is organized as follows. Section 3.1 motivates the work and presents the proposed ML-based CA defect characterization methodology. Sections 3.2 and 3.3 describe how cell transistor netlists and cell-internal defects are represented and manipulated by the proposed methodology. Section 3.4 presents the modifications made to account for the specificities of sequential and complex standard cells. Section 3.5 and 3.6 report some experimental results and comparison with a simulation-based approach.

## 3.1 Learning-Based CA Defect Characterization

### 3.1.1 Motivation

The motivation behind the use of ML for defective cell characterization is the result of several observations made while performing comparisons between several CA models coming from different libraries and technology nodes:

- Several cell internal defects, such as stuck-open defects, are independent of technology node and transistor size [57], [58].
- For the same function, two cell schematics are quite similar for two different technology nodes.
- 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 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 we 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 technology nodes.
- A large database of CA models is available and can be used to train a ML algorithm.

All these observations intuitively indicated that CA model generation through ML

could be possible. However, the first challenging task was 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 was therefore mandatory to standardize the description of cells and corresponding defects for our ML-based defect characterization methodology. Heuristic solutions developed to this purpose are described in Sections 3.2 and 3.3. The second challenging task was to find a way to represent all these information / input data so that they can be ML friendly. A matrix description of cells and corresponding defects was chosen to this purpose.

### 3.1.2 Proposed Defect Characterization Methodology

The proposed learning-based defect characterization methodology is used to predict the behavior of a cell affected by intra-cell defects, hence avoiding costly analog defect simulations. The proposed flow is sketched in Fig. 3.1. It is based on supervised learning that takes a known set of input data and known responses (labeled data) used as training data (*'CA matrices'* obtained from *'CA models'* in Fig. 3.1), trains a model (*'Random Forest Classifier'*) to classify those data, and then uses this model to predict (infer) the class of new data (*'CA matrices without defect detection status'* obtained from cells description files).

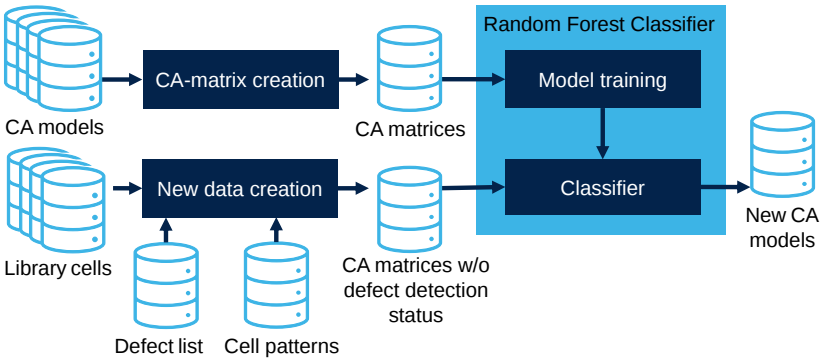



Figure 3.1: Generic view of the ML-based defect characterization flow

**Training data** are made of various and numerous CA models formerly generated by relying to brute-force analog 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. Cells with the same number of inputs and having the same number of transistors are grouped together to form the Training dataset.

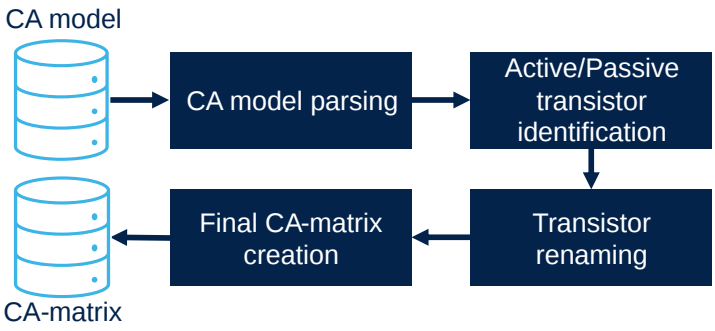



Figure 3.2: Generic view of the CA-matrix creation flow

The CA-matrix creation flow is depicted in Fig. 3.2. The flow starts by rewriting the CA model so that it can be ML friendly (*'CA model parsing'*). Then, it identifies the activation conditions of each transistor with respect to input stimuli (*'Active/Passive transistor identification'*). Once the activation conditions for each transistor have been identified, *Transistor renaming* is done. This is a critical step in this flow since it allows the usage of the training data across different libraries and technologies. Finally, the CA-matrix is created with the above information.

Table 3.1 shows an example of a training dataset for an AND2 cell. It is composed of four types of information:

- *Cell inputs & responses*: This gives the values applied on input (A, B) as well as the cell response on output Z. As can be seen, the test pattern sequence provides all the possible input stimuli that can be applied to a standard 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).

- *Transistor switching activity*: 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. Columns "About Defect" contain extra information, such as the defect's type and name, in human-readable format, for monitoring purposes.

- *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 3.1: 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. They are obtained for each standard cell from cell description, corresponding list of defects and cell patterns. The format of a new data instance is similar to 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. Note that, as for training data, new data are grouped together according to their cell input number and transistor number, so that inference can be done at the same time for cells with the same number of inputs and transistors.

Figure 3.1 depicts the two main steps of the supervised learning process used for ML-based CA model generation. In this work, we use a Random Forest Classifier for predicting the class of each new data instance. This choice comes from the results obtained after experimenting 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 has to 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 Standard cell representation in the CA-matrix

In this section, we detail the various steps required to take a cell from a transistor-level (SPICE) netlist and represent it in the CA-matrix in a standardized and ML friendly manner. This description must be accurate enough to clearly identify each transistor and each net of the cell transistor schematic. Moreover, this description must also be able to associate each transistor to its sensitization patterns and to report the output response for

each pattern. For this reason, the cell description process requires a number of 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

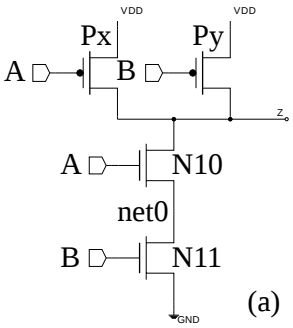
The first step consists in identifying active and passive transistors in the cell netlist with respect to an input stimulus. This information is used to represent the transistor netlist in the matrix format. To this end, we perform a single defect-free (golden) electrical simulation of each cell to be characterized, so as to identify active and passive transistors for each input stimulus (test pattern) and to measure the corresponding cell output value. An active NMOS (resp. PMOS) transistor is a transistor with a logic-1 (resp. logic-0) value measured on its gate terminal. A passive NMOS (resp. PMOS) transistor is a transistor with a logic-0 (resp. logic-1) value measured on its gate terminal. Note that a Verilog simulation, with a CDL (Circuit Description Language) netlist that should be written using NMOS and PMOS primitives, can replace the single defect-free electrical simulation. With this information, we can associate each cell pattern 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, we use the '-' character before the PMOS values.

Let us consider the transistor schematic of an NAND2 cell as shown in Fig. 3.3.a. In the partial representation of the CA-matrix of the cell shown in Fig. 3.3.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 and two passive NMOS transistors.

This matrix representation of the cell schematic is strongly dependent on the transistor names and the order they are defined 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 avoid this problem, a second step in the cell description process is required. This step consists in renaming all cell transistors independently of their initial names and order in the input SPICE netlist. The algorithm developed to this purpose is detailed in the next subsection.




| 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)

Figure 3.3: Example standard cell NAND2: (a) cell transistor schematic and (b) partial CA-matrix representation

### 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, which drives the gates of the next branch. 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.

Let us consider the example schematic in Fig. 3.4. 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 (Ninv|Pinv). 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. 3.4 is therefore ((1n&(1n|1n))|1n).

*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.

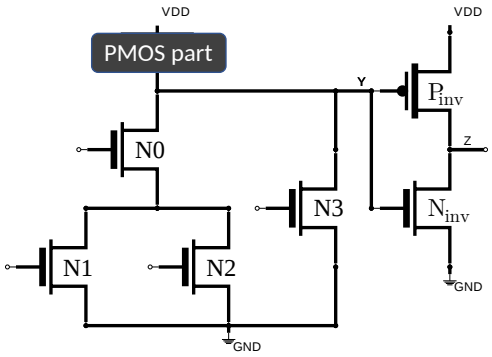



Figure 3.4: Example schematic

### 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. 3.4, 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, we cannot accept such confusing situation. 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). We define the activity value 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, we need to know whether the transistor is active or passive for each input stimulus. We already have this information in the CA-matrix as described in Section 3.2.1. To illustrate this process, activity values for the transistors of the NAND2 cell in Fig. 3.3.a are given in Table 3.2.

Finally, transistors of each branch are sorted by increasing activity value to give the final description of the cell in the CA-matrix. For our example in Fig. 3.3, 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.

Table 3.2: Activity values for the NAND2 cell in Fig. 3.3.a

| | | | **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** |

## 3.3 Defect Representation in the CA-Matrix

In this section, we detail 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. In order to describe these defects, all transistor terminals are listed as a column in the CA-matrix (cf. Table 3.1). 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, we use the same representation mechanism as for intra-transistor defects.

Table 3.3 is an example of defect description in the CA-matrix of the example NAND2 cell in Fig. 3.3. 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 3.3: NAND2 cell in Fig. 3.3: Example of defect columns

| **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 |

## 3.4 Modifications for sequential and complex cells

The learning-based CA model generation as proposed in previous sections is used to predict the behavior of a combinational cell affected by intra-cell defects. Modifications to training and new data allow to use the same flow to predict the behavior of a sequential cell affected by intra-cell defects.

*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. Sequential cells with the same number of inputs and having the same number of transistors are grouped together to form the Training dataset.

The CA-matrix creation flow is similar to the one in Fig. 3.2. The flow starts by rewriting the CA model of a given cell in a library so that it can be ML friendly. This step is called CA model parsing. Then, the flow identifies the activation conditions of each transistor inside the cell with respect to input stimuli. Once the activation conditions for each transistor have been identified, transistor renaming is done. This is a critical step in this flow since it allows the usage of the training data across different libraries and technologies. Finally, the CA-matrix for the considered cell is created with the above information.

In order to illustrate the various steps of the CA-matrix creation flow, let us consider the block-level representation of a scan flip-flop as depicted in Fig. 3.5. It consists of three main blocks (MUX, MASTER latch and SLAVE latch) plus two transmission gates. It has four inputs (D, TI, TE, CLK), one virtual input (Q-), and one output (Q). The virtual input represents the value loaded in the flip-flop before applying the test stimulus.

Table 3.4 shows an example of a training dataset for the scan flip-flop in Fig. 3.5. It is composed of four types of information:

- *Cell inputs & outputs*: This gives the values applied on inputs (D, CLK, TE, TI, Q-) of the cell as well as the cell response on output Q. The test pattern sequence

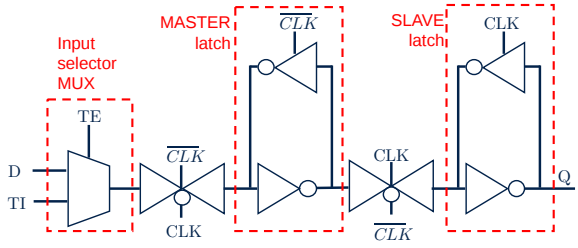



Figure 3.5: Block-level representation of an example scan flip-flop

provides all the possible input stimuli that can be applied to the cell. For the sake of readability, they are represented partially in Table 3.4. These stimuli must also be efficient to detect sequence-depending defects like stuck-open defects. For this reason, a six-valued logic algebra made of 0, 1, R, F, P and A 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). P (resp. A) represents a Pulse 010 (resp. Anti-pulse 101) and is used for the input clock signal of the cell.

- *Transistor Switching Activity*: This indicates the activation conditions of each transistor (e.g., N0, N1, etc. for NMOS transistors, and P0, P1, etc. for PMOS transistors) in the cell schematic. Each transistor can be in one of the following states: active, passive, switching to active state, switching to passive state, pulsing, anti-pulsing.

- *Defect description*: This gives information about all defect locations in the cell transistor schematic. In Table 3.4, "N1_D" stands for "defect on the drain of transistor N1", "N1_G" stands for "defect on the gate of transistor N1", and so on. The name and type of each defect are also given in this description.

- *Defect detection*: This is the class of the data sample (the output of the ML classifier). A value '1' ('0') means that the defect is detected (undetected) by the input pattern at the corresponding output of the cell.

The first three types of information are used as inputs for the ML algorithm.

Table 3.4: Example of training dataset for a scan flip-flop with a non-inverting output. the cell has four physical input pins: data (D), clock (CLK), test enable (TE), test input (TI), and a virtual input (Q-) which correspond to the previous state of the output pin (Q)

| Cell inputs & outputs | | | | | | Transistor switching activity | | | | | | Defect description | | | | | | | About defect | | | Defect detection |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | CLK | TE | TI | Q- | Q | N0 | N1 | ··· | P0 | P1 | ··· | N1_D | N1_G | N1_S | ··· | P3_D | P3_G | P3_S | name | type | | fZ |
| 0 | P | 0 | 0 | 0 | 0 | P | 0 | ··· | P | 1 | ··· | 0 | 0 | 0 | ··· | 0 | 0 | 0 | free | free | | 0 |
| R | P | 0 | 1 | 0 | R | P | R | ··· | P | F | ··· | 0 | 0 | 0 | ··· | 0 | 0 | 0 | free | free | | 0 |
| 0 | P | 0 | F | 1 | 0 | P | 0 | ··· | P | 1 | ··· | 0 | 0 | 0 | ··· | 0 | 0 | 0 | free | free | | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ |
| 0 | P | 0 | 1 | 0 | 0 | P | 0 | ··· | P | 1 | ··· | 1 | 0 | 1 | ··· | 0 | 0 | 0 | D15 | short | | 1 |
| F | P | 0 | 1 | 1 | F | P | F | ··· | P | R | ··· | 0 | 0 | 0 | ··· | 0 | 0 | 1 | D47 | open | | 1 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ |

*New data* represent the cells to be characterized and are obtained for each sequential standard cell from the cell description, corresponding list of defects and cell patterns. The format of a new data instance is similar to 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.

### 3.4.1 Sequential cell representation in the CA-matrix

In this section, we detail the various steps required to take a sequential cell from a transistor-level (SPICE) netlist and represent it in the CA-matrix in a standardized and ML friendly manner. This description must be accurate enough to clearly identify each transistor and each net of the cell transistor schematic. Moreover, this description must also be able to associate each transistor to its sensitization patterns and to report the output responses for each pattern. For this reason, the cell description process requires a number of successive operations that are detailed below. Note that this process is applied to all sequential cells in a library to be characterized.

#### 3.4.1.1 Identification of Active, Passive and Pulsing Transistors

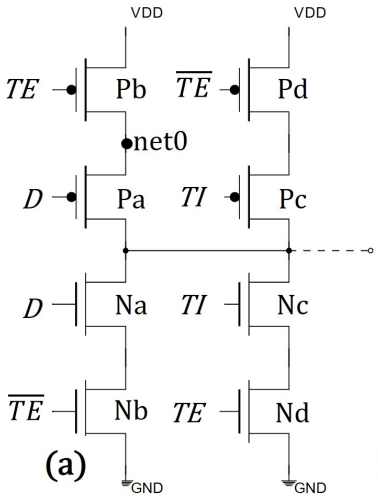
The first step consists in identifying active, passive, and pulsing transistors in the cell netlist with respect to an input stimulus. This information is used to represent the transistor netlist in the matrix format. To this end, we perform a single defect-free (golden) electrical simulation of each cell to be characterized, so as to identify active, passive and pulsing transistors for each input stimulus (test pattern) and to measure the corresponding cell output values. An active NMOS (resp. PMOS) transistor is a transistor with a logic-1 (resp. logic-0) value appearing on its gate terminal during application of the test pattern whose duration is one clock cycle. A passive NMOS (resp. PMOS) transistor is a transistor with a logic-0 (resp. logic-1) value appearing on its gate terminal during application of the test pattern. 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. Note also that a Verilog simulation, with a CDL (Circuit Description Language) netlist that should be written using NMOS and PMOS primitives, can replace the single defect-free electrical simulation. After this step, the CA-matrix contains the following columns:

- *Cell inputs & outputs*: They contain all input stimuli (test patterns) that can be applied to the sequential cell, and the corresponding responses.
- *Transistor switching activity*: They contain six possible values indicating if the

transistor is active (1), passive (0), switching from an active state to a passive one (F), switching from a passive state to an active one (R), pulsing (P) and anti-pulsing (A). Since PMOS and NMOS transistors are activated in opposite way, we use the '-' character before the PMOS values.

Figure 3.6.a represents the transistor schematic of the MUX block inside the scan flip-flop depicted in Fig. 3.5. In the partial representation of the CA-matrix of the scan flip-flop shown in Fig. 3.6.b, columns D, TE, TI list all the possible input stimuli for the MUX. For each stimulus, active, passive, and pulsing information about each transistor of the cell are entered in the CA-matrix. For example, [D,TE,TI]=[000] leads to three active PMOS transistors and three passive NMOS transistors.

Figure 3.7 represents the transistor schematic of the MASTER latch inside the scan flip-flop depicted in Fig. 3.5. In the partial representation of the CA-matrix of the scan flip-flop shown in Fig. 3.7.b, columns D and CLK list all the possible input stimuli. Note that, in this example, it is assumed that input D has been selected by signal TE of the MUX in Fig. 3.5, so that it drives the MASTER latch.




| D | TE | TI | Na | Nb | Nc | Nd | Pa | Pb | Pc | Pd |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | -1 | -1 | -1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | -1 | -1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | -1 | 0 | -1 | -1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | -1 | 0 | 0 | -1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| R | 0 | R | R | 1 | R | 0 | -F | 0 | -F | 1 |

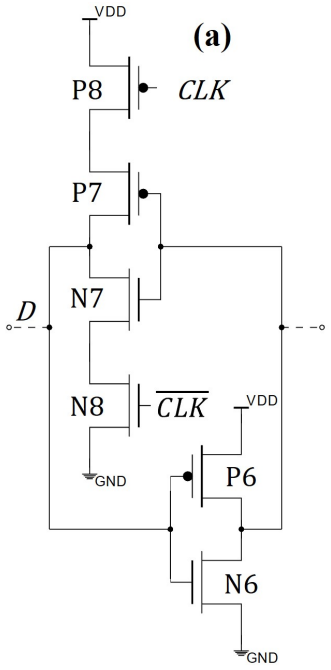
(b)

Figure 3.6: MUX of the scan flip-flop in Fig. 3.5: (a) transistor schematic and (b) partial CA-matrix representation

As with combinational cells, transistors need to be renamed, in order not to rely on SPICE names, The algorithm is similar to the one presented for combinational cells in Section 3.2.2. The differences and additions to account for sequential cells specifics are detailed in the next subsection.

#### 3.4.1.2 Renaming of Transistors

*Determination of branch equations.* The definition of branches is identical to the one for combinational cells. However, sequential cells, such as the one presented in Fig. 3.5,




(b)

| D | CLK | N6 | P6 | N7 | P7 | N8 | P8 |
|---|---|---|---|---|---|---|---|
| 0 | P | 0 | -1 | 1 | 0 | A | A |
| 1 | P | 1 | 0 | 0 | -1 | A | A |
| ... | ... | ... | ... | ... | ... | ... | ... |
| R | P | R | -F | F | -R | A | A |

Figure 3.7: MASTER latch of the scan flip-flop in Fig. 3.5: (a) transistor schematic and (b) partial CA-matrix representation

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.

Let us consider the example schematic in Fig. 3.8. The structure is composed of three 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 the branch output and the power nets, so its branch equation is (Ninv|Pinv). The equation of the left-most branch (PMOS branch driving net X) is (P3|(P0&(P1|P2))). 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 PMOS branch driving net X in Fig. 3.8 is therefore (1p|(1p&(1p|1p))). These two branches are separated by the third branch, i.e., a transmission gate composed of N4 and P4, which is anonymized as '1t'.

*Sorting of branch equations.* Once all the branch equations for the considered se-

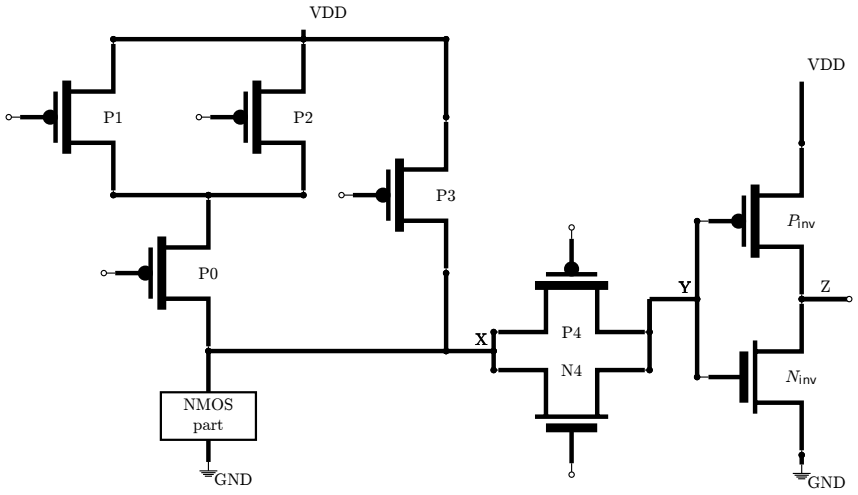



Figure 3.8: Example schematic

Table 3.5: Branch equations for the schematic of Fig.3.8

| Level | Number of transistors | Anonymized equation | Comment |
|---|---|---|---|
| 1 | 2 | (1n\|1p) | Inverter |
| 2 | 2 | 1t | Transmission gate |
| 3 | 4 | (1p\|(1p&(1p\|1p))) | PMOS structure |

quential cell have been determined, they are sorted by using the same deterministic criteria used for combination cells. Table 3.5 reports all the sorted branch equations of the schematic in Fig. 3.8.

#### 3.4.1.3 Identification of Parallel Transistors

Identifying branch equations is not enough to rename all transistors of a given cell. Extending the method presented for combinational cells, an activity value is computed for every transistor of the sequential cell. This value summarizes the states of the transistor (active, passive, pulsing) for all possible stimuli applied to the cell. The input stimuli range from (P,0…,0) to (P,1…,1). For each of these stimuli, the transistor is either active (1), passive (0), pulsing (P) or anti-pulsing (A). We define the activity value as a word made of 0, 1, P and A, in which the first symbol corresponds to the state of the transistor when the first stimulus is applied, second symbol for second stimulus, and so on for the whole stimuli range.

To illustrate this process, activity values for the transistors of the MUX illustrated in Fig. 3.5 are given in Table 3.6. Finally, transistors of each branch are sorted by their activity values (alphabetical order) to give the final description of the cell in the CA-matrix. For the MUX in Fig. 3.5, the renaming process is detailed in Table 3.6.

Table 3.6: Activity values for the mux in Fig. 3.5

| | | | old names | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| D | TI | TE | Na | Nb | Nc | Nd | Pa | Pb | Pc | Pd |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| Activity value | | | 0011 | 1111 | 0101 | 0000 | 1100 | 1111 | 1010 | 0000 |
| | | | ↓ Renaming ↓ | | | | | | | |
| | | | N1 | N3 | N2 | N0 | P4 | P3 | P1 | P0 |

## 3.5 Experimental results for combinational cells

We implemented our method in a python program. The ML algorithms were taken from the publicly available python module called scikit-learn [59]. Our 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)). Investigations on CA models generated by commercial tools showed that up to 5% difference exist between CA models for different PVT corners. As mentioned earlier, an industrial test flow can tolerate this difference. From this data, we chose 97% as quality criterion, meaning that the ML-predicted CA models should have less than 3% difference compared to the electrical simulated ones to be considered as valid, to remain under the tolerated 5%. All these cells already had a CA model generated by a commercial tool. We generated the CA-matrix for each cell. The method was experimented in two separate ways. First, the ML algorithm was trained and evaluated using cells belonging to one technology. Second, we trained the model on one technology and evaluated it on another one.

### 3.5.1 Predicting defects' behavior on the same technology

We first trained the ML algorithm 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 a given group, we trained the ML algorithm 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 this work, we considered all possible open and short defects (static and dynamic) in each cell. Results presented below report the prediction for open defects. Results achieved for short defects are similar.

Table 3.7 presents the prediction accuracy achieved for open defects. Results are reported for cells with less than 7 inputs and 48 transistors. 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 green background indicates that the maximum prediction accuracy in this group is 100%, i.e., the ML algorithm 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 circled box in Table 3.7. We have 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%, (iii) the average prediction accuracy over all 24 cells is 99.97%. These results show that the ML algorithm 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.

### 3.5.2 Predicting defects' behavior on another technology

We also conducted experiments on cells belonging to two different technologies. Evaluation was slightly different compared to the previous one. Here, the ML algorithm 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 3.8 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 3.7. After investigating on this point, we 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 in Section 3.5.3.

To verify the efficiency of our method when different transistor sizes are considered, we trained the ML algorithm over the 28SOI standard cells and used it to predict the behavior of C40 cells. Table 3.9 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.

### 3.5.3 Analysis - Discussion

We first analyzed cells for which the defect characterization methodology gives excellent prediction accuracy as well as those for which the prediction accuracy was quite low. Then, we investigated the limitation of the proposed method for CA model generation.

Table 3.7: Average prediction accuracy for cells in the same technology

| Number of transistors | Prediction accuracy (%) | Number of inputs 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 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 | 99.7 | 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 | 99.7 | 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 3.8: Average prediction accuracy for cells in a different technology

| Number of transistors | Prediction accuracy (%) | Number of inputs 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 6 | 98.21 | 99.47 | | | |
| | 8 | 94.56 | 96.86 | 99 | | |
| | 9 | | | | | |
| | 10 | 94.69 | 96.01 | 99.27 | | |
| | 12 | 87.73 | 98.05 | 99.1 | | 99.76 |
| | 14 | 85.69 | 97.35 | 98.75 | | |
| | 16 | 91.74 | | 99.2 | | |
| | 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 3.9: Average prediction accuracy for cells with different transistor sizes

| Prediction accuracy (%) | Number of inputs 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| Number of transistors 6 | 98.21 | 99.47 | | | |
| 8 | 94.56 | 96.86 | 99 | | |
| 9 | | | | | |
| 10 | 94.69 | 96.01 | 99.27 | | |
| 12 | 87.73 | 98.05 | 99.1 | | 99.76 |
| 14 | 85.69 | 97.35 | 98.75 | | |
| 16 | 91.74 | | 99.2 | | |
| 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 | | | | | |

After running several experiments on different configurations using one fault model at a time, we 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 very similar one. The difference between very similar cells is always the same and is represented in Fig. 3.9. More precisely, cells giving good results are always composed of one of the configurations presented in Fig. 3.9 and at least one cell of the training dataset contains the other configuration. The difference between these two transistor configurations is the presence or absence of the red net. The logic function of these configurations is the same. These configurations are mostly found in high-drive cells.

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

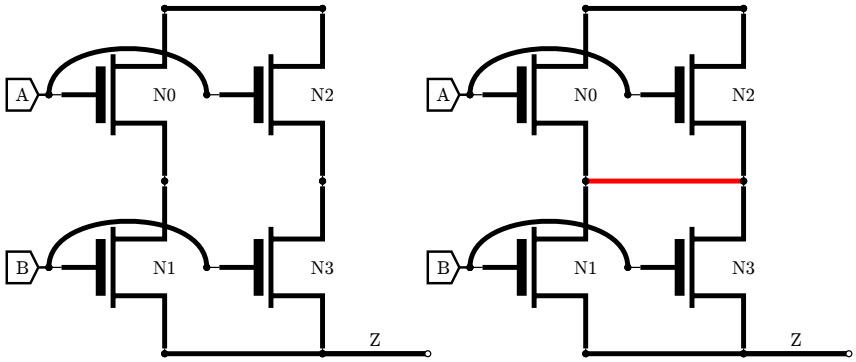



Figure 3.9: Typical transistor configurations leading to good prediction

training dataset, or (ii) a transistor configuration which is completely new when compared to cells in the training dataset.

## 3.6 Experimental results for sequential cells

Our python program was expanded with support for sequential cells. Our dataset was composed of 219 sequential cells coming from standard cell libraries developed using three technologies (C40 (27 cells), 28SOI (108 cells) and C28 (84 cells)). All these cells already had a CA model generated by a commercial tool. We generated the CA-matrix for each cell. The method was experimented in two separate ways. First, the ML model was trained and evaluated using cells belonging to one technology. Second, we trained the model on one technology and evaluated it on another one.

### 3.6.1 Predicting defects' behavior on the same technology

We first trained the ML model on a subset of cells of C40 standard cell libraries. As mentioned earlier, cells were grouped according to their number of transistors and inputs. For m cells available in a given group, we trained the ML model 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 4.5 cells. In this work, we considered all possible open and short defects (static and dynamic) in each cell. Results presented below report the prediction for short defects. Results achieved for open defects are similar.

Table 3.10.a presents the prediction accuracy achieved for short defects. Results are reported according to the number of transistors and number of inputs of each cell. 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. As can be seen, we always obtain the maximum prediction accuracy (100%) for each group, i.e., the ML model can perfectly predict the defective behavior of all cells in each group. This means that the CA model generated by ML fit the real behavior

achieved with electrical simulations.

The above cell category with good prediction score has been analyzed manually to identify why it led to good results. The analysis showed that all these cells have at least one cell in the training dataset with the same transistor structure (i.e., same structural pattern) or a very similar one. These results show that the ML model can accurately predict the behavior of a sequential 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.

### 3.6.2 Predicting defects' behavior on another technology

We also conducted experiments 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 3.10.b shows the prediction accuracy achieved on short defects of the 28SOI cells after training on the C28 cells. Results are averaged over all cells in each group.

Similarly, we conducted experiments to analyze the efficiency of our method when different transistor sizes are considered. This time, we trained the ML model over the C40 standard cells and used it to predict the behavior of more technologically advanced 28SOI cells. Table 3.10.c shows the prediction accuracy achieved on short defects.

In the above two scenarios, the average prediction accuracies are globally very low (around 50%), thus demonstrating the inappropriateness of our method. After investigation, we observed that in most cases, functionally-equivalent sequential cells in libraries from different technologies were designed differently and hence do not have neither the same transistor structure nor a very similar one. A 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. Considering the main property of our learning method for CA model generation, which is based on the recognition and use of identical structural patterns, it is not surprising to get such low-quality results.

### 3.6.3 Controlled experiments

In an attempt to check the above conclusion, we performed "controlled experiments" by considering three scan flip-flops (SDFPQ cells) coming from three different technologies (C40, 28SOI, C28). We manually modified the SPICE description of each cell so as to

Table 3.10: Average prediction accuracy for (a) cells in the same technology; (b) cells in different technologies; (c) cells with different transistor sizes

| Prediction accuracy (%) | | Number of inputs | | | |
|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 |
| Number of transistors | 32 | 100 | | | |
| | 34 | | | | |
| | 36 | | | | |
| | 38 | | | | |
| | 40 | | | 100 | |
| | 42 | | | 100 | |
| | 44 | | | 100 | |
| | 46 | | | 100 | |
| | 48 | | | 100 | 100 |
| (a) | 50 | | | | 100 |
| | 52 | | | | 100 |

| Prediction accuracy (%) | | Number of inputs | | | |
|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 |
| Number of transistors | 32 | | 53 | | |
| | 34 | | 50 | | |
| | 36 | | 50 | 54 | |
| | 38 | | 50 | 51 | |
| | 40 | | | 50 | |
| | 42 | | | 50 | |
| | 44 | | | | |
| | 46 | | | | |
| | 48 | | | | |
| (b) | 50 | | | | |
| | 52 | | | | |

| Prediction accuracy (%) | | Number of inputs | | | |
|---|---|---|---|---|---|
| | | 4 | 5 | 6 | 7 |
| Number of transistors | 32 | | | | |
| | 34 | | | | |
| | 36 | | | | |
| | 38 | | 50 | | |
| | 40 | | | 50 | |
| | 42 | | 48 | 50 | |
| | 44 | | | 50 | |
| | 46 | | | 48 | |
| | 48 | | | 48 | 57 |
| (c) | 50 | | | | |
| | 52 | | | | 58 |

get the same schematic for all of them. This was done by removing some buffers and duplicate transistors, which were initially inserted in the cell descriptions for driving strength purpose. After modification, each flip-flop contained 5 inputs and was made of 32 transistors. We successively considered all open and short defects in each SDFPQ cell.

We performed three types of experiments. First, we train the ML model by considering all short defects of the C28 SDPFQ cell, and successively evaluate its prediction accuracy over all short defects (576) of the C40 SDPFQ cell and over all short defects (928) of the 28SOI SDPFQ cell. The same procedure was done for open defects (the C40 SDPFQ cell and the 28SOI SDPFQ cell each contain 387 open defects). Next, we train the ML model by considering all short defects of the 28SOI SDPFQ cell, and successively evaluate its prediction accuracy over all short defects of the C40 SDPFQ cell and over all short defects (1016) of the C28 SDPFQ cell. Again, the same procedure was done for open defects (the C28 SDPFQ cell contains 394 open defects). Finally, we train the ML model by considering all short defects of the C40 SDPFQ cell, and successively evaluate its prediction accuracy over all short defects of the 28SOI SDPFQ cell and over all short defects of the C28 SDPFQ cell. The same procedure was done for open defects.

To visualize the efficiency of our ML method, we generated a confusion matrix in which each row of the matrix represents the instances in a predicted class (defects that are predicted by the ML algorithm to be detected / not detected by a given input pattern) while each column represents the instances in an actual class (defects that are actually detected / not detected by a given input pattern). By this way, the confusion matrix reports the number of true positives, false positives, false negatives, and true negatives.

Results are reported in Table 3.11 for the three types of experiments. The confusion matrix can be found at the center of the table (columns "True P" to "True N"), this time represented using only a horizontal axis. For example, let us consider the first experiment, when we train the ML model by considering all defects of the C28 SDPFQ cell, and evaluate the prediction accuracy over all defects of the 28SOI SDPFQ cell (third row in Table 3.11). The number of true positives, false positives, false negatives, and true

Table 3.11: Results of the controlled experiments

| Train | Predict | Defect type | True P | False P | False N | True N | Accuracy |
|---|---|---|---|---|---|---|---|
| C28 | C40 | short | 248 | 40 | 52 | 236 | 84% |
| | | open | 215 | 8 | 2 | 162 | 97% |
| | 28SOI | short | 442 | 23 | 40 | 423 | 93% |
| | | open | 215 | 2 | 2 | 168 | 98% |
| 28SOI | C40 | short | 252 | 36 | 59 | 229 | 83% |
| | | open | 215 | 8 | 2 | 162 | 97% |
| | C28 | short | 483 | 27 | 77 | 429 | 89% |
| | | open | 221 | 2 | 3 | 168 | 98% |
| C40 | 28SOI | short | 416 | 49 | 153 | 310 | 78% |
| | | open | 215 | 2 | 8 | 162 | 97% |
| | C28 | short | 446 | 64 | 182 | 324 | 75% |
| | | open | 221 | 2 | 9 | 162 | 97% |

negatives is 442, 23, 40 and 423 respectively, thus leading to a prediction accuracy of 93%. From the overall results reported in Table 3.10, this time it appears that the prediction accuracy achieved with our method ranges from 75 to 98%, thus clearly demonstrating its efficiency. As for combinational cells, one or more structural patterns have been identified in functionally-equivalent cells from various libraries, so that the ML algorithm can exploit them efficiently for training and inference purpose.

## 3.7 Comparison of results for combinational cells and sequential cells

Table 3.12: Comparative results – average prediction accuracy - combinational vs sequential cells

| Train & predict | Combinational Cells | Sequential Cells |
|---|---|---|
| Same technology<br>-<br>Same transistor size | 99.9% | 100% |
| Different technology<br>-<br>Same transistor size | 94.7% | 50.9% |
| Different technology<br>-<br>Different transistor size | 96.8% | 50.3% |

The same experiments have been done in Section 3.5 and Section 3.6 for generating CA models by using ML for combinational cells and sequential cells, respectively. The dataset of combinational cells was composed of 1712 standard cells coming from cell libraries developed using three technologies (C40 (446 cells), 28SOI (825 cells) and C28 (441 cells)). For sequential cells, the dataset was composed of 219 sequential cells coming

from standard cell libraries developed using the same technologies (C40 (27 cells), 28SOI (108 cells) and C28 (84 cells)).

In both cases, the ML-based method was experimented in two separate ways. First, the ML model was trained and evaluated using cells belonging to one technology. Second, we trained the model on one technology and evaluated it on another one. For combinational cells, the main results are the following:

- When predicting defects' behavior on the same technology, the prediction accuracy was close to perfect (100%) for most of the cells,
- When predicting defects' behavior on two different technologies considering the same transistor size, the behavior of most of the cells (70% of cells) was accurately predicted (accuracy $> 97\%$),
- When predicting defects' behavior on two different technologies considering different transistor sizes, 80% of cells were accurately predicted (accuracy $> 97\%$).

For sequential cells, predicting defects' behavior on the same technology gives close to perfect (100%) prediction accuracy for all cells. However, when predicting defects' behavior on two different technologies, the prediction accuracy is quite low (~50%).

A summarizing view of these results is proposed in Table 3.12, in which the average prediction accuracy for all experiments in Section 3.5 is reported and compared with the average prediction accuracy obtained with the ML-based CA model generation flow proposed in Section 3.6 for sequential cells. It can be seen from these results that using a learning-based flow for CA model generation is definitively appropriate when predicting defects' behavior on the same technology. Conversely, when different technologies are considered, i.e., train on cells from a given technology and infer for a cell in a different technology, relying only on a learning-based flow for CA model generation is not appropriate. A solution to address this issue is proposed in the next chapter.

# Chapter conclusion

In this chapter, we have presented a methodology to obtain CA models using machine-learning techniques, using ML algorithms to predict the defective behavior of combinational and sequential cells. In this sense, the ML algorithms replace the SPICE simulators.

The main contribution of this chapter is a ML compatible format description for the standard cells, their intra-cell defects and cell-pattern input stimuli applied to their input pins. We called the proposed representation the CA-matrix. Each line of this matrix contains a description a standard cell, of a potential cell-internal defect affecting it and

an input stimulus applied to the cell input pins. Each line represents the question "Does this stimulus detects this defect inside this cell?" asked to the ML algorithms. The ML algorithms are used to provide a Yes/No (Boolean) answer to those questions.

The ML algorithms are trained in a supervised manner. Once trained, the ML algorithms are used to predict the answers for new questions.

Experiments performed over 1600 combinational cells and over 200 sequential cells have demonstrated that the proposed ML-based CA characterization flow is efficient at predicting the defective behavior of a new (a.k.a., not in the ML training database) if the new cell has some resemblance with standard cells which are part of the ML training database. This limitation of the proposed methodology makes sense in the context of supervised learning (the ML algorithm infer from what it has "learned", anything too different is uncharted territory). Enriching the ML training database with new types of standard cells will help further improve the ML algorithms.

The next chapter will propose a flow to leverage the proposed ML-based CA-models generation tool, while ensuring quality of CA models for standard cells incompatible with ML-based CA characterization in the current state of the ML training database.

Chapter 4

# Hybrid flow for CA models generation

In this chapter, we present a methodology to leverage the presented machine-learning based methodology to generate high-quality Cell-Aware models for all cells in a cell library. Considering the results presented in Chapter 3, it appears that the presented method suffers from some limitation for complex cells.

This chapter proposes an analysis and discussion of previous chapter results and presents some properties of the cells achieving low prediction accuracy with the ML-based CA-model generation flow.

Considering that analysis, a mixed solution, which consists in combining ML-based CA model generation and conventional (simulation-based) CA model generation — called the *hybrid flow* — is introduced and tested.

The rest of the chapter is organized as follows. Section 4.1 presents the so-called hybrid-flow and its structural-analysis based decision block. Next, Section 4.2 details the improved hybrid flow's decision block based on machine-learning. Finally, Section 4.3 presents some experimental results using the hybrid-flow.

## 4.1 Hybrid flow presentation

Considering the above analysis and results, it appeared that the ML-based CA model generation cannot be used to characterize all cells in a standard cell library. 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.

We propose to use the hybrid flow sketched in Fig. 4.1 for accelerating the CA model generation. Typically, when the CA model for a new cell is needed, this new cell is analyzed against cells being used to train the ML algorithm. This is done by analyzing

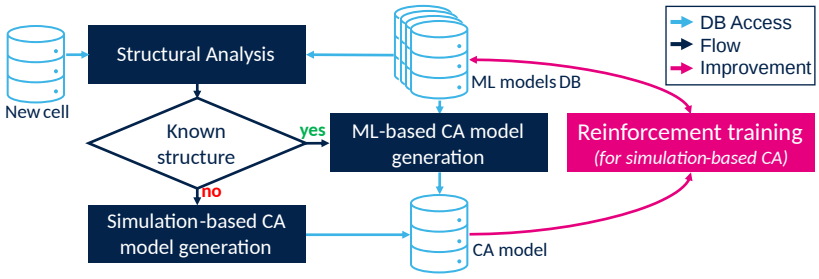



Figure 4.1: Hybrid flow for CA model generation

the structure of the new cell and check whether the training dataset contains a cell with identical or similar structure (as presented in 3.5.3). If the new cell's behavior is expected to be accurately predicted by the ML algorithm, the cell goes through the ML part of the flow and its CA model is quickly generated. The output information is then parsed to the desired file format. Otherwise, to ensure quality of the CA model, the cell goes through the simulation-based part of the flow. A feedback loop uses this new simulated CA model to supplement the training datasets and improve the ML algorithm for further prediction.

### 4.1.1 Runtime reduction results

To estimate the improvement in CA model generation time achieved with the flow in Fig. 4.1, we performed the following experiments. We trained the Random Forest model 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 3.5.3) 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, we can estimate the reduction in generation time to 99.7%. Now, if we consider 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, we observed that the ML-based CA model generation works well for about 80% of cells of the C40 subgroup. Surprisingly,

Table 4.1: Average prediction accuracy per defect type for a population of C40 cells

| | **Open defects** | | | **Short defects** | | |
|---|---|---|---|---|---|---|
| | **Source** | **Drain** | **Gate** | **S-D** | **S-G** | **G-D** |
| **Known structure** | ∼97% | ∼97% | ∼90% | ∼99% | ∼93% | ∼81% |
| **Unknown structure** | ∼87% | ∼87% | ∼78% | <60% | <60% | <60% |

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 our flow, which is the subject of the next section.

## 4.2 Improvement using ML-based decision block

To exploit all the possibilities of a CA model generation based on ML, a new decision block with major novelties has been developed. This new decision block replaces the one based on the structure analysis of cells presented in Section 4.1. Its objective remains the same, i.e., orienting a standard cell through the hybrid flow by choosing if the CA model for the new cell should be simulated or can be predicted by ML. A comparison of performance of the new decision block versus the previous one is proposed in Section 4.3.1. Figure 4.2 is a schematic of the updated hybrid flow with the new decision block.

### 4.2.1 Fine-graining

The first major novelty relies on a new paradigm. Indeed, the decision block of 4.1 considered each standard cell as atomic, meaning that a cell is characterized either through a ML-based CA model generation or through a simulation-based CA model generation. However, we observed that inside each cell, the prediction accuracies of each defect type vary. Table 4.1 illustrates this phenomenon. It contains averaged prediction accuracy for several types of cell-internal defects. The row labelled as "known structure" reports the values of accuracy for cells which have at least a cell with an identical or similar structure in the standard cells database. The row labelled as "unknown structure" reports values for the other cells.

From Table 4.1, we noticed that (i) the presence or absence of a cell with an identical/similar structure in the ML training group has a real impact on the prediction accuracy, confirming the result from [60]; and (ii) in some cases, a single badly predicted intra-cell defect type brings down the prediction accuracy of the entire cell under the quality criterion, preventing the cell from using a ML-based CA model generation. To alleviate this issue, fine-graining has been introduced in this work to mitigate this effect. Rather than considering the entire cell as a candidate for ML-based CA model generation, several combinations of (cell; cell-internal defect type) are considered independently. All

components of the hybrid-flow have been modified to allow such fine-graining during the training and prediction or simulation phases, and then merging the results in a single CA model.

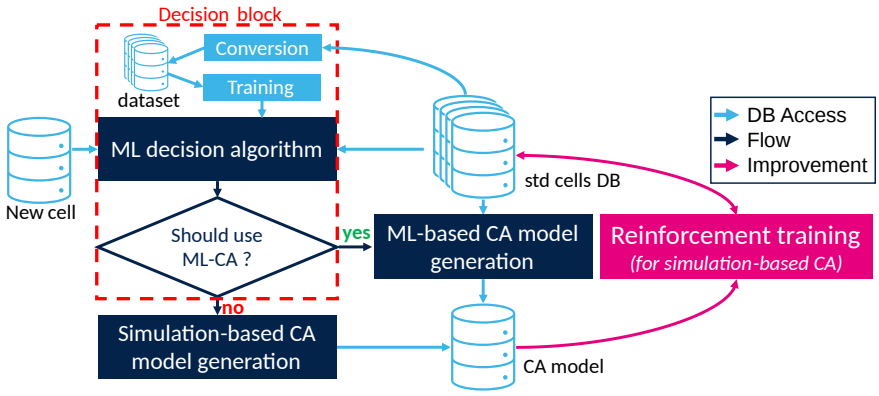



Figure 4.2: Updated hybrid flow with the new decision block based on machine learning

The decision block now analyzes every combination of (cell; defect type) and decides which ones should have their behavior predicted by ML and which ones should be electrically simulated. After generation, the predicted and simulated results are merged in the CA model of the cell. This fine-graining allows to use of the ML algorithm more extensively, thus decreasing the number of analog simulations, and ultimately decreasing the run-time and cost. Note that the quality criterion is still used on a (cell; defect type) combination basis rather than simply on the cell basis, meaning that a badly predicted defect type will no longer prevent the cell to be entirely ML predicted, i.e., all eligible defect types will go through ML while the remaining ones will go through simulation. From now on, any % of population should be understood as a % of the total number of (cell; defect type) combinations. Moreover, every decision by the decision block is taken on a (cell; defect type) basis.

### 4.2.2 Presentation of ML-based decision block

The second major novelty of the new decision block is the use of ML. As already pointed out, this block is in charge of orienting (cell; defect type) combinations to the ML-based CA model generation or to the simulation-based CA model generation parts of the flow. The decision is taken by a new ML algorithm which is independent from the one used in the ML-based part of the hybrid flow. This ML-based decision algorithm (now referred to as ML-decision) should decide whether or not each (cell; defect type) combination should go through the ML-based part of the flow depending on what is available in the standard cells database. This new ML-decision algorithm uses supervised learning and its specificities regarding the training phase will be presented hereafter.

Let us first consider the usage of such a ML-based decision block. To be oriented, the

structure of the new cell is computed and compared to the ones available in the standard cells database. The number of times the new cell's structure is present in the standard cells database is used by the ML-decision algorithm. The ML-decision algorithm is bound to answer the question "Considering the content of the standard cells database, should this (cell; defect type) combination use the ML-based CA model prediction part of the flow?". A 'yes/no' Boolean answer is expected. To this end, the ML-decision algorithm receives the following information as inputs:

- The considered cell name (string)
- The considered defect type (string)
- The considered cell function (string)
- The considered cell drive strength (float)
- The number of transistors of the cell (integer)
- The number of inputs of the cell (integer)
- The cell structure (string)
- The presence or absence of the cell structure in the standard cells database (Boolean)
- The number of times the cell structure is present in the standard cells database (integer)

### 4.2.3 Generating a dataset for ML-based decision block

The supervised training of this ML-decision algorithm is the most challenging part of this work. The creation of a dataset for the training and evaluation of the ML-decision algorithm should be carefully considered, as it should account for the numerous situations that the ML-decision algorithm may encounter in the field. The dataset is a list of samples, each of them contains the information given as inputs to the ML-decision algorithm along with the expected 'yes/no' answer. The dataset allows both the training and the evaluation of the ML-decision algorithm.

The dataset should prepare the ML-decision algorithm to orient the (cell; defect type) combinations according to what is available on the standard cells database at the time of decision. In a real industrial application, the content of the standard cells database depends on the historic of the company (e.g., electrically simulated CA models anterior to the deployment of the proposed methodology) and on the historic of the hybrid-flow usage, i.e., the simulated CA models obtained using the proposed methodology and added to the standard cells database by the improvement loop introduced in Fig. 4.2. To deal with this

changing database, the ML-decision algorithm should have a fast-training phase so it can be quickly trained every time the database changes or be compatible with reinforcement training. Our chosen ML-decision algorithm is again a RFC, which can be trained in a fraction of second over the dataset. Creating the first version of the dataset by scanning the standard cells database takes a few minutes. Updating the dataset because a new cell has been added to the standard cells database by the reinforcement loop takes a few seconds (which is negligeable compared to the simulation-based CA model generation run time). All these timings make the usage of the ML-decision algorithm realistic and appropriate.

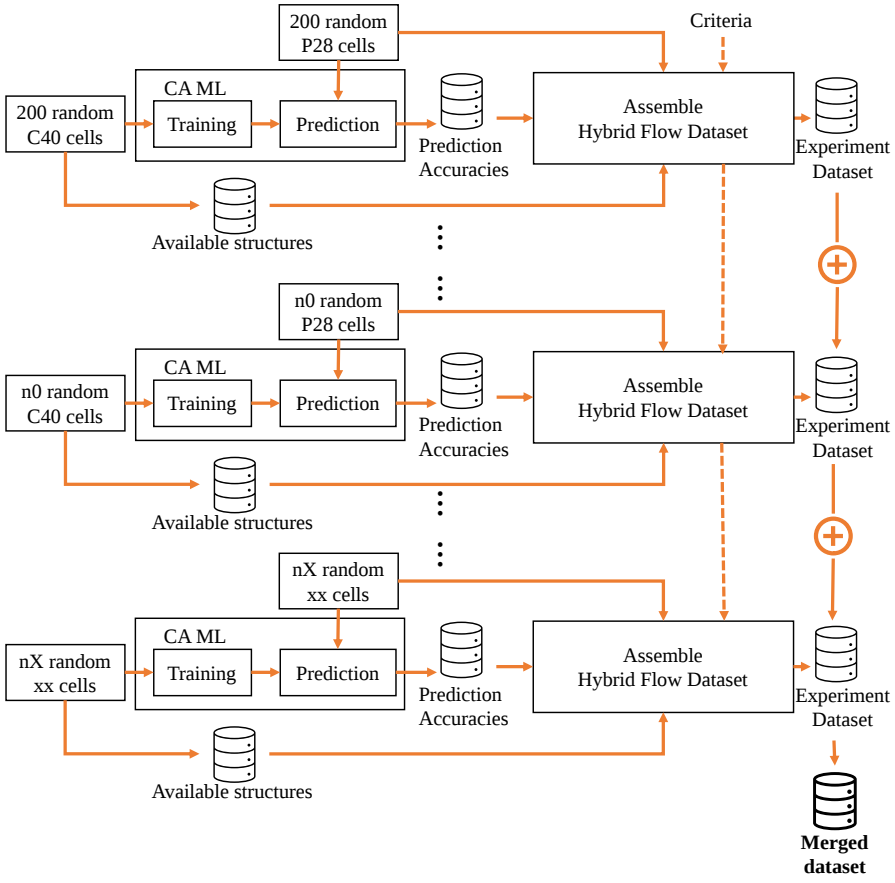



Figure 4.3: Process to create a dataset for the ML-decision algorithm. It is composed of several independent experiments with changing parameters. The results of all experiments are merged in a main dataset.

To create a dataset accounting for a wide variety of industrial usages and standard cell databases, the process depicted in Fig. 4.3 has been proposed. This process is composed of several experiments, each of them represents a different industrial setting and standard cells database. Each experiment randomly chooses cells to constitute its own standard cells database, then randomly choose cells for which it will generate ML-CA models. The experiment then generates CA models for the chosen cells using its standard cells database and assess the quality (accuracy) of the prediction. Note that

in the framework of this study, every cell has a CA model which has been obtained by simulation and acts as a reference when evaluating the prediction accuracy of the ML-based CA model generation. The accuracy of the prediction for each (cell; defect type) combinations is compared to a threshold (similar to the quality criterion presented in Section 3.5). This comparison gives an answer to the (re-phrased) question: "Will this (cell; defect type) combination be correctly predicted, considering the content of the standard cells database?". A prediction accuracy higher than the threshold leads to a 'yes', while lower accuracy leads to a 'no'.

Using the content of the cells database and the 'yes/no' answer, the experiment's dataset is assembled. The top experiment of Fig. 4.3 presents one experimental example where 200 cells belonging to the C40 technology are chosen to be in the cells database, while CA models will be predicted for 200 cells belonging to the P28 one. Figure 4.3 also shows the process where the experiments are repeated a large number of times with variation in the technologies and number of cells. The main merged dataset is the concatenation of all experiments' datasets. This process leads to a dataset which contains a large number of samples for various (cell; defect type) combinations with different standard cells databases to represent the variety of industrial usage. The correctness of our ML-decision algorithm is evaluated using this dataset and the results are presented in the next section.

## 4.3 Experimental results

This section presents experimental results using standard cells coming from five technologies. In the first sub-section, the process of Fig. 4.3 is used to obtain several datasets, which correspond to realistic industrial situations. The new ML-decision block is evaluated using these datasets and its performances are compared with the structural decision block. The second sub-section uses the CA models generated by the hybrid flow with an ATPG to generate test patterns for several IC designs and shows that these models offer similar results and thus can efficiently replace the CA models obtained by simulation only.

### 4.3.1 Machine-Learning decision block evaluation

The experiments in Fig. 4.3 are repeated many times and lead to a dataset for the ML-decision algorithm. With such a dataset, the ML-decision algorithm can be evaluated using conventional ML techniques. Figure 4.4 presents the simplest method where the dataset is randomly split between a training dataset and a validation dataset (80% and 20% respectively). The training and validation datasets are mutually exclusive. Several runs were performed to evaluate on different random training/validation splits. Other

ML evaluation methods, such as the k-fold validation, were used and led to comparable results. The ML-decision block is trained on the training dataset and then used to predict an answer for each sample in the validation dataset. The predicted answers are compared to the expected ones to evaluate the ML-decision algorithm. The responses being Boolean values, the comparison with expected answers is straightforward and allows to compute the decision error.

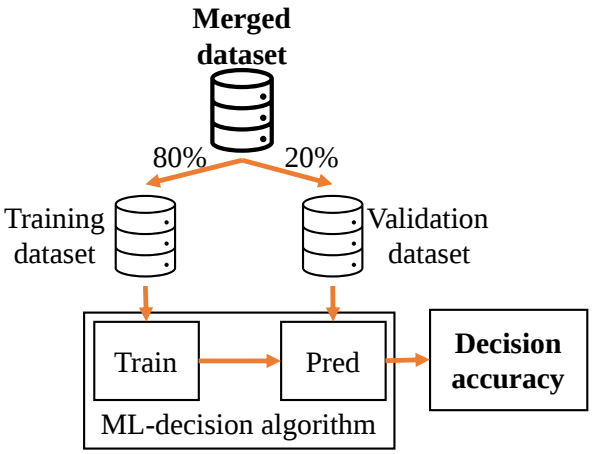



Figure 4.4: Method to evaluate the ML-decision algorithm.

The decision error is the fraction of badly predicted answers which have a potential impact on the quality of CA model over the total number of samples in the validation dataset (the lower, the better). This fraction represents the number of times the ML-decision block takes the wrong decision when orienting a (cell; defect type) combination through one part of the hybrid flow, considering the content of the cells database. Among the two possible mis-decisions, only the false-positive one can have a negative impact on the quality of the CA models. The false-positive error consists in sending a (cell; defect type) combination through the ML-based CA model generation even though the content of the cells database is not sufficient to reliably predict the behavior of the defects in the cell. The other type of mis-decision consists in sending a (cell; defect type) combination through the simulation-based CA model generation even though its defective behavior could have been accurately predicted by the ML-based CA model generation part of the flow (false-negative). The false-negative mis-decision does not negatively impact the quality of the generated CA model and is therefore not accounted for in the decision error.

The experiments presented in this sub-section mimic the usage of the proposed hybrid flow, i.e., when the cells of a new library/technology need to be oriented toward one part of the hybrid flow by the decision block, the standard cells database only contains cells from older technologies. We used library cells from five technologies (from older to recent): C40, C28, P28, P18 and FinFET16. For example, the chronologically realistic application makes that only cells from C40 and C28 are available in the standard cells database when

Table 4.2: Decision error (%) of the ML-decision blocks, for several experiments and several quality threshold (85%/93%/97%)

| **Experiment** | **ML-decision** | **Structural analysis** |
|---|---|---|
| $C40 \rightarrow C28$ | 5/3/2 | 11/18/22 |
| $C40 + C28 \rightarrow P28$ | 3/6/5 | 8/17/29 |
| $C40 + C28 + P28 \rightarrow P18$ | 4/6/4 | 4/11/22 |
| $C40 + C28 + P28 + P18 \rightarrow FF16$ | 3/3/3 | 12/22/26 |

considering P28 cells for the first time. The ML-decision dataset reflects this by choosing the cells in each experiment's cell database from older technologies compared to the cells chosen for CA model generation.

Table 4.2 presents the decision error of the ML-decision algorithm using a growing dataset detailed in the first column. The decision error is computed when orienting the cells of the new technology (right side of the arrow), considering cells databases filled with cells from the technologies mentioned on the left side of the arrow. For each row in Table 4.2, the process of Fig. 4.3 has been repeated at least five times to form the dataset.

The expected 'yes/no' answer for (cell; defect type) combination sample depends on the quality threshold introduced in the previous section. Accepting a lower prediction accuracy of the ML-based CA model generation algorithm means that the predicted CA models can widely differ from the reference simulated CA model. We intuitively consider that the lower the threshold, the more (cell; defect type) combinations will use the ML-based CA model generation part of the flow. The decision block should be efficient at orienting the cells through the hybrid flow irrespective of the chosen quality criterion. Table 4.2 contains decision errors for several quality thresholds (respectively 85%/93%/97%) for each experiment.

The quality of the new ML-based decision block is compared to the old one based on structural analysis. To this end, the decision error of the old decision block is computed. The meaning of the decision error remains the same, i.e., how often does the decision block orient a (cell; defect type) combination through the ML-based CA model generation even though the content of the cells database is not yet sufficient to reliably predict the behavior of the defects in the cell. Note that the decision of the old decision block is based only on the content of the cells database and its decision for a given (cell; defect type) combination does not change with the quality threshold. However, the expected decision for the sample changes with the threshold, making ultimately the decision error of the old decision block to vary with the quality threshold.

The last column of Table 4.2 reports the decision errors of the old decision block based on structural comparison. Table 4.2 clearly shows the supremacy of our new decision

block over the structural one. The new ML-decision block will more often orient a (cell; defect type) combination through the appropriate CA model generation part of the flow, resulting in a better usage of the quick ML-based CA model generation part of the flow, and allowing to quickly get CA models for compatible cells and guarantying CA models of high quality for cells that would be badly predicted considering the current content of the cells database.

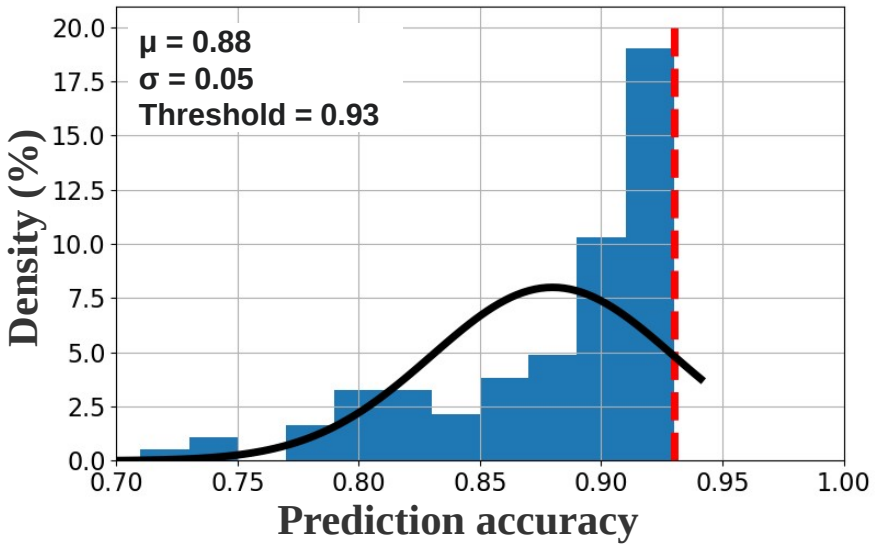



Figure 4.5: Histogram of the prediction accuracy of the badly oriented samples in the $C40 + C28 \rightarrow P28$ experiment

While studying the low number of (cell; defect type) combinations which have been badly oriented by the ML-decision block, we noticed that these combinations mostly have a prediction accuracy close to the threshold, making the 'yes/no' decision difficult and explaining the misprediction of the developed ML-decision algorithm. Figure 4.5 presents the histogram distribution of the prediction accuracies of samples that have been mis-predicated in the $C40 + C28 \rightarrow P28$ experiment. Providing more information about the cells and about the content of the cells database as inputs to the ML-decision algorithm may help to differentiate these cases, thus further decreasing the decision error and increase the global quality of the ML-based decision block. The mis-decision of sending a (cell; defect type) combination through the ML-based CA model generation part of the flow is susceptible to lead to a bad prediction of the behavior of these defects in this cell. This is an improvement compared to the previous work presented in Section 4.1 as a mis-prediction by the structural decision block would have sent the entire cell through the inappropriate CA model generation part of the flow. The new fine-graining and the marginal number of mis-predictions lead the behavior of other defect types in the same cell to be accurately predicted or simulated. The CA model of the entire cell will only contain a few percentages of error. For example, on a population of nearly 2000

cells, sending all cells through the ML-based CA-model generation part of the flow would generate CA models that are on average 91.3% identical to the reference (i.e., simulation only CA models). This average is 92.4% when the old structural hybrid flow ensure quality by orienting some cells through the simulation-based CA model part of the flow. Finally, when fine graining and the new ML-based decision block are used, the average resemblance of the hybrid generated CA models is 99.1%. These few percentages of error can be tolerated by the industry. The next sub-section will demonstrate this point.

### 4.3.2 Industrial validation: ATPG with ML-based CA models

The ultimate goal of this work is to generate CA models for standard cells using the proposed hybrid flow and use them in an exigent industrial environment. This sub-section demonstrates the appropriateness of our method to obtain CA models of quality that can replace the ones obtained by SPICE simulation only.

#### 4.3.2.1 Methodology

The proposed method is sketched in Fig. 4.6 and uses our CA models with an ATPG to generate test patterns targeting cell-internal defects for several industrial designs. The fault coverage of these patterns is computed and compared to the fault coverage of patterns obtained while using commercial CA models (i.e., those obtained using SPICE simulation). More specifically, the hybrid flow is used to generate CA models for cells involved in the designs. These CA models describe the behavior of cell-internal defects, as predicted or simulated by one or the other part of the hybrid flow. Our CA models are written to files in a format compatible with a commercially available ATPG tool. The ATPG uses these files to create test patterns targeting the cell-internal defects. The generated test patterns should allow the detection of a high fraction of the cell-internal defects described in the CA models generated by the hybrid flow (ideally 100%). More importantly, the test patterns should allow the detection of a high fraction of the real cell-internal defects (ideally 100%). The real cell-internal defects are represented by the fault models contained in the CA models generated by SPICE simulation only.

To ensure the latter point, a fault simulation is performed on the designs using a commercial tool. The faults from the simulation-only CA models are injected into the designs and the fault coverage of the test patterns obtained with CA models from the hybrid-flow is computed. Results are presented in the next sub-section. To act as a comparison, another set of test patterns, targeting the real defects contained in the simulation-only CA models is generated by the ATPG. Its fault coverage on real defects is also computed. The important points of comparison between the two sets of patterns are the number of patterns and above all the fault coverage of real cell-internal defects. Close

values between the two sets are the proof that the CA models generated by the proposed updated hybrid flow can efficiently replace the ones obtained by simulation only.

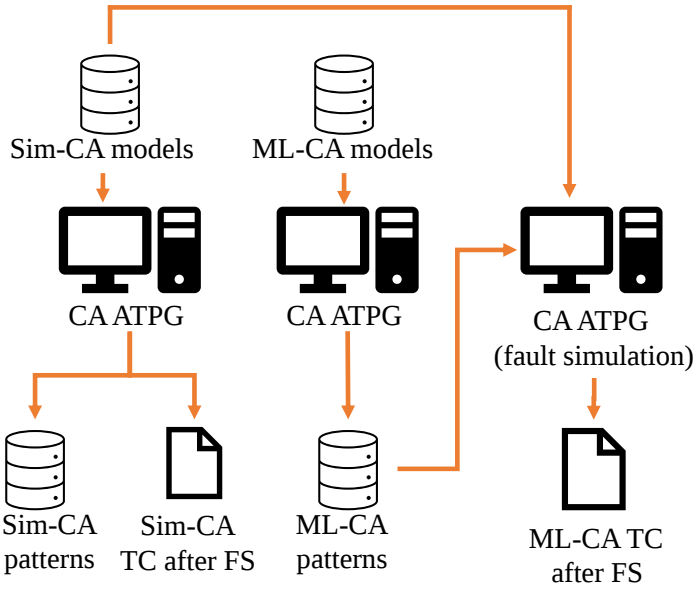



Figure 4.6: ATPG flow to assess quality of the CA models obtained by the hybrid flow

#### 4.3.2.2 Experiments

The proposed methodology has been applied on 15 designs from STMicroelectronics. These designs are blocks of a testchip designed using the P28 technology. The blocks have various complexities and are meant to be independently tested by different sets of ATPG patterns. The CA models for the cells used in the designs have been obtained by the updated hybrid flow. To mimic a realistic industrial usage, the CA models for these P28 cells have been obtained with a cells database filled with C40 and C28 cells. The quality criterion has been set to 93%. In our working group of 1932 cells, 79% of (cell; defect types) combinations have had their behavior predicted by the ML-based part of the flow.

Table 4.3 presents the number of patterns in each set as well as their corresponding fault coverage when targeting the real static cell-internal defects, while Table 4.4 reports the same data when targeting the real dynamic cell-internal defects. The fault coverage is computed by the ATPG as the fraction of detected faults over the entire fault population. The average difference of fault coverage between the two sets of patterns is 0.002 percentage point. The slight difference in the number of patterns (6% on average) results from the slightly different number of faults marked as detected in the two types of CA models.

Tables 4.3 and 4.4 clearly shows that the CA models obtained by the hybrid flow can be used to target real cell-internal defects with the same coverage that CA models

Table 4.3: Number of patterns and associated faults coverages when targeting real static cell-internal defects in several designs

| # | #cells | With hybrid-flow | | With simulation only | |
|---|---|---|---|---|---|
| | | #patterns | %coverage | #patterns | %coverage |
| 0 | 44k | 2922 | 97.21% | 2837 | 97.53% |
| 1 | 44k | 2859 | 98.19% | 2801 | 97.57% |
| 2 | 44k | 2887 | 97.25% | 2490 | 97.43% |
| 3 | 44k | 2955 | 97.87% | 2717 | 97.59% |
| 4 | 44k | 2706 | 97.33% | 2596 | 97.47% |
| 5 | 4.1k | 17 | 100% | 17 | 100% |
| 6 | 4.1k | 17 | 100% | 17 | 100% |
| 7 | 4.1k | 17 | 100% | 16 | 100% |
| 8 | 4.1k | 16 | 100% | 15 | 100% |
| 9 | 4.1k | 25 | 100% | 26 | 100% |
| 10 | 11k | 1738 | 91.48% | 1451 | 91.70% |
| 11 | 13k | 1747 | 90.79% | 1451 | 91.70% |
| 12 | 9k | 1017 | 84.53% | 793 | 84.75% |
| 13 | 11k | 1641 | 83.83% | 1263 | 83.98% |
| 14 | 10k | 1794 | 62.18% | 1686 | 62.26% |

Table 4.4: Number of patterns and associated faults coverages when targeting real dynamic cell-internal defects in several designs

| # | #cells | With hybrid-flow | | With simulation only | |
|---|---|---|---|---|---|
| | | #patterns | %coverage | #patterns | %coverage |
| 0 | 44k | 2802 | 94.38% | 2746 | 94.01% |
| 1 | 44k | 2957 | 96.17% | 2849 | 95.99% |
| 2 | 44k | 2713 | 93.60% | 2636 | 93.10% |
| 3 | 44k | 2964 | 96.19% | 2847 | 96.04% |
| 4 | 44k | 2707 | 93.43% | 2596 | 92.97% |
| 5 | 4.1k | 37 | 100% | 35 | 100% |
| 6 | 4.1k | 35 | 100% | 35 | 100% |
| 7 | 4.1k | 36 | 100% | 36 | 100% |
| 8 | 4.1k | 36 | 100% | 37 | 100% |
| 9 | 4.1k | 40 | 100% | 39 | 100% |
| 10 | 11k | 2125 | 94.32% | 2173 | 95.27% |
| 11 | 13k | 2179 | 94.39% | 2239 | 95.10% |
| 12 | 9k | 1311 | 93.79% | 1404 | 93.54% |
| 13 | 11k | 1958 | 94.36% | 1984 | 93.91% |
| 14 | 10k | 1547 | 71.52% | 1650 | 71.06% |

obtained by simulation only. These results demonstrate the pertinence and effectiveness of our approach for CA models generation.

# Chapter conclusion

The quality of prediction of the ML-based CA model generation flow presented in Chapter 3 depends on the content of the ML training database at moment of use. For this reason, the ML-based CA model generation flow should not be used to predict the CA model of every cell in a library. Instead, when the CA model for a new cell is needed, this new cell should be analyzed and compared to the content of the ML training database. If the new cell resembles some of the cells in the training database, then the Chapter 3's fast ML-based CA model generation flow should and will be used to decrease run-time. If the new cell has new and unknown properties, then the conventional simulation-based CA model generation flow should be used, to ensure obtainment of a CA model of high quality for this cell.

In this chapter, we have presented a hybrid flow used to obtain CA models for all cells in a cell library. This mixed solution consists in combining ML-based CA model generation and conventional (simulation-based) CA model generation. A **decision block** analyses the new cell, compares it with the content of the ML training database and orients it through one of the CA model generation parts of the flow.

Our latest decision block is based on a ML algorithm, which training dataset and input features have been carefully elaborated to best represent the problem of analyzing and comparing standard cells. This ML-decision algorithm offers high level of accuracy, meaning that a high fraction the new cells are oriented through the correct flow, allowing a reduction of run-time as Chapter 3's ML-based CA model generation flow is used whenever applicable.

Experimental results on 5 technologies and thousands of standard cells have shown that (i) the average quality of CA models generated by using the hybrid flow is 99.1%, to be compared to an average quality of 91.3% for CA models obtained by prediction only is. (ii) Providing those CA models to use an ATPG on several industrial designs leads to sets of test patters achieving the same levels of fault coverage than CA models obtained by conventional simulation-based method. These results demonstrate the pertinence and effectiveness of our approach for CA models generation.

Chapter 5

# Machine-learning for cell-aware diagnosis

The first step when a customer return occurs in industry is to reproduce the failure mechanism with any original test and test conditions (voltage, temperature, etc.) used during initial post-fabrication test. Depending on the test scenario (test sequence, test scheme, test conditions), different types of defects and failure mechanisms are targeted, thus leading to different diagnosis outcomes. Based on our previous work, this chapter presents a comprehensible framework for cell-aware diagnosis of customer returns. The first part of the chapter summarizes our previous work in which a generic cell-aware diagnosis flow using a Bayesian classification method was proposed to precisely identify defect candidates in combinational cells of a defective circuit according to a given test scenario. The second part of the chapter extends the previous work by dealing with defects in sequential cells of defective circuits. New experiments done on a silicon test chip and a customer return for a given test scenario have proven the effectiveness of our flow in terms of diagnosis accuracy and resolution.

The rest of this chapter is organized as follows. Section 5.1 introduced CA diagnosis. Section 5.2 discusses the test scenarios envisaged during customer return diagnosis. Section 5.3 presents the generic CA diagnosis flow used in [16], [20]–[24] and summarizes the various contributions. Section 5.4 shows how defects in sequential cells can be dealt with the CA diagnosis flow. Section 5.5 presents some results obtained on benchmark circuits as well as on a silicon test chip. Section 5.6 concludes the chapter and draws some perspectives.

This chapter first summarizes work done by the research team prior to my arrival and published in [16], [20]–[24]. The second part of this chapter presents extensions to the method performed during my Ph.D. work. In the context of this chapter, all work is designated as "our work".

## 5.1 Cell-Aware Diagnosis

One technology that has received a lot of attention recently in the field of digital IC design and quality assessment is Cell-aware (CA) test and failure diagnosis [12]. With more defects inside cells at leading-edge technology nodes, CA test, which deterministically targets defect locations inside standard cells, and CA diagnosis, which can identify the location and type of cell-internal defects at the transistor level, are quality assessment solutions widely adopted today in industry [14].

Cell-aware diagnosis is valuable in the context of large, complex cells such as adders, multipliers, and multi-bit sequential elements. Even when a defect is known to be within a cell, finding defects in such a complex cell during Physical Failure Analysis (PFA) can be challenging and time-consuming, especially if the defect is exhibited for a specific test condition and undetected for some others. CA diagnosis shortens the lengthy investigation process by pinpointing a small subsection of the suspected cell. It improves results for diagnosis scenarios such as customer return analysis as well as volume diagnosis applications like yield analysis [13].

Customer returns are defective ICs that passed all functional, structural, and parametric tests after manufacturing but failed in the field. Industrial experiences show that they are mostly due to latent defects [61]. This is especially true for critical products (e.g., automotive) where a comprehensive test flow has been applied to ensure zero test escape after manufacturing. In this context, the first step when a customer return occurs is to reproduce the failure mechanism with any original test and test conditions. Next, 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 (system, core, and cell levels) [3], [4], [62]–[65].

The quality of a diagnosis outcome is usually evaluated owing to two metrics: accuracy and resolution. A diagnosis is accurate if the actual defect is included in the reported list of candidates. Resolution refers to the total number of candidates reported for each actual defect. An accurate diagnosis with perfect resolution (i.e., one) is the ideal case. Achieving such a perfect resolution with conventional diagnosis approaches based on cause-effect and/or effect-cause analysis becomes increasingly difficult [66]–[68]. The study of a commercial chip in [69] demonstrates that just over 30% of the diagnosed defects exhibit ideal resolution.

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, especially in the volume diagnosis scenario [70]–[81]. Most of these techniques focuses on supervised

root-cause analysis because it naturally aligns with the common practice of training with labeled historical data and usually performs well in industrial diagnosis tasks [82]. However, though efficient, these techniques address volume diagnosis for yield ramp-up, which is a different problem than fault diagnosis of customer returns. Indeed, during volume diagnosis, a lot of 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, bad). It is therefore possible to use these data for failure diagnosis of a new failed chip. Conversely, during fault diagnosis of a customer return, 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). For this reason, learning-guided approaches existing for volume diagnosis cannot be reused for diagnosis of customer returns.

In [16], [20]–[24], we proposed several learning-guided solutions for CA diagnosis of mission mode failures in customer returns. All solutions are based on a Bayesian classification method for accurately identifying defect candidates in combinational standard cells of a defective IC. The selection of one solution over another depends on the considered test scenario (test sequence, test scheme, test conditions), thus leading to different types of targeted defects and failure mechanisms. Experiments on benchmark circuits and industrial customer returns as well as comparison with a commercial CA diagnosis tool have shown the effectiveness of all solutions.

## 5.2 Previous works

When conducting the diagnosis of a customer return in industry, different test scenarios may be considered, most of the time (or in a first attempt) reproducing the test scenarios used initially during manufacturing test. Figure 5.1 gives an example of a multi-run ATPG flow used in industry for screening defects in logic parts of a System-on-Chip (SoC). As can be seen, several ATPG runs are executed at various speeds (low or at-speed) in order to get different test sequences targeting different defect types (static or dynamic cell-internal defects) and fault models (SA – stuck-at, or TF – transition). Static defects are defects that require one-vector test patterns to be detected. Dynamic defects are defects that require two-vector test patterns to be detected. With the advent of nanometric 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 (operational, environmental, etc.).

From the ATPG flow illustrated in Fig. 5.1, three test sequences are obtained and can be applied sequentially to a CUT to achieve a targeted test coverage. During diagnosis

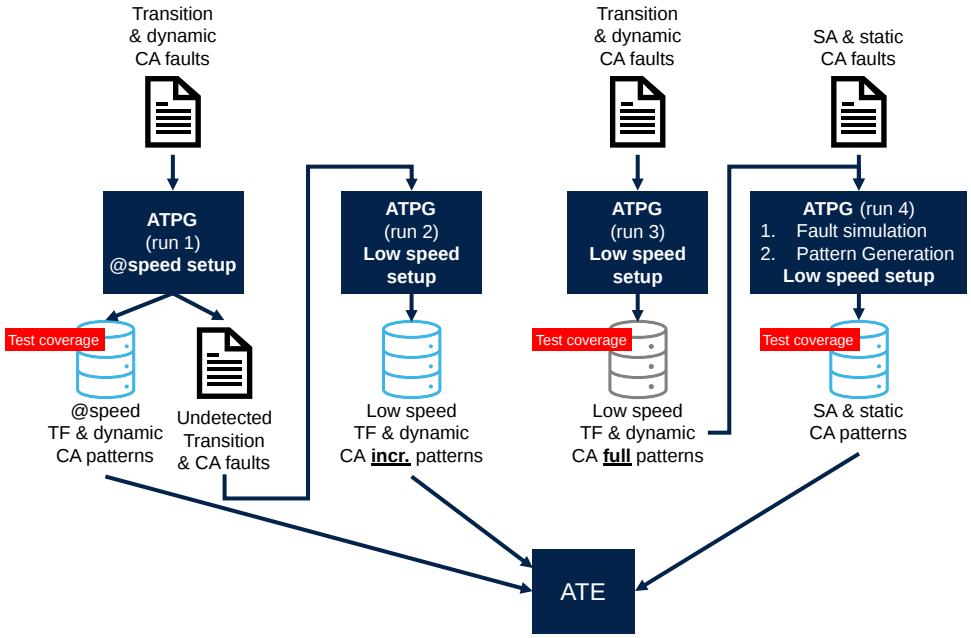



Figure 5.1: Example of a multi-run industrial ATPG flow

of a customer return, the same test sequences can be reused to exhibit the failure observed during mission mode.

### 5.2.1 Summary of our Previous Work

In [16], [20]–[24], we considered various test scenarios for CA diagnosis and described the corresponding diagnosis algorithm implementations. The test scenarios are sketched in Table 5.1. In [20], [21], two distinct processes were developed to diagnose static and dynamic defects separately. In [20], we assumed a basic scan testing scheme for static CA test sequences application, thus targeting stuck-at faults plus static intra-cell defects during diagnosis. In [21], we assumed a fast sequential testing scheme for dynamic CA test sequences application, thus targeting transition faults plus dynamic intra-cell defects during diagnosis. Note that [22] is just a combination of [20] and [21] from a test scenario point of view. The main limitation of the solutions in [20]–[22] is the required a priori knowledge of the targeted defects type in the Circuit-Under-Diagnosis (CUD). In other words, a test engineer needs to know what type of defect is screening before choosing between [20] or [21].

Table 5.1: Test scenarios considered in [16], [20]–[24]

| **Test Sequence** / **Fault & Defect** | Static (low speed) | Dynamic (at-speed) | Static + Dynamic |
|---|---|---|---|
| Stuck-at & Static CA | [20][22] | [24] | [23][16] |
| Transition & Dynamic CA | | [21][22][24] | [23][16] |

In an attempt to deal concurrently with all types of defect that may occur in cus-

tomer returns, without any a priori knowledge of the targeted defect type, we proposed a new implementation of the CA diagnosis flow in [16], [23]. Note that [16] is a fully extended version of [23]. We assumed a test scenario in which two test sequences (static and dynamic) are used successively, each one assuming 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.

Constructing such a comprehensive flow imposed setting up a new framework with specific rules 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. The flow was experimented and validated owing to an industrial test chip and a STMicroelectronics customer return.

In [24], we proposed a new version of the CA diagnosis flow 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. According to the flow depicted in Fig. 5.1, 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) after the first run of ATPG in Fig. 5.1. 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 taken into account 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 losing information and hence without decreasing the quality of the training and inference phases. As in [16], [23], experimental results gathered on a silicon test chip have shown the benefit of using such a new version of the CA diagnosis flow.

### 5.2.2 Improvement of Diagnosis Resolution

Each test scenario as exemplified in Table 5.1 may lead to the occurrence of one or more failing patterns producing failure files to be diagnosed. The knowledge of failing and passing test scenarios can be used to improve the diagnosis resolution. To this end, it

is crucial to have perfect knowledge about which defect type is detected by each test sequence and related test patterns. Let us clarify this point as follows:

- Static defects require only one test pattern to be detected and they are usually targeted by static test patterns. However, these defects can also be detected by low speed and at-speed dynamic test patterns. In fact, dynamic test patterns are two-vector test patterns, and it may happen that the second one exercises a stimulus detecting static defects.

- Low speed dynamic defects require two-vector test patterns to be detected. The test frequency used to catch these defects is not relevant as the delay they induce is so huge that even a very low speed test can catch them. Therefore, low speed dynamic defects can be easily detected either by a low speed test or by a dynamic at-speed test. It is also worth mentioning that for scan designs, low speed defects can be accidentally detected by static test patterns. Indeed, it happens that during the last scan shift cycle plus the capture cycle, a two-vector test pattern that exercises a dynamic defect is created.

- At-speed dynamic defects require two-vector test patterns applied at high frequency to be detected. They induce a delay that can be measured during the cell-aware model creation. Only at-speed tests can detect these defects. The knowledge of the design max frequency can help to identify if the delay caused by the defect is large enough to be detected or not.

The ATPG flow as illustrated in Fig. 5.1 is an incremental flow, which means that only undetected defects are targeted by each new test pattern. However, it may happen that a defect targeted by, e.g., run 1, is also detected by run 2 or 3 unintentionally. Therefore, in a preprocessing step, we fault simulate the entire set of CA defects by each test pattern and identify detectable and undetectable defects. This information is further used to improve diagnosis resolution.

Table 5.2: Failing and passing test scenarios and corresponding analysis for diagnosis improvement

| Test Scenario | @speed test | Low speed test | Static test | Analysis |
|---|---|---|---|---|
| Static & Dynamic patterns with static or dynamic defects | F | F | F | Remove detectable @speed dynamic defects |
| Dynamic patterns with dynamic defects | F | F | P | Remove detectable @speed and static defects |
| Static & Dynamic patterns with static defects | F | P | F | Remove detectable dynamic defects |
| Dynamic patterns with dynamic defects | F | P | P | Remove detectable low speed and static defects |
| Static & Dynamic patterns with static or dynamic defects | P | F | F | Remove detectable @speed dynamic defects |
| Dynamic patterns with dynamic defects | P | F | P | Remove detectable @speed and static defects |
| Static patterns for static defects | P | P | F | Remove detectable dynamic defects |

According to the above discussion, seven failing and/or passing test scenarios may occur as depicted in Table 5.2. Each scenario is specified in Column 1. For example, in the first scenario, static and dynamic test patterns are generated to target static and

dynamic defects in the customer return. For each test scenario, results obtained when applying the corresponding test patterns using various test scheme (at-speed, low speed and static) are given in Column 2, 3 and 4. Note that, for at-speed and low speed test, the same diagnosis algorithm can be used. However, the analysis of the results will be different. Column 5 explains how we proceed to improve diagnostic resolution and hence quality. For example, in the second test scenario where only one or more at-speed patterns are failing, we know that this type of pattern can detect all CA defect types. However, as low speed and static test patterns were passing, we can remove detectable low speed and static defects from the list of candidates.

## 5.3 Learning-based cell-aware diagnosis flow

Figure 5.2 is a generic view of the learning-based CA diagnosis flow presented in [16], [20]–[24]. 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.

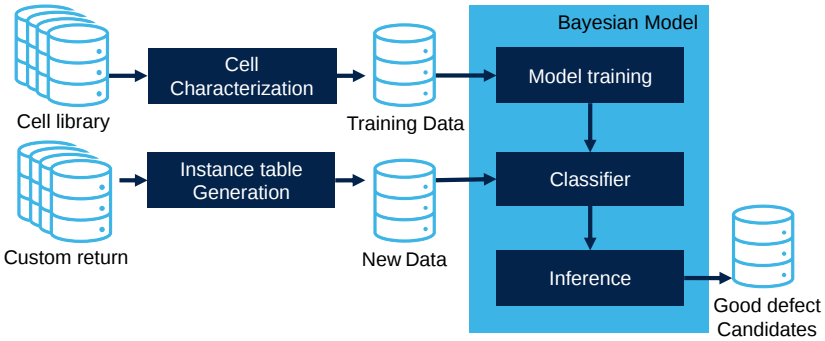



Figure 5.2: Generic view of the cell-aware diagnosis flow used in [16], [20]–[24]

After investigating several machine learning algorithms and observing their inference accuracies in [20], a Bayesian classification method has been chosen for the learning and inference phases. So, the first main step of our 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. Following the k-fold cross-validation methodology, 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 (validation dataset) [83]. More details about performance evaluation as done in our 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. In the next subsections, we detail the various steps of the CA diagnosis flow able to deal with any type of cell-internal defect (i.e., static and dynamic) that may occur in customer returns.

### 5.3.1 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 [16], [20]–[23] and containing six instances for an arbitrary two-input cell, is shown in Table 5.3. Each instance is associated to a static defect (D1, D2, D3) or a dynamic defect (D11, D12, D13). A 1 (0) indicates that defect Di is detectable (not detectable) at the output of the cell when the cell-level test pattern Pj is applied at the inputs of the cell. Cell-level test patterns (called cell-patterns in the sequel) are static (one input vector - P1 to P4 in Table 5.3) or dynamic (two input vectors - P5 to P16 in Table 5.3 in which R (F) indicates a rising (falling) transition at the cell input). For an n-input cell, there exists $2^n$ static cell-patterns and $2^n.(2^n1)$ dynamic cell-patterns.

Table 5.3: Example of training dataset for all defect types (static and dynamic) in a two-input cell as used in [16], [20]–[23]

| 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 |

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 (D11, D12, D13) 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.

As only dynamic test sequences are considered in [24], the representation of training data as used in [16], [20]–[23] 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 (V1V2) in which the values of V1 and V2 have to be properly defined for the defect to be detected. Conversely, only the value of V2 is significant for a static defect to be detected by such pattern, irrespective of the value taken by V1. When looking at Table 5.3, one can see that P1=00 is embedded in P6=0F, P11=F0 and P12=FF, and the same for P2, P3 and P4. Similarly, we can see that static defect D2 is detectable by P1 and P4, and hence by P6, P8, P10, P11, P12, and P15. So, by "compacting" a training dataset as shown in Table 5.4, 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 Table 5.3) 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 defect.

Table 5.4: Example of training dataset for all defect types (static and dynamic) in a two-input cell as used in [24]

| 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 |

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 D1 and D11 where D1 is static and detectable by 00 and D11 is dynamic and detectable by F0 (note that 00 is the second vector of F0). As can be seen in Table 5.4, 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.

### 5.3.2 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. 5.3. 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, we 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 [16], [23].

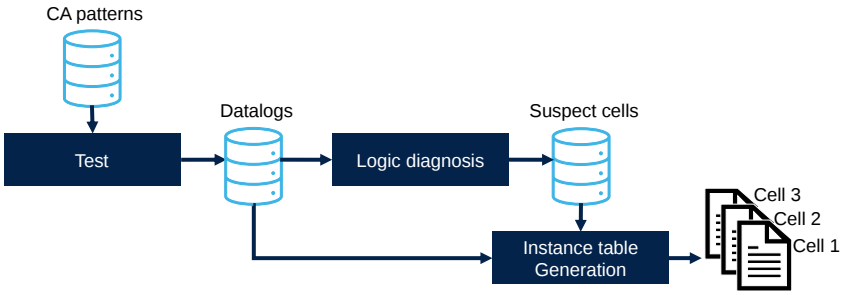



Figure 5.3: Generation flow of instance tables

The format of a static instance table is illustrated in Fig. 5.4 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. 5.4 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.

### 5.3.3 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, we can extract one or more defect candidates

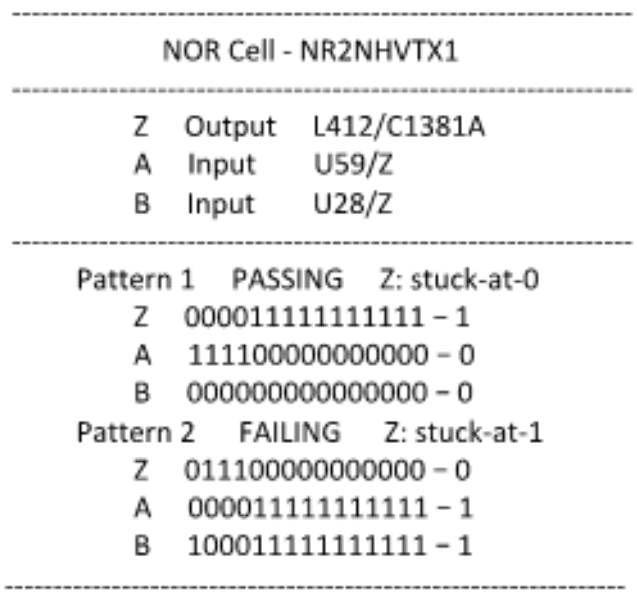



Figure 5.4: Example of static and dynamic instance tables

that have to 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.

Table 5.5: Format of a new data instance for a two-input cell

| 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 | $D_i$ |

Table 5.6: Format of a new data instance as used in [24]

| 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 | $D_i$ |

The formats of a new data instance as used in [16], [20]–[23] and [24] are illustrated in Table 5.5 and Table 5.6 respectively. These formats are 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 Pi for a given defect candidate, meaning that the candidate is indeed detectable (resp. undetectable) by the cell-pattern Pi 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.

## 5.4 Diagnosis of defects in sequential cells

All the work carried out in [16], [20]–[24] 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. In this section, we show how the previous diagnosis flow can handle sequential cells and related defects by adding new information to the training dataset.

The two main differences between combinational cells and sequential cells are that i) the latter have 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 extended 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 Table 5.7. 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 Table 5.7. 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, we include static cell-patterns (e.g., P1 to P4 in Table 5.7) in the training data of sequential cells.

Table 5.7: Example of training dataset for all defect types (static and dynamic) in a sequential cell. The pin order is clock-data-previous output.

| 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 |

With the above representation of training data for sequential cells, one can see that the diagnosis flow in Fig. 5.2 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 manipulated standard

cell type.

## 5.5 Experimental results

This new CA diagnosis flow targeting both combinational and sequential cells has been implemented in a Python program. For validation purpose, we have experimented the proposed flow in three different ways:

- First, we conducted experiments on ITC'99 benchmark circuits with defect injection campaigns targeting combinational cells in each circuit. Various results are reported in [16], [20]–[24] to show the superiority of our framework when compared to commercial diagnosis solutions. A summary of these results is given in Subsection 5.5.1.

- Next, we considered a test chip developed by STMicroelectronics and designed using a 28 nm FDSOI technology, and we conducted two defect injection campaigns targeting sequential cells. Results are reported in subsection 5.5.2 and also demonstrate the effectiveness of our diagnosis framework.

- Finally, we considered a customer return from STMicroelectronics, and we performed a silicon case study with a real defect subsequently analyzed and identified during PFA. Results are reported in subsection 5.5.3.

### 5.5.1 Summary of Results Reported in [16]

In [16], we conducted experiments on ITC'99 benchmark circuits synthesized in a full scan manner using a 28 nm FDSOI technology from STMicroelectronics. A commercial CA ATPG tool was used to generate static and dynamic CA test sequences targeting maximum fault coverage for each circuit. The behavior of the tester was simulated by performing a defect injection campaign in each circuit (about 2000 injections per circuit). Defects were injected into a number of randomly selected combinational cells by considering each cell transistor and targeting all possible static and dynamic defects affecting that transistor. As several defects have the same impact on the logic behavior of the cell, and hence are logical-equivalent defects, they were grouped in defect classes. After test sequences application, test information were collected to build the tester data log. Afterwards, we used a commercial logic diagnosis tool to determine the lists of suspected cells after static and dynamic test sequences application. Main circuit characteristics (number of cells and scan flip-flops) and test parameters (number of static and dynamic test patterns, stuck-at and static CA fault coverage, transition, and dynamic CA fault coverage) can be found in [16].

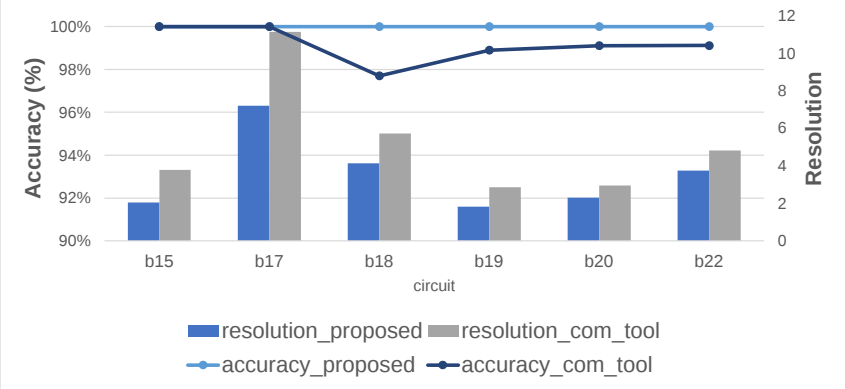



Figure 5.5: Overall Cell-Aware Diagnosis Results [16]

For generating training data, we used characterization data provided by a commercial tool and ST libraries. For generating new data instances, we performed post-processing of instance tables obtained as shown in Fig. 5.3. From the training data and the Gaussian NB model, we make predictions on new data instances. Results obtained are a list of defect candidates with the highest probability to be the root cause of failure.

Figure 5.5 summarizes the results obtained on the biggest ITC'99 benchmark circuits. Histograms report the accuracy and give, for each circuit, the percentage of cases in which the injected defect was reported in the list of suspects provided by the proposed CA diagnosis and the commercial CA diagnosis tool, respectively. The commercial tool is non-probabilistic and provides the list of all suspects obtained after CA diagnosis with a ranking and a matching score. The same characterization data and test scenario were used in both cases. These results show that the real (injected) defect is always identified by the proposed diagnosis flow. Sometimes, it is the only candidate and has a probability of 1 to be the best candidate. Sometimes, it is reported with some other candidates identified in one or more suspected cells. Conversely, we can observe that the commercial tool is not always able to report the injected defect as candidate. This proves the superiority of our proposed framework in terms of accuracy (always 100%), which is not the case of the commercial tool that sometimes provides inaccurate results (at least for 5 out of 6 circuits). The reason of these misdiagnosis cases with the commercial tool are explained in [16].

The curves in Fig. 5.5 report the resolution and give, for each circuit and considering all injection campaigns, the average number of suspects reported by the proposed method and the commercial tool, respectively. As can be seen, the resolution achieved with our method is always better. So, overall, these results confirm the superiority of our approach.

In our experiments, suspected defects were classified using a publicly available machine learning software package called scikit-learn [59], which is an integrated development

Table 5.8: Main features of the silicon test chip

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

environment with a suite of ML tools [84]. The single defect assumption was considered, although the proposed framework is able to manage situations where multiple defects have occurred, provided that those defects are not in the same cell. This significant feature comes from the fact that our diagnosis flow considers all suspected cells one at a time, and then incrementally constructs a list of suspected defects identified in each of these cells. Finally, in-field failure mechanisms related to premature aging, such as NBTI or HCI, essentially lead to resistive opens and shorts. These mechanisms, which need to be considered in the context of customer returns, can be appropriately taken into account with our CA diagnosis flow.

The CPU time taken by the proposed flow to provide a list of defect candidates is always very low (few seconds) and does not depend on the circuit size. Only the number of suspected cells obtained after logic diagnosis may have an impact on the CPU time (for the generation of instances tables) but in a very light way (as this number is always very low). 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.

### 5.5.2 Simulated Test Case Studies

We also 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.

We performed a first simulated case study with a static defect injection campaign. We successively injected all possible static defects into three scan flip-flops (SFF) of a single full-scan digital block. We tested this block 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 our CA diagnosis flow and averaged over all defect injections have shown an accuracy of 100% (the injected defect was always reported in the list of suspects) with a resolution of 1.25. The resolution ranges between 1 and 3, and Fig. 5.6 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 5.9: 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 |

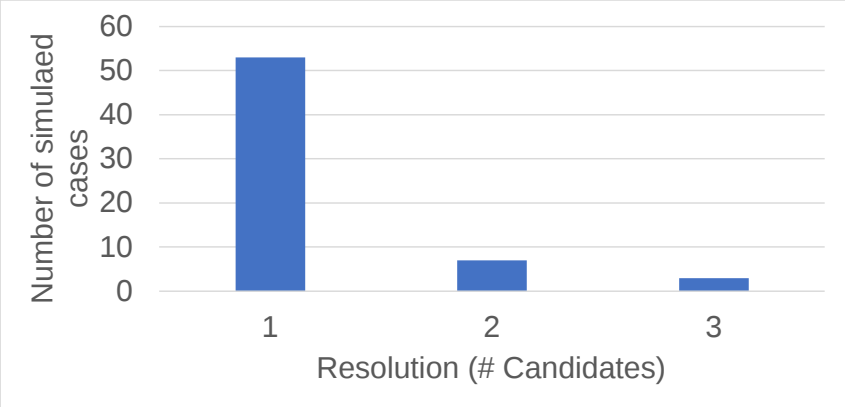



Figure 5.6: Distribution of the resolution w.r.t. the simulated cases

We then performed a second simulated case study with another defect injection campaign on the same test chip. We successively injected all possible dynamic defects into three scan flip-flops of a single full-scan digital block. This time, we applied a dynamic CA test sequence achieving 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 our learning-based 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.

### 5.5.3 Silicon Test Case Study

Finally, we performed a silicon case study on a customer return designed with a 28 nm FDSOI technology from STMicroelectronics. The test conditions used to run the experiments were the following: a nominal supply voltage of 0.83 V, a scan shifting 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. We experimented our CA diagnosis flow, and we obtained the following results:

- 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).

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 our 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 (cf. Fig. 5.7) and observing the failure analysis cross-sectional view.

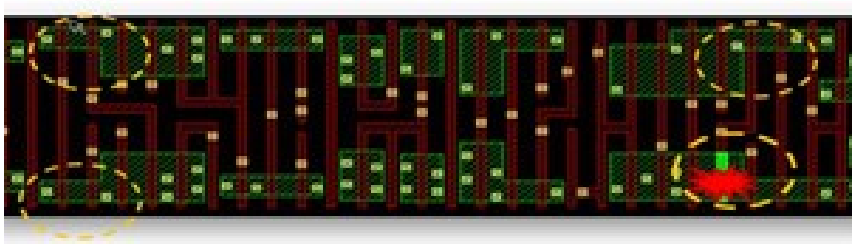


Figure 5.7: Layout view of the suspected cell and the incriminated transistor. Yellow circles indicate defect candidates and red mark indicates actual observed defect.

## 5.6 Conclusion, discussion, and future work

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

Results of these experiments prove the appropriateness of a learning-based method to solve our problem, despite the small size of the training dataset used (only one sample for one defect class). When multiple defect sizes and test conditions will be used, this will be even truer. Indeed, multiple samples (one for each defect size or defect size range, one for each PVT test condition) will be associated to a given defect class, simply because the behavior of the defect will differ when applying the same set of test patterns. In

terms of timing and complexity, this will just slightly impact our method, since training dataset is extracted from characterized cell libraries that are generated anyway for test and diagnosis purpose. So, even with large cell libraries with a huge number of defects to be simulated (e.g., 631 cells in a library, each with 4 to 6 inputs, 951 shorts and 749 opens on average – typical example of an ST library), our framework will still be easily and time-efficiently applicable.

It is worth noting that among other factors, the effectiveness of our framework can be explained by the fact that Naïve Bayes algorithm usually offers good classification performance [85]. The NB classifier requires a small amount of training data to estimates its parameters [86], which is the case in our method, as only one instance per class (i.e., CA defect) is available. On the other hand, other popular ML classification algorithms, such as K-Nearest Neighbors (KNN) classifier or classifiers based on a Support Vector Machine (SVM), which estimate the class of a new sample by analyzing the classes of similar training samples, cannot properly work when only one sample per class is available.

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

Chapter 6

# Conclusions and future work

The manufacturing process of nanometric integrated circuits is prone to errors. Manufacturing defects or process variations can cause a circuit to deviate from its intended behavior or specifications. Fault models are abstraction models of the defects' and variations' effects on the circuits and are used to guide ATPG tools in the selection of quality test patterns for manufacturing test.

*Cell-Aware* (CA) fault models have been proposed to represent the subtle defects found inside the standard cells used to design digital ICs. The CA models contain a dictionary mapping cell-internal defects to the cell-level patterns that expose them at the standard cell outputs. During *CA test*, ATPG tools activate and propagate CA faults to create test patterns explicitly targeting cells internal defects. *CA diagnosis* tools use the fault dictionary to identify the location and type of intra-cell defects, providing valuable insights in the context of large and complex standard cells. The CA models are conventionally obtained using a costly and time-consuming process employing numerous analog simulations (SPICE).

To accelerate the CA models generation process and help spreading the CA methodologies across the industry, this thesis has presented an innovative use of Machine-Learning (ML) techniques to perform defect characterization of standard cells. The ML algorithms are used to replace the SPICE simulator to determine how a potential defect can be detected inside a standard cell. A ML-compatible representation of standard cells in a matrix format has been presented. In a supervised training manner, ML algorithms were trained on existing CA models data, then used to predict the defective behaviors of new standard cells, regardless of the technology or the transistor features size.

To alleviate the limitations of supervisory trained ML algorithms, which are better at predicting the behavior of objects similar to the ones they were trained on, we have proposed a hybrid-flow. The hybrid-flow mixes the proposed fast ML-based CA models generation method with the conventional simulation-based CA model generation method to save time with new cells resembling the ones used during training, and guarantee quality CA models for the others.

We have presented decision blocks to analyze the new cells, compare them with the cells in the ML training database and orient them though the hybrid flow. The latest decision block is also based on a ML algorithm and offers a high rate of decision accuracy. An innovative process has been proposed to establish a training dataset for the ML decision block.

Once logic diagnosis has located cell candidates, CA diagnosis is run to pinpoint intra-cell candidates, helping increase diagnosis quality and decrease the search area for PFA teams. Extending previous work, this thesis presented the use of ML for CA diagnosis of customer returns. The ML algorithm is trained using CA models to differentiate cell internal defects based on their activation conditions.

All the presented methods and processes have been experimented and validated on real industrial data and designs from STMicroelectronics.

# Future work

The key enabler of this work is the ML-compatible representation of the standard cells. By improving/enriching it, future works could add support for more types of cells, defects, or cell-patterns. An appropriate representation could also benefit to the recently presented device-aware test methods [87]. Improving the matrix representation would improve the quality of the ML-predicted CA data and decrease the cost of generating CA models for compatible standard cells, allowing the usage of CA methodologies on more cells, ultimately improving the quality of digital ICs.

The CA methodologies could be extended to the test of memories, adapting the innovations from the digital IC world, and switching memory testing paradigm from functional to structural test. The definition of "special standard cells" abstracting memories' analog behavior would allow logic tools to test and diagnose memory blocks, helping define structural coverage metrics for memories and generate extra sets of patterns to maximize these metrics. Once defined the "special standard cells" constituting IC's memories, the ML-based CA models generation tools could be adapted to account for the memories' specificities. Moreover, by proposing a ML-compatible representation of an entire memory and its subtle analog specificities, one could then manipulate and characterize a memory using ML.

The ML decision block and the hybrid flow could also be extended to other situations where a ML algorithm is not efficient enough at its task yet, by providing a fallback and allow over-time improvement of the ML algorithm. This allows for the replacement of time-consuming or costly applications by their ML counterpart without sacrificing quality.

# Publications of the author

---

[60] P. d'Hondt, A. Ladhar, P. Girard and A. Virazel, **"A Learning-Based Methodology for Accelerating Cell-Aware Model Generation,"** 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1580-1585, doi: 10.23919/DATE51398.2021.9474227.

[88] P. d'Hondt, A. Ladhar, P. Girard and A. Virazel, **"A Comprehensive Learning-Based Flow for Cell-Aware Model Generation,"**, 2022 IEEE International Test Conference (ITC), 2022

[89] P. d'Hondt, S. Mhamdi, P. Girard, A. Virazel, A. Ladhar, **"A comprehensive framework for cell-aware diagnosis of customer returns,"** Journal of Microelectronics Reliability, Volume 135, 2022, 114595, ISSN 0026-2714, doi j.microrel.2022.114595

Pierre d'Hondt, Aymen Ladhar, Patrick Girard, Arnaud Virazel, **"Cell-Aware Model Generation by Using Machine Learning,"** to be published in Springer's Frontiers of Electronic Design in 2022.

P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, **"IC Test Quality and Yield Improvement through the use of Machine Learning,"** at ST TechWeek 2022 (peer-reviewed ST-Microelectronic internal international conference)

P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, **"Accelerating Cell-Aware Model Generation Through Machine Learning,"** Poster at GdR SoC2 colloquium, 2021

P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, **"Accelerating Cell-Aware Model Generation Through Machine Learning,"** Poster at LIRMM PhD students day, 2022

# Glossary

| | |
|---|---|
| ATPG | **A**utomatic **T**est **P**attern **G**eneration |
| CA | **C**ell **A**ware |
| CUT | **C**ircuit **U**nder **T**est |
| DL | **D**eep **L**earning |
| DPPM | **D**efective **P**art **P**er **M**illion |
| IC | **I**ntegrated **C**ircuit |
| ML | **M**achine **L**earning |
| RFC | **R**andom **F**orest **C**lassifier |

# Bibliography


[1] A. Karel, "Comparative Study of FinFET and FDSOI Nanometric Technologies Based on Manufacturing Defect Testability," Theses, Université Montpellier, Oct. 2017. [Online]. Available: `https://tel.archives-ouvertes.fr/tel-01952734`.

[2] K. Y. Cho, S. Mitra, and E. J. McCluskey, "Gate exhaustive testing," en, in *Proc. IEEE International Test Conference*, 2005.

[3] A. Ladhar, M. Masmoudi, and L. Bouzaida, "Efficient and accurate method for intra-gate defect diagnoses in nanometer technology," en, in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2009.

[4] Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and E. Auvray, "Effect-cause intra-cell diagnosis at transistor level," en, in *Proc. IEEE International Symp. on Quality Electronic Design*, 2013.

[5] F. Hapke, M. Reese, J. Rivers, and A. Over, "Cell-aware production test results from a 32-nm notebook processor"," en, in *Proc. International Test Conference*, 2012.

[6] Z. Gao, M.-C. Hu, J. Swenton, *et al.*, "Optimization of cell-aware atpg results by manipulating library cells' defect detection matrices," en, in *Proc. IEEE International Test Conference in Asia (ITC-Asia)*, 2019.

[7] E. Amyeen, D. Nayak, and S. Venkataraman, "Improving precision using mixed-level fault diagnosis," en, in *Proc. International Test Conference*, 2006.

[8] H. Tang, A. Jain, and S. Pillai, "Using cell aware diagnostic patterns to improve diagnosis resolution for cell internal defects," en, in *Proc. Asian Test Symposium*, 2017, pp. 231–236.

[9] X. Fan, M. Sharma, W.-T. Cheng, and S. Reddy, "Diagnosis of cell internal defects with multi-cycle test patterns"," en, in *Proc. Asian Test Symposium*, 2012.

[10] B. Archer and C. Schuermyer, "Cell-aware test for lower dppm and faster silicon diagnosis," en, in *Proc. Synopsys User Group (SNUG)*, 2017.

[11] N. Feldman, "Accelerating silicon diagnosis using a cell-aware flow," en, in *Proc. Synopsys User Group (SNUG)*, 2017.

[12] F. Hapke and al, "Cell-aware test," en, *IEEE Trans. on Computer-Aided Design*, vol. 33, no. 9, pp. 1396–1409, 2014.

[13] P. Maxwell, F. Hapke, and H. Tang, "Cell-aware diagnosis: Defective inmates exposed in their cells," en, in *IEEE European Test Symp*, 2016.

[14] S. Pateras, "Ic test solutions for the automotive market," en, *Mentor Graphics, White Paper*, May 2017.

[15] F. Hapke, R. Krenz-Baath, A. Glowatz, *et al.*, "Cell-aware fault model creation and pattern generation," en, US Patent 12/718,799 12/718,799, 2010.

[16] S. Mhamdi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "A learning-based cell-aware diagnosis flow for industrial customer returns," en, in *Proc. IEEE International Test Conf*, 2020.

[17] F. Lorenzelli, Z. Gao, J. Swenton, S. Magali, and E. Marinissen, "Speeding up cell-aware library characterization by preceding simulation with structural analysis," en, in *Proc. IEEE European Test Symp*, 2021.

[18] Z. Gao, S. Malagi, M. Hu, *et al.*, "Application of cell-aware test on an advanced 3nm cmos technology library," en, in *Proc*, IEEE International Test Conf, 2019.

[19] R. Guo, B. Archer, K. Chau, and X. Cai, "Efficient cell-aware defect characterization for multi-bit cells," en, in *Proc. IEEE International Test Conf. in Asia*, 2018.

[20] S. Mhandi, A. Virazel, P. Girard, *et al.*, "Towards improvement of mission mode failure diagnosis for system-on-chip," en, in *Proc. IEEE International On-Line Testing Symposium*, 2019.

[21] S. Mhandi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "Cell-aware diagnosis of automotive customer returns based on supervised learning," en, in *IEEE Automotive Reliability and Test Workshop*, 2019.

[22] S. Mhamdi, P. Girard, A. Virazel, A. Bosio, E. Faehn, and A. Ladhar, "Cell-aware defect diagnosis of customer returns based on supervised learning," en, *IEEE Trans. on Device Material and Reliability*, vol. 20, no. 2, 2020.

[23] S. Mhandi, P. Girard, A. Virazel, A. Bosio, and A. Ladhar, "Learning-based cell-aware defect diagnosis of customer returns," en, in *Proc. IEEE European Test Symposium*, 2020.

[24] ——, "Cell-aware diagnosis of customer returns using bayesian inference," en, in *Proc. IEEE International Symposium on Quality Electronic Design*, 2021.

[25] L. I. Rodríguez Gómez, "Machine learning support for logic diagnosis," en, Ph.D. dissertation, Universität Stuttgart, 2017.

[26] M. Renovell, P. Huc, and Y. Bertrand, "Cmos bridging fault modeling," in *Proceedings of IEEE VLSI Test Symposium*, 1994, pp. 392–397. DOI: 10.1109/VTEST.1994.292283.

[27] V. Sar-Dessai and D. Walker, "Accurate fault modeling and fault simulation of resistive bridges," in *Proceedings 1998 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (Cat. No.98EX223)*, 1998, pp. 102–107. DOI: 10.1109/DFTVS.1998.732156.

[28] S. Millman and J. Garvey Sr, "An accurate bridging fault test pattern generator," de, in *Proc. Int*, Test Conf, 1991, pp. 411–418.

[29] M. Renovell, P. Huc, and Y. Bertrand, *The Concept of Resistance Interval: A New Parametric Model for Realistic Resistive Bridging Fault*, en. IEEE VLSI Test Symp, 1995, pp. 184–189.

[30] M. Dalpasso, M. Favalli, P. Olivio, and B. Ricco, "Parametric bridging fault characterization for the fault simulation of library-based ics," en, in *Proc. Int*, Test Conf, 1992, pp. 486–495.

[31] E. McCluskey and C.-W. Tseng, "Stuck-fault tests vs. actual defects," de, in *Proc. Intl. Test Conf*, 2000, pp. 336–343.

[32] S. Ma, P. Franco, and E. McCluskey, "An experimental test chip to evaluate test techniques experimental results," it, in *Proc. Intl. Test Conf*, 1995, pp. 663–672.

[33] J. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition fault simulation," fr, *IEEE Design & Test Comput*, vol. 4, pp. 32–38, 1988.

[34] C.-W. Tseng and E. McCluskey, "Multiple-output propagation transition fault test," en, in *Proc. Intl Test Conf*, 2001, pp. 358–366.

[35] L. Wang, C. Wu, and X. Wen, *VLSI Test Principles and Architectures*, es. Design for Testability. Elsevier, 2006.

[36] C. Liu and K. Chakrabarty, "Compact dictionaries for fault diagnosis in scan-bist," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 775–780, 2004. DOI: 10.1109/TC.2004.4.

[37] S. Venkataraman and S. Drummonds, "Poirot: A logic fault diagnosis tool and its applications," in *Proceedings International Test Conference 2000 (IEEE Cat. No.00CH37159)*, 2000, pp. 253–262. DOI: 10.1109/TEST.2000.894213.

[38] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "Derric: A tool for unified logic diagnosis," in *12th IEEE European Test Symposium (ETS'07)*, 2007, pp. 13–20. DOI: 10.1109/ETS.2007.16.

[39] H.-J. Wunderlich, M. Elm, and S. Holst, "Debug and diagnosis: Mastering the life cycle of nano-scale systems on chip (invited paper)," en, in *Proceedings of 43rd International Conference on Microelectronics, Devices and Material with the Workshop on Electronic Testing (MIDEM'07)*, MIDEM, 2007, pp. 27–36.

[40] E. McCluskey, "Quality and single-stuck faults," in *Proceedings of IEEE International Test Conference - (ITC)*, 1993, pp. 597–. DOI: 10.1109/TEST.1993.470645.

[41] J. Geuzebroek, E. J. Marinissen, A. Majhi, A. Glowatz, and F. Hapke, "Embedded multi-detect atpg and its effect on the detection of unmodeled defects," in *2007 IEEE International Test Conference*, 2007, pp. 1–10. DOI: 10.1109/TEST.2007.4437649.

[42] F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, and S. Eichenberger, "Defect-oriented cell-aware atpg and fault simulation for industrial cell libraries and designs"," en, in *Proc. IEEE International Test Conference*, 2009.

[43] X. Lin, W.-T. Cheng, T. Kobayashi, and A. Glowatz, "On modeling cmos library cells for cell internal fault test pattern generation," in *2021 IEEE 30th Asian Test Symposium (ATS)*, 2021, pp. 103–108. DOI: 10.1109/ATS52891.2021.00030.

[44] R. Blanton and J. Hayes, "Properties of the input pattern fault model," in *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, 1997, pp. 372–380. DOI: 10.1109/ICCD.1997.628897.

[45] A. Jas, S. Natarajan, and S. Patil, "The region-exhaustive fault model," in *16th Asian Test Symposium (ATS 2007)*, 2007, pp. 13–18. DOI: 10.1109/ATS.2007.78.

[46] E. Burns, *What is machine learning and why is it important?* Mar. 2021. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML.

[47] *1.1. linear models*. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html.

[48] *1.4. support vector machines*. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html.

[49] *1.6. nearest neighbors*. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html.

[50] *1.7. gaussian processes*. [Online]. Available: https://scikit-learn.org/stable/modules/gaussian_process.html#.

[51] *1.9. naive bayes*. [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html.

[52] *1.10. decision trees*. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html.

[53] *1.17. neural network models*. [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[54] Silvaco. "Nanometer library characterization: Challenges and solutions"." en. (2019).

[55] M. White Paper, "Improving library characterization with machine learning"," en, 2018.

[56] *Unified library characterization tool leverages machine learning in the cloud"*, en, White Paper, 2018.

[57] S. Venkataraman and S. Drummonds, "A technique for logic fault diagnosis of interconnect open defect"," fr, in *Proc. IEEE VLSI Test Symp*, 2000.

[58] C.-M. Li and E. McCluskey, "Diagnosis of resistive-open and struck-open defects in digital cmos ics"," en, *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1748–1759, 2005.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, "A learning-based methodology for accelerating cell-aware model generation," en, in *Proc. IEEE/ACM Design Au-*

*tomation and Test in Europe*, 2021, pp. 1580–1585. DOI: 10.23919/DATE51398.2021.9474227.

[61] J. Tikkanen, N. Sumikawa, L.-C. Wang, and M. Abadir, "Multivariate outlier modeling for capturing customer returns – how simple it can be," en, in *Proc. IEEE On-Line Test Symposium*, 2014.

[62] A. Bosio, P. Girard, S. Pravossoudovitch, and A. Virazel, "A comprehensive framework for logic diagnosis of arbitrary defects," en, *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 289–300, 2010.

[63] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and O. Riewer, "A comprehensive system-on-chip logic diagnosis," pt, in *Proc. IEEE Asian Test Symposium*, 2010.

[64] L. Huisman, "Diagnosing arbitrary defects in logic designs using the single location at a time (SLAT)," en, *IEEE Trans. on Computer-Aided Design*, vol. 23, no. 1, p. 91, 2004.

[65] S. Holst and H.-J. Wunderlich, "Adaptative debug and diagnosis without fault dictionaries," en, in *Proc. IEEE European Test Symposium*, 2007.

[66] J. Waicukauski and E. Lindbloom, "Failure diagnosis of structured vlsi," en, *IEEE Design & Test of Computers*, vol. 6, no. 4, pp. 49–60, 1989.

[67] M. Abramovici, P. Menon, and D. Miller, "Critical path tracing-an alternative to fault simulation," en, in *Proc. ACM Design Automation Conference*, 1983.

[68] P. Girard, C. Landrault, and S. Pravossoudovitch, "Delay-fault diagnosis by critical-path tracing," en, *IEEE Design & Test of Computers*, vol. 9, no. 4, pp. 27–32, 1992.

[69] Y. Xue, X. Li, R. Blanton, and C. Lim, "Improving diagnostic resolution of failing ics through learning," en, *IEEE Trans. on Computer-Aided Design*, vol. 37, no. 6, pp. 1288–1297, 2018.

[70] F. Liu, P. Nikolov, and S. Ozev, "Parametric fault diagnosis for analog circuits using a bayesian framework," pt, in *Proc. IEEE VLSI Test Symposium*, 2006.

[71] S. Wang and W. Wei, "Machine learning-based volume diagnosis," en, in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2009.

[72] J. Nelson, W. Tam, and R. Blanton, "Automatic classification of bridge defects," en, in *Proc. IEEE International Test Conference*, 2010.

[73] L.-C. Wang, "Data learning based diagnosis," en, in *Proc. ACM/IEEE Asia and South Pacific Design Automation Conference*, 2010.

[74] R. Blanton, "Dreams: Dfm rule evaluation using manufactured silicon," en, in *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 2013.

[75] Y. Xue, O. Poku, X. Li, and R. Blanton, "Padre: Physically-aware diagnostic resolution enhancement," en, in *Proc. IEEE International Test Conference*, 2013.

[76] R. Tikkanen, S. Siatkowski, L.-C. Wang, and M. Abadir, "Yield optimization using advanced statistical correlation methods," en, in *Proc. IEEE International Test Conference*, 2014.

[77] X. Ren, M. Martin, and R. Blanton, "Improving accuracy of on-chip diagnosis via incremental learning," pt, in *Proc. IEEE VLSI Test Symposium*, 2015.

[78] Y. Huang, W. Yang, and W. Cheng, "Advancements in diagnosis driven yield analysis: A survey of state-of-the-art scan diagnosis and yield analysis technologies," en, in *Proc. IEEE European Test Symposium*, 2015.

[79] Y. Xue, X. Li, R. Blanton, and C. Lim, "Diagnosis resolution improvement through learning-guided physical failure analysis," en, in *Proc. IEEE International Test Conference*, 2016.

[80] Q. Huang, C. Fang, and R. Blanton, "Diagnosis outcome prediction on limited data via transferred random forest," en, in *Proc. IEEE International Test Conference*, Asia, 2020.

[81] ——, "Laidar: Learning for accuracy and ideal diagnostic resolution," en, in *Proc. IEEE International Test Conference*, 2020.

[82] R. Pan, Z. Zhang, X. Li, K. Chakrabarty, and X. Gu, "Unsupervised root-cause analysis for integrated systems," en, in *Proc. IEEE International Test Conference*, 2020.

[83] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," pt, *Informatica*, vol. 31, no. 3, 2007.

[84] Webpage, *User guide: Contents — scikit-learn 0.24.1 documentation"*, fr. [Online]. Available: `http://scikit-learn.org/stable/user_guide.html`..

[85] H. Zhang, "The optimality of naive bayes"," en, in *Proc. 17th International Florida Artificial Intelligence Research Society Conference*, May 2004.

[86] Webpage, *1.9. naive bayes — scikit-learn 0.24.1 documentation"*, en. [Online]. Available: `https://scikit-learn.org/stable/modules/naive_bayes.html`..

[87] M. Fieback, L. Wu, G. C. Medeiros, *et al.*, "Device-aware test: A new test approach towards dppb level," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–10. DOI: `10.1109/ITC44170.2019.9000134`.

[88] P. d'Hondt, A. Ladhar, P. Girard, and A. Virazel, "A comprehensive learning-based flow for cell-aware model generation," in *2022 IEEE International Test Conference (ITC)*, 2022.

[89] P. d'Hondt, S. Mhamdi, P. Girard, A. Virazel, and A. Ladhar, "A comprehensive framework for cell-aware diagnosis of customer returns," *Microelectronics Reliability*, vol. 135, p. 114 595, 2022, ISSN: 0026-2714. DOI: `https://doi.org/10.1016/j.microrel.2022.114595`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0026271422001196`.