Cell-Aware Model Generation Using Machine Learning

Pierre D'hondt, Aymen Ladhar, Patrick Girard, Arnaud Virazel

Chapter 1
Cell-Aware Model Generation by Using Machine Learning

Pierre d'Hondt, Aymen Ladhar, Patrick Girard, Arnaud Virazel

Abstract

Characterizing cell-internal defects of standard cell libraries is an essential step to ensure high test and diagnosis quality. However, such a characterization process, called *cell-aware model generation*, usually resorts to extensive electrical defect simulations that are costly in terms of run time and utilization of SPICE simulator licenses. Typically, the generation time of cell-aware models for few hundreds of cells may reach up to several months considering a single SPICE license. This chapter presents a methodology that does not use any electrical defect simulation to predict the response of a cell-internal defect once it is injected in a standard cell. More widely, this methodology uses existing cell-aware models (generated from electrical simulations) from various standard cell libraries and technologies to predict cell-aware models (learning-based) for new standard cells independently of the technology. Experiments done on several industrial cell libraries using different technologies demonstrate the accuracy and performance of the prediction method.

1 Introduction

Digital Integrated Circuits (ICs) are commonly synthesized with pre-defined libraries of standard cells of various nature and complexity. As the semiconductor industry moves to increasingly smaller geometries, new types of manufacturing defects appear and need to be targeted by industrial test flows. 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, these 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 within the logic cell structures. They are called *intra-cell* or *cell-internal defects* [1-3]. 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 [4].

Pierre d'Hondt
STMicroelectronics, 38920 Crolles, FR, and LIRMM, University of Montpellier / CNRS, 34095, Montpellier, FR, e-mail: pierre.dhondt@st.com

Aymen Ladhar
STMicroelectronics, 38920 Crolles, FR, e-mail: aymen.ladhar@st.com

Patrick Girard, Arnaud Virazel
LIRMM, University of Montpellier / CNRS, 34095, Montpellier, FR
E-mail: patrick.girard@lirmm.fr, arnaud.virazel@lirmm.fr

Cell-Aware (CA) test and diagnosis have been proposed recently to target those subtle defects in ICs requiring highest product quality [5-9]. 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 [10-11]. 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) [12-13].

One bottleneck of CA model generation is that it requires extensive computational efforts to characterize all standard cells of a library [14-15]. 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 ICs and make it a standard in the qualification process of silicon products [16]. To this end, Machine Learning (ML) can be used to drastically accelerate the CA model generation flow.

This chapter presents a comprehensive flow experimented on industrial cell libraries and preliminary introduced in [17]. The flow is based on a learning method that uses existing CA models of various standard cells developed using different technologies to predict CA models for new standard cells independently of the technology. This is the first work to address this problem since previous works on ML focused on cell library characterization without defect injection [18-20]. Experiments performed on a standard cell population of reasonable size (about two thousands of cells from different technology nodes and transistor sizes) show that the generation time of CA models can be reduced by more than 99% (a few hours instead of almost 3 months when CA models are generated using a single SPICE license). Part of these results are extracted from [17] in which the proposed flow has been experimented on combinational cells of industrial libraries.

The remainder of this chapter is organized as follows. Section 2 gives some background on standard cell characterization, first for design purpose, and then for test and diagnosis purpose. The last part of the section explains why using ML for cell characterization can help reducing the generation time of CA models. Section 3 presents the ML-based CA model generation flow and details the two main steps of the flow, namely the generation of training data and the generation of new data. Section 4 shows how cell transistor netlists and cell-internal defects are represented and manipulated by the proposed methodology. Section 5 presents experimental results gathered on industrial cell libraries and proposes a performance comparison with a simulation-based approach. Section 6 presents the hybrid CA model generation flow developed for an industrial usage of the ML-based methodology. Section 7 summarizes the contribution and concludes the chapter.

2 Background on Standard Cell Characterization

2.1 Standard Cell Characterization for Design Purpose

Digital circuit designers use pre-defined standard cells to synthesize circuits with various sizes and complexities [21-27]. As the simulation of a full circuit design can take a huge amount of time, designers rely on standard cell characterization, a process that produces simple models of functionality, timing, and power consumption at the cell level. The (simplified) design and characterization flow for a standard cell is summarized in Fig. X.0. It starts with the functional specification, which describes the logical function of the cell (AND, flip-flop, etc.) by using a Hardware Description Language (HDL). The next step defines the cell's transistors and their connections in a SPICE netlist. This netlist is known as the cell's schematic or structure. The layout describes the physical implementation of the cell on silicon, using several layers and materials (metal, polysilicon, …) [28], and is designed from the SPICE netlist. A parasitic extraction is then performed on the obtained layout, in order to specify the parasitic resistors and capacitors introduced in the physical implementation. The parasitic components are appended to the SPICE transistors netlist in the Detailed Standard Parasitic Format (DSPF).

Cell characterization for design purpose uses the generated cell descriptions (also called cell views) to perform electrical simulations of standard cells and extract the power and delay information, as well as the identification of timing constraints (setup and hold times). Typically, cell characterization requires the definition of global parameters such as Process, Voltage, and Temperature, known as PVT corners, and global constraints such as wire loads and time limits for transitions. The cell schematic and layout are iteratively modified until quality and constraint requirements are met in terms of functionality, timing, and power consumption.

Once done, data describing every aspect (transition time, internal power, capacitance, sequential cells constraints, etc.) of the cell are written to dedicated files, known as cell models. By using electrical simulations considering different values of the global parameters, cell models are created to determine the behavior of standard cells in every condition that may occur during the lifetime of the circuit.

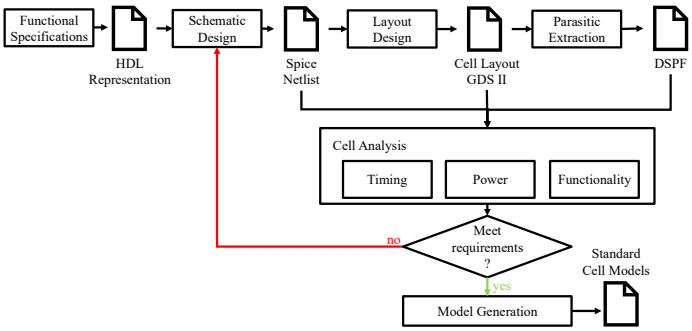


Fig. X.0 Schematized process of standard cell creation and characterization

2.2 Cell Internal Defect Universe

The first step of a standard cell characterization process for test and diagnosis purpose (i.e., CA model generation) is to extract all potential and realistic defects within each cell to be able to simulate their effect in a defective cell [29-33].

Figure X.1 gives an example of internal defects that may occur at the cell level. These defects can be classified according to two main categories:

- *Transistor defects*, which are defects occurring at the transistor ports (source, drain, gate and bulk). These defects can be modeled as short or open defects at the transistor ports. As illustrated in Fig X.1.a, for a CMOS transistor, six shorts (gate-drain, source-drain, gate-source, and each port to bulk) and three open defects (gate, source and drain) can be identified. These nine defects are added to the potential defects list for every transistor in the standard cell.
- *Inter-transistor defects*, which are defects occurring at the interconnexion between two different transistors. These defects can also be modeled as short or open defects between two internal nodes. Their existence is bound to the actual layout of the cell (e.g., two close polygons may be defectively shorted), so inter-transistors defects require layout extraction to be identified.

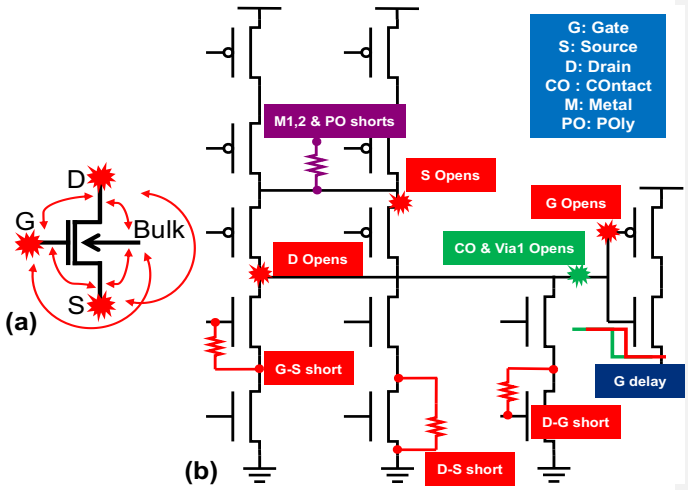


Figure X.1: (a) Illustration of the six short defects and three open defects that can affect a CMOS transistor's ports (b) Example of cell-internal defects in a simple structure made of various transistors

Figure X.2 gives an example of inter-transistor defects and their locations on the cell layout. There are two possible solutions to extract inter-transistor defects. The first one consists in reading the layout database of each standard cell and creating a SPICE transistor netlist in the DSPF format including parasitic elements like resistors and capacitors. These elements represent the list of inter-transistor defects to be considered during the characterization. A parasitic capacitor exists between two polygons that are supposed not to be connected. Consequently, the location of a potential short defect and a defective resistor can become an open defect. Even if this method is easy to apply, its main drawbacks are the huge number of parasitic elements listed by the DSPF netlist (on average, 61 times the number of transistors in the cell) and the fact that some of these parasitic elements cannot be considered as realistic defect locations (e.g., the distance between two nets may be large enough to ensure non-defective manufacturing but still described by a small value parasitic capacitor, some layers are not sensitive to open defects but still described with their own resistors, etc.). In addition, several parasitic elements are equivalent, and there is no solution to recognize them without characterization (e.g., a single physical net is described by several serial resistors and any defect on one of these resistors is equivalent to a defect on the whole net).

To address these limitations, a second method based on Design Rule Checking (DRC) can be used. This solution allows the localization of neighbored internal nets as well as the localization of potential open defects that can be identified for the cell

characterization. The DRC-based method limits the number of potential defect locations to 4.3 times the number of transistors in the cell, on average.

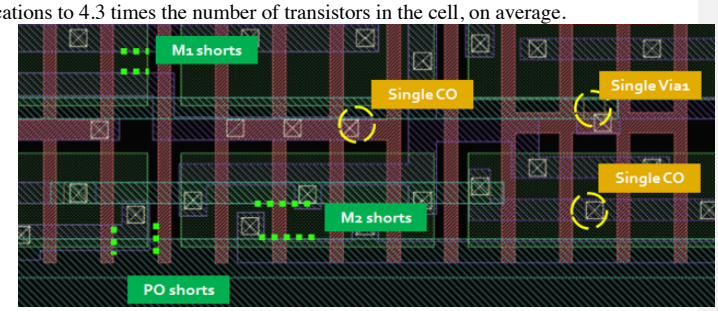


Fig. X.1: Example of inter-transistor defects

2.3 Standard Cell Characterization for Test and Diagnosis Purpose

A typical CA model generation flow, as shown in Fig. X.3, has as input a SPICE netlist representation of a standard cell which is usually derived from a layout description, e.g., a GDSII file. This DSPF cell netlist is then used by an electrical simulator to simulate each potential defect against an exhaustive set of stimuli. Those stimuli include static (one vector) and dynamic (two vectors) input patterns of the cell (called cell-patterns in the sequel). Once the simulation is completed, all cell-internal defects are classified into defect equivalence classes with their detection information (required input values for each defect within each cell) and are synthetized into a CA model. As standard cells may have more than ten inputs, and thousands of cells with different complexities are usually 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.

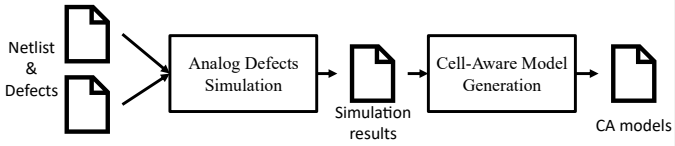


Figure X.2 Conventional cell-aware model generation flow

Once the CA model of a given standard cell is generated, it can be used either for Automatic Test Pattern Generation (ATPG) or for fault diagnosis:

- *ATPG usage*. Using the CA models, which is a dictionary mapping cell-patterns to the cell-internal faults they detect, an ATPG tool identifies for each cell in the CUT the minimum set of stimuli detecting all cell-internal defects. Then, it generates test patterns exercising this test stimuli at the input pins of the cell under test and ensures the fault propagation to an observation point.
- *Fault diagnosis*. A diagnostic tool extracts the failing and passing logic values at the input pins of the defective cell. This information is then matched with the CA model of the defective cell in order to identify the suspect internal defect.

2.4 Cell-Aware Model Generation: A Machine-Learning Friendly Process

Machine learning can be used to significantly accelerate the CA model generation process. The motivation behind the use of ML is the result of several observations made while performing comparisons between several CA models coming from different standard cell libraries and technologies:

- Several cell-internal defects, such as stuck-open defects, are independent of the technology and transistor size [34-35].
- For the same function, two cell-internal structures are usually quite similar for two different technologies.
- 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 the ML-based prediction.
- Very simple CA models are used to emulate short and open defects, for which resistance values are often identical for all technologies.
- A large database of CA models is usually available and can be used to train a ML algorithm.

All these observations intuitively indicate that CA model generation through ML is possible. However, the first challenging task is to be able to describe cell transistor netlist as well as corresponding cell-internal defects in a uniform (standardized) manner, so that a ML algorithm can learn and infer from data irrespective of their

incoming library and technology. Indeed, similar cells (e.g., cells with same logic function, same number of inputs and same number of transistors) may be described differently in transistor-level (SPICE) netlists of various libraries (e.g., a transistor label does not always correspond to the same transistor in two similar cells coming from two different libraries). It is therefore mandatory to standardize the description of cells and corresponding defects for the ML-based defect characterization methodology. Heuristic solutions developed to this purpose are described in Section 4. The second challenging task is 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 is used to this purpose.

3 Learning-Based Cell-Aware Model Generation Flow

The learning-based CA model generation flow initially introduced in [17] is used to predict the behavior of a cell (combinational or sequential) when affected by intra-cell defects. The flow is sketched in Fig. X.4. It is based on supervised learning that takes a set of input data and known responses (*labeled data*) used as training data, trains a model to classify those data, and then uses this model to predict (*infer*) the class of new data.

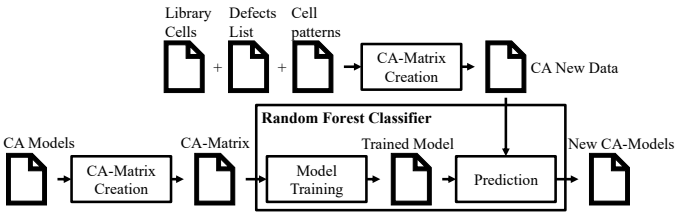


Fig. X.4: Generic view of the ML-based CA model generation flow

Figure X.4 depicts the **two main steps** of the **supervised learning process** used for ML-based CA model generation. A Random Forest Classifier is used 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.
The **first main step** of the CA model generation 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 the question: "Does this stimulus detects this defect affecting this cell?". Answering this question allows obtaining a new CA model for a given standard cell.

3.1 Generation of Training Data

Training data are made of various and numerous CA models formerly generated by relying to brute-force electrical defect simulations. For each cell (combinational or sequential) 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.

The CA-matrix creation flow is depicted in Fig. X.5. The flow starts by rewriting the CA model so that it can be ML friendly. To this end, the CA model file is parsed and its content is organized into a matrix which contains numbers and categories of certain values (more details are given later on). Then, it 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 is created with the above information.

Commenté [AV1]: Peux-tu donner en une phrase de plus en quoi cela consiste

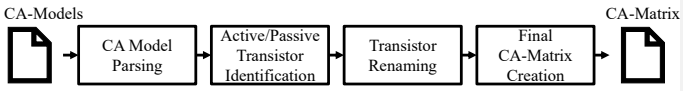


Fig. X.5: CA-matrix creation flow

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

- *Cell patterns and responses*. This gives the values applied on inputs (A, B) of the cell 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 the 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 (1), passive (0), switching to active state (R), switching to passive state (F).

- *Defect description*. This gives information about defect locations inside the cell transistor schematic. This part contains a column for each transistors' ports. In Table I, 'N1_D' stands for the drain port of the NMOS transistor named N1, and 'N1_S' for its source port. In these columns, a '1' (resp. '0') indicates that the port is concerned (resp. non-concerned) by the described defect. For example, D15 is a short between the drain and the source of transistor N1, so columns 'N1_D' and 'N1_S' contains a one, while other columns are filled with zeros. The name and type of each defect are also given in this description. The matrix also includes rows describing the cell with no defects ('free'). This is presented in more detail in Section 4.4.
- *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 cell pattern.

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

TABLE I. 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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

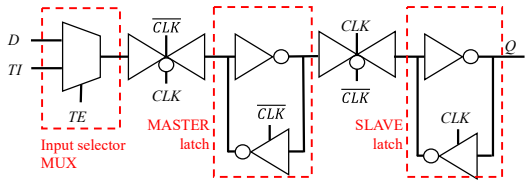


Fig. X.6: Block-level representation of a scan flip-flop example

In order to illustrate the various steps of the CA-matrix creation flow in the case of sequential cells, let us consider the block-level representation of a scan flip-flop as depicted in Fig. X.6. 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 II. 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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table II shows an example of a training dataset for the scan flip-flop shown in Fig. X.6. It is composed of four types of information:

- *Cell inputs and 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 provides all the possible input stimuli that can be applied to the cell. For the sake of readability, they are represented partially in Table II. 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 (1), passive (0), switching to active state (R), switching to passive state (F), pulsing (P), anti-pulsing (A).
- *Defect description*. This gives information about all defect locations in the cell transistor schematic. In Table II, "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.

As for combinational cells, the first three types of information are used as inputs for the ML algorithm.

3.2 Generation of New Data

New data represent the cells to be characterized and are obtained for each standard cell from the cell description, the corresponding list of defects and the 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.

4 Cell and Defect Representation in the Cell-Aware Matrix

This section details the various steps required to represent a standard cell in a CA-matrix. The starting point of this process is a transistor-level (SPICE) netlist of the standard cell. The CA-matrix must be accurate enough to clearly identify each transistor and each net of the cell transistor schematic. This description also associates each transistor to its sensitization patterns and reports the output response for each cell-pattern. For this reason, the cell description process requires several successive operations that are detailed below. Note that this process is applied to all cells in a library to be characterized.

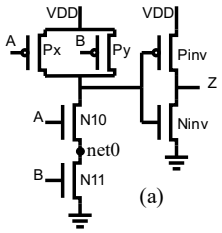
4.1 Identification of Active, Passive and Pulsing Transistors

The first step consists in identifying active and passive transistors in the cell netlist with respect to an input stimulus. To this purpose, a single golden (defect-free) electrical simulation of the cell to be characterized is first performed. By monitoring the voltage of cell's transistors gates, active and passive transistors are identified for each input stimulus (cell-pattern). An active NMOS (resp. PMOS) transistor is a transistor with a logic-1 (resp. logic-0) value measured on its gate port. A passive NMOS (resp. PMOS) transistor is a transistor with a logic-0 (resp. logic-1) value measured on its gate port. Note that for sequential cells, an active NMOS (resp. PMOS) transistor is a transistor with a logic-1 (resp. logic-0) value appearing on its gate port 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 port 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 port 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. This simulation also provides the cell output value. With this information, each cell pattern can be associated to the list of active transistors in the cell. After this step, the CA-matrix contains the following columns:

- **Cell inputs & responses columns.** They contain all input stimuli (cell patterns) that can be applied to the cell, and the corresponding responses.
- **Transistor switching activity columns.** 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). Note that 'P' and 'A' are only used for sequential cells. Since PMOS and NMOS transistors are activated in opposite way, the '-' character is used before the PMOS values.

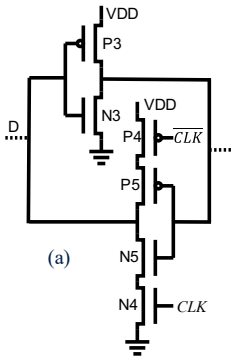
Figure X.7 shows (a) the transistor schematic of a 6-transistor AND2 cell and (b) a partial representation of the CA-matrix of the cell. Columns A and B list all the possible input stimuli for this cell. For each stimulus, active and passive information about each transistor of the cell is entered in the CA-matrix. For example, AB=00 leads to two active PMOS transistors and two passive NMOS transistors in the NAND2 block and one passive PMOS transistor and one active NMOS transistor in the output inverter.



A	B	Px	Py	Pinv	N10	N11	Ninv
0	0	-1	-1	0	0	0	1
0	1	-1	0	0	0	1	1
1	0	0	-1	0	1	0	1
1	1	0	0	-1	1	1	0

(b)

Fig. X.7: Example of a AND2 cell: (a) cell transistor schematic and (b) partial CA-matrix representation



D	CLK	P3	P4	P5	N3	N4	N5
0	P	-1	P	0	0	P	1
1	P	0	P	-1	1	P	0
…	…	…	…	…	…	…	…
R	P	-F	P	-R	R	P	F

(b)

Fig. X.8: Example of a LATCH structure: (a) cell transistor schematic and (b) partial CA-matrix representation

Figure X.8.a represents the transistor schematic of a LATCH such as the ones used inside the scan flip-flop depicted in Fig. X.6. In the partial representation of the CA-matrix of the latch (Fig. X.8.b), columns D and CLK list all the possible input stimuli for this structure.

4.2 Renaming of Transistors

In the CA model generation flow, the goal is to train a ML algorithm using this representation of standard cells coming from different libraries and technologies. However, this matrix representation is dependent on the transistor names and the order they are defined in the SPICE netlist. Two standard cells having the same schematic may have different transistor naming and the order of transistors in the SPICE netlist may differ as well. This is because standard cell libraries are created several months or years apart, by different teams, with sometimes new guidelines in terms of best practices. 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 mitigate this issue, a second step consisting in renaming all cell transistors independently of their initial names and order in the input SPICE netlist is required. The algorithm developed to this purpose is detailed in the following.

In order to ensure that the CA-matrix is unique for a given cell and that the CA-matrices of two cells having the same structure have identical transistor switching activity columns (i.e. they have the same transistor names irrespective of their incoming library and technology), a transistor renaming procedure is required. The

first step consists in sorting the transistors of a standard cell in an algorithmic way that only depends on the cell's *transistors structure*. A transistors 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. Once the transistors are sorted, they are consistently and unambiguously renamed. The transistor-renaming algorithm consists of the following two steps: determination of branch equations and sorting of branch equations.

4.2.1 *Determination of Branch Equations*

The transistors structure of a standard cell is composed of one or more branches. A branch is a group of transistors connected by their drain and source ports. The entry (or gate) of each branch is the set of transistor gates and its exit (or drain) is the connection net between the NMOS and PMOS transistors, which drives the gate of the next branch. A branch's source 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.

Sequential cells and complex combinational cells tend to integrate transmission gates in their structures. 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 transistors structure. The entry of such a branch is the set of transistor gates plus the exit of the previous branch, and the exit is the entry of the following branch.

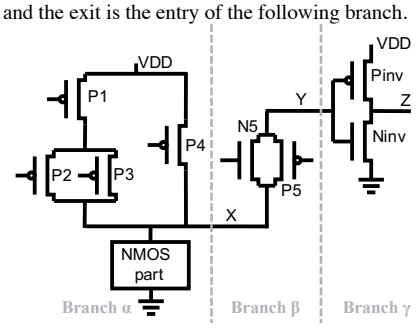


Fig. X.9: Example schematic

Figure X.9 shows an example of transistors structure. It is composed of three branches, named as α, β and γ. The two-transistors output-inverter is the simplest

Commenté [AV2]: Tu devrais les mettre en évidence sur la Figure.

branch whose input is net Y and output is net Z (branch γ). The inverter creates two paths between the branch output and the power nets, so its branch equation is (N_{inv}!P_{inv}). The equation of the left-most branch α (PMOS branch driving net X) is (P4|(P1&(P2|P3))). In order not to 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. X.9 is therefore (1p|(1p&(1p|1p))). These two branches are separated by the β branch, i.e., a transmission gate composed of N5 and P5, which is anonymized as '1t'.

4.2.2 *Sorting of Branch Equations*

Once all the branch equations for the considered cell have been determined, they are sorted by using the following 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* - in alphabetical order.

Table III reports all the branch equations of the schematic in Fig. X.9 sorted according to the above criteria.

TABLE III. BRANCH EQUATIONS FOR THE SCHEMATIC OF FIG.X.9

Level	Number of transistors	Anonymized equation	Comment
1	2	(1n|1p)	Branch α Inverter
2	2	1t	Branch β Transmission gate
3	4	(1p|(1p&(1p|1p)))	Branch γ PMOS structure

4.3 Identification of Parallel Transistors

Because of parallel transistors, the identification of branch equations is not enough to unambiguously rename all transistors. Specifically, two or more parallel transistors in a branch share the same drain and source, making their identification quite difficult. For example, transistors P2 and P3 in Fig. X.9 can be either represented as “P2|P3” or as “P3|P2”, thus leading to a confusing situation. A solution to solve this problem 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. This value summarizes the states of the transistor (active, passive, pulsing) for all possible stimuli applied to the cell. The input stimuli range from (0…,0) to (1…,1) for combinational cells and from (P,0…,0) to (P,1…,1) for sequential cells. For each of these stimuli, the transistor is either active (1), passive (0), pulsing (P) or anti-

pulsing (A). The activity value is defined 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 compute the activity values, one needs to know whether the transistor is active or passive for each input stimulus. This information is already available in the CA-matrix as described in Section 3. To illustrate this process, activity values for the transistors of the AND2 cell given in Fig. X.7 are listed in Table IV.

TABLE IV. ACTIVITY VALUES FOR THE AND2 CELL IN FIG.X.7

Commenté [AV3]: Utilises le même style pour les tableau. Le précédant a un contour en trait épais.

			Old names					
A	**B**	**Comments**	**Px**	**Py**	**N10**	**N11**	**Pinv**	**Ninv**
0	0	First stimulus	1	1	0	0	0	1
0	1		1	0	0	1	0	1
1	0		0	1	1	0	0	1
1	1	Last stimulus	0	0	1	1	1	0
Activity value			1100	1010	0011	0101	0001	1110
			↓ Renaming ↓					
			P2	P1	N0	N1	P0	N2

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 AND2 cell in Fig. X.7, the renaming process is illustrated in Table IV.

The whole transistors renaming process for the transistors of the LATCH structure presented in Fig. X.8 is summarized in Table V, starting with the structure branches extraction and sorting, the computation of activity values, then the renaming process itself.

TABLE V. ACTIVITY VALUES FOR THE LATCH STRUCTURE IN FIG.X.8

Branches Extraction and sorting (left to right)	Level		1			2		
	Number of transistors		2			4		
	Anonymized equation		(1n|1p)			(1n&1n|1p&1p)		
	CLK	D	N3	P3	N4	N5	P4	P5

Activity Value within branches	P	0	0	1	P	1	P	0
	P	1	1	0	P	0	P	1
	Activity value		01	10	PP	01	PP	01
Renaming of transistors			↓ Renaming ↓					
			N1	P1	N3	N2	P3	P2

4.4 Defect Representation in the Cell-Aware Matrix

To describe cell-internal defects in a standardized and ML friendly manner, the CA-matrix contains a set of categorical columns representing the cell's transistors' ports. Cell internal defects are classified into:

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

TABLE VI. Example of defect columns for the AND2 presented in Fig. X.7

Py_S	Py_D	Px_S	Px_D	…	N10_S	N10_D	N11_S	N11_D	Comment
0	0	0	0	…	1	1	0	0	source-drain short on N10
1	0	1	0	…	1	0	0	1	net0 & VDD short

Table VI is an example of defect description in the CA-matrix of the AND2 cell in Fig. X.7. The row with red cells describes the intra-transistor short defect between drain and source ports of transistor N10 (newly N0). The row with purple cells describes the inter-transistor short defect between VDD at PMOS sources and "net0" (net0 connects N10-source and N11-drain).

5 Validation on Industrial Cell Libraries

The ML-based CA model generation flow has been implemented in a python program. The ML algorithms were taken from the publicly available python module

called scikit-learn [35]. A dataset composed of 1712 combinational standard cells coming from standard cell libraries developed using three technologies – C40 (446 cells), 28SOI (825 cells) and C28 (441 cells) – was assembled. Another dataset composed of 219 sequential cells coming from the same libraries and technologies – C40 (27 cells), 28SOI (108 cells) and C28 (84 cells) – was also used for validation. All these cells already had a CA model generated by a commercial tool. The CA-matrix was generated for each cell. The flow was experimented in two different ways. First, the ML model was trained and evaluated using cells belonging to the same technology. Second, the model was trained on one technology and evaluated on another one. Combinational and sequential cells were considered separately. Part of these results are extracted from [17].

Commenté [AV4]: Ajouter une reference.

5.1 Predicting Defect Behavior on the Same Technology

5.1.1 Combinational Standard Cells

The ML model was first trained on cells of the 28SOI standard cell library. Cells were grouped according to their number of transistors and inputs. For *m* cells available in a given group, the ML model was trained over *m-1* cells and its **prediction accuracy** was evaluated 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. All possible open and short defects (static and dynamic) were considered for each cell. Results presented in Tables VII report the prediction accuracy for open defects. Results achieved for short defects are similar.

TABLE VII. Average Prediction Accuracy for combinational cells in the same technology

Number of transistors	Predicion 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 VII presents the prediction accuracy achieved for open defects. For the sake of conciseness, only results for cells with less than 7 inputs and 48 transistors are

reported, although experiments have been done on cells with up to 8 inputs and 112 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 model can perfectly predict the defective behavior of at least one cell. In contrast, white background indicates that no cell was perfectly predicted in that group (all prediction accuracies are less than 100%). For example, let us consider the circled box in Table VII, that corresponds to 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%.

5.1.2 Sequential Standard Cells

For these experiments, the ML model was trained on a group of sequential standard cells coming from C40 standard cell libraries. Cells were grouped according to their number of transistors and inputs. As for combinational cells, for *m* cells available in a given group, the ML model was trained over *m-1* cells and the **prediction accuracy** was evaluated 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. All possible open and short defects (static and dynamic) in each cell were considered. Results in Table VIII report the prediction accuracy for short defects. Results achieved for open defects are similar.

TABLE VIII. Average Prediction Accuracy for sequential cells in the same technology

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
	50				100
	52				100

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 sequential cells**. Empty boxes mean that there is zero or one cell available and that the group cannot be evaluated. As can be seen in Table VIII, the maximum prediction accuracy (100%) was always obtained 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 transistors structure 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. The goal of the next subsection is to leverage on existing CA models to generate CA models for a new technology.

5.2 Predicting Defect Behavior on Another Technology

5.2.1 Combinational Standard Cells

Another set of experiments was conducted on combinational standard 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 IX 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 VII. After investigation, it appears 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 5.2.2.

To verify the efficiency of the ML-based CA model generation method when different transistor sizes are considered, the ML model was trained over the 28SOI standard cells and used to predict the behavior of C40 cells. Table X 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 the ML-based characterization methodology could be used to generate CA models for a (large) part of combinational cells of a new technology.

TABLE IX. Average Prediction Accuracy for combinational cells in different technologies

Number of transistors	Predicion accuracy (%) 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 X. Average Prediction Accuracy for combinational cells using different transistor sizes

Number of transistors	Predicion accuracy (%) 2	3	4	5	6
6	100.0	99.8			
8	87.39	99.14	99.03		
9		97.19			
10	92.07	95.49	99.32	98.46	
12	91.71	98.07	99.24	98.47	99.46
14	90.1	95.84	98.63	98.79	99.52
16	91.17	93.59	99.23		99.59
18	88.5	97.15	97.14	97.74	
20	83.87	97.73	97.15	98.94	
22	87.26		98.98	98.44	
24	93.96	99.34	99.58	98.84	99.63
26	87.52	97.55	99.04	99.02	99.92
28	98.19		98.79	99.31	99.44
30			99.13	99.37	99.58
32	92.91			98.92	99.78
42					
44	92.03	98.82			
46		99.23			
47		98.29	99.76		

5.2.2 Analysis and Discussion

A first analysis was done on cells for which the defect characterization methodology gives excellent prediction accuracy as well as those for which the prediction accuracy was quite low. Then, the limitations of the CA model generation method were investigated. After running several experiments on different configurations using one fault model at a time, the following behaviors were noticed:

- 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 simulations**.
- 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 transistors structure or a very similar one. The difference between very similar cells is always the same and is represented in Fig. X.10. More precisely, cells giving good results are always composed of one of the configurations presented in Fig. X.10 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.

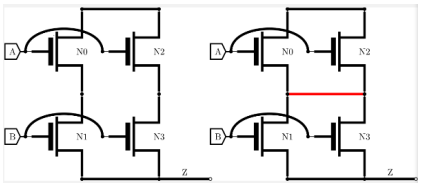


Fig. X.10: Typical transistor configurations leading to good prediction

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

5.2.3 Sequential Standard Cells

We also conducted experiments on sequential cells belonging to two different technologies. As for combinational cells, the ML model was trained over all available sequential standard cells of a given technology and the evaluation was done on one cell of another technology. Cells were grouped according to their number of inputs and transistors. Table XI 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 XII shows the prediction accuracy achieved on short defects.

In the above two scenarios, the average prediction accuracies are globally very low (around 50%), indicating that our method needs good training dataset representative of every type of standard cells and transistors structures. Indeed, investigations showed that in most cases, functionally-equivalent sequential cells in libraries from different technologies were designed differently and hence do not have neither the same transistors structure nor a very similar one (as discussed in 5.2.2). 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 structures, it is not surprising to get such low-quality results. Adding more cells and thus more known structures to the training database should help to increase the prediction accuracy.

TABLE XI. Average Prediction Accuracy for sequential cells different technologies

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				
	50				
	52				

TABLE XII. Average Prediction Accuracy for sequential cells with different transistor sizes

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
	50				
	52				58

5.2.4 Controlled Experiments

In an attempt to check the above hypothesis, “controlled experiments” were performed by considering three scan flip-flops (SDFPQ cells) coming from three different technologies (C40, 28SOI, C28). The SPICE description of each cell was manually modified so as to 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, was made of 32 transistors and can be affected by the same intra-

transistor defects. The physical layouts of the modified cells have not been made identical and carefully modified to the minimum in an attend to keep the technological specificities of each cell. Therefore, the list of potential inter-transistor defect locations is different for each cell. A CA model has been generated for each modified flip-flop, using the simulation-based flow implemented by a commercial tool.

Three types of experiments were performed. First, the ML model was trained by considering all short defects of the C28 SDPFQ cell, and its prediction accuracy was successively evaluated 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, the ML model was trained by considering all short defects of the 28SOI SDPFQ cell, and its prediction accuracy was successively evaluated 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, the ML model was trained by considering all short defects of the C40 SDPFQ cell, and its prediction accuracy was successively evaluated 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 the ML-based characterization method, a confusion matrix was generated 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 cell 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.

TABLE XIII. 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%

Results are reported in Table XIII for the three types of experiments. The confusion matrix can be found at the center of the table (green and red headed columns), this time represented using only a horizontal axis. For example, let us consider the first experiment, when the ML model is trained by considering all defects of the C28 SDPFQ cell, and the prediction accuracy is evaluated over all defects of the 28SOI SDPFQ cell (third row in Table XIII). The number of true positives, false positives, false negatives, and true negatives is 442, 23, 40 and 423

Commenté [AV5]: Juste avant tu dis que les 3 Scan FF sont rendues identiques. Comment se fait-il que le nb de défaut soit différent !!!

Commenté [PDH6R5]: Les layout ne sont pas identiques. Phrases ajoutées dans le paragraphe précédent.

respectively, thus leading to a prediction accuracy of 93%. From the overall results reported in Table XIII, this time it appears that the prediction accuracy achieved with the ML-based 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.

6 Hybrid Flow for CA Model Generation

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

The hybrid flow for accelerating the CA model generation is sketched in Fig. X.11. Typically, when the CA model for a new cell is needed, the first step consists in checking whether the ML-based generation will lead to high-quality CA models. This is done by analyzing the structure of the new cell and check whether the training dataset contains a cell with identical or similar structure (as discussed in Section 5.2.2). If the ML algorithm is expected to give good results, the new cell is prepared (representation in a CA-matrix) and submitted to the trained ML algorithm. The output information is then parsed to the desired file format. Conversely, if the ML algorithm is expected to give poor prediction results, the standard generation flow presented in Fig. X.3 is used to obtain the CA model. A feedback loop uses this new simulated CA model to supplement the training datasets and improve the ML algorithm for further prediction.

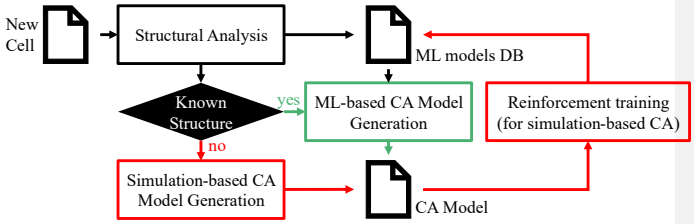


Fig. X.11: Hybrid flow for CA model generation

The experiments performed to estimate the improvement in CA model generation time achieved with the hybrid flow in Fig. X.11 are described in the following.

6.1 Runtime Saving for Combinational Cells

For these experiments, the Random Forest model was first trained on 28SOI combinational standard cells and CA models were then generated for a subgroup of the C40 combinational standard cell libraries. A subgroup is composed of cells representing all the cell functions available in C40 libraries. In these 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 5.2.2) 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, when considering 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, it appears that the ML-based CA model generation works well for about 80% of cells of the C40 subgroup. Surprisingly, the structural analysis revealed that only 50% (205 cells) could be evaluated using the ML-based generation part of the flow. This shows that there is still room for further improvement of the structural analysis in the flow, and hence get better performance of the ML-based CA model generation process.

6.2 Runtime Saving for Sequential Cells

In these experiments, the goal was to (re-)generate CA models for the 27 C40 sequential standard cells in an efficient manner. To achieve this goal, the number of CA models obtained by the simulation-based flow had to be minimized. The lowest achievable number of simulation-based CA models is given by: one cell per training group (same number of inputs and transistors), plus the number of cells that are alone in their training group (and thus cannot go through the ML-based flow). In these experiments, 13 cells had to go through the simulation-based flow (9 groups + 4 individual cells). The simulation-based generation flow for those 13 cells took **~4.1 hours**. The ML-based generation flow for the remaining 14 cells took **~35s**. By comparison, the simulation-based model generation for these 14 cells would take **~4.2 hours**. The ML-based flow thus provides a run-time reduction of **99.8%** for the cells it can handle. Now, if we consider the whole C40 group composed of 27 cells, the hybrid generation flow would require ~4.1 hours + ~35s, to be compared with

~4.1 hours + ~4.2 hours = 8.3 hours by using only the simulation-based generation flow. This represents a reduction in generation time of about **51%**.

It is worth mentioning that most of the run-time in the hybrid-flow is taken by the simulation-based flow. Therefore, as long as new CA models generated by simulation are added to the database, the ML-based flow can use them and then handle more and more cells, further reducing the generation run-time.

7 Discussion and Conclusion

A novel approach based on machine learning was presented in this chapter to generate CA models. The main goal is to speed up the characterization process of standard cell libraries for test and diagnosis purpose, which usually resort to SPICE simulations and hence is very time-consuming. The methodology is based on the recognition of identical structural patterns between cells already characterized by simulation and those to be characterized by using machine learning.

Experiments done on both combinational and sequential cell from industrial libraries demonstrate the accuracy and performance of the method when predicting defect behavior has to be done on the same technology. In this case, the generation run-time of CA models can be significantly reduced for experimented cells having other cells with similar structure in the training dataset.

In order to deal with functionally-equivalent cells having different internal structures, a hybrid flow combining learning-based and simulation-based CA model generation can be used. Experiments carried out on a subset of cells from an industrial library have shown that the generation time of CA models can be reduced by more than 50%.

Experiments reported in this chapter have been carried out on a small size of standard cell population. Considering that more than 10000 cells have usually to be characterized for a given technology, the hybrid flow described in this chapter is expected to provide even better results, especially owing to the reinforcement training that uses simulation generated models for supplementing the training datasets, and hence reduce the number of electrical simulations.

Acknowledgements

References

1. A. Ladhar, M. Masmoudi, and L. Bouzaida, “Efficient and Accurate Method for Intra-Gate Defect Diagnoses in Nanometer Technology,” in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2009.
2. Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Virazel, and E. Auvray, “Effect-Cause Intra-cell Diagnosis at Transistor Level,” in *Proc. IEEE International Symp. on Quality Electronic Design*, 2013.
3. F. Hapke, M. Reese, J. Rivers, A. Over, et al“, "Cell-Aware Production Test Results from a 32-nm Notebook Processor", in *Proc. International Test Conference*, Nov. 2012.
4. Z. Gao, M-C Hu, J. Swenton, S. Magali, J. Huisken, K. Goosens, and E.J. Marinissen, “Optimization of Cell-Aware ATPG Results by Manipulating Library Cells’ Defect Detection Matrices,” in *Proc. IEEE International Test Conference in Asia (ITC-Asia)*, 2019.
5. E. Amyeen, D. Nayak, and S. Venkataraman, “Improving Precision Using Mixed-level Fault Diagnosis,” *in Proc. International Test Conference*, Oct. 2006.
6. H. Tang, A. Jain, and S.K. Pillai, “Using Cell Aware Diagnostic Patterns to Improve Diagnosis Resolution for Cell Internal Defects,” *in Proc. Asian Test Symposium*, pp. 231-236, Nov. 2017.
7. X. Fan, M. Sharma, W.-T. Cheng, and S.M. Reddy, “Diagnosis of Cell Internal Defects with Multi-Cycle Test Patterns”, *in Proc. Asian Test Symposium*, Nov. 2012
8. B. Archer, C. Schuermyer, “Cell-Aware Test for Lower DPPM and Faster Silicon Diagnosis,” in Proc. Synopsys User Group (SNUG), March 2017.
9. N. Feldman, “Accelerating Silicon Diagnosis Using a Cell-Aware Flow,” *in Proc.* Synopsys User Group (SNUG), March 2017
10. F. Hapke, et al., “Cell-Aware Test,” *IEEE Transactions on Computer-Aided Design*, vol. 33, no. 9, pp. 13–6 - 1409, 2014.
11. P. Maxwell, F. Hapke, and H. Tang, “Cell-Aware Diagnosis: Defective Inmates Exposed in their Cells,” in *IEEE European Test Symp.*, 2016.
12. F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, P. Weseloh, M. Wittke, M. Kassab, and C. W Schuermyer, “Cell-Aware Fault Model Creation and Pattern Generation,” US Patent 12/718,799, 2010.
13. S. Mhamdi, P. Girard, A. Virazel, A. Bosio and A. Ladhar, “A Learning-Based Cell-Aware Diagnosis Flow for Industrial Customer Returns,” in *Proc. IEEE International Test Conf.*, 2020.
14. F. Lorenzelli, Z. Gao, J. Swenton, S. Magali, and E.J. Marinissen, “Speeding up Cell-Aware Library Characterization by Preceding Simulation with Structural Analysis,” in *Proc. IEEE European Test Symp.*, 2021.
15. Z. Gao, S. Malagi, M. Chun Hu, J. Swenton, R. Baert, J. Huisken, B. Chehab, K. Goossens, and E.J. Marinissen, “Application of Cell-Aware Test on an Advanced 3nm CMOS Technology Library,” in *Proc. IEEE International Test Conf.*, 2019.
16. R. Guo, B. Archer, K. Chau, and X. Cai, “Efficient Cell-Aware Defect Characterization for Multi-bit Cells”, in *Proc. IEEE International Test Conf. in Asia*, 2018.
17. P. d’Hondt, A. Ladhar, P. Girard, and A. Virazel, “A Learning-Based Methodology for Accelerating Cell-Aware Model Generation,” in *Proc. IEEE/ACM Design Automation and Test in Europe*, 2021.
18. “Nanometer Library Characterization: Challenges and Solutions”, *Webinar*, Silvaco, March 2019.

19. “Improving Library Characterization with Machine Learning”, *White Paper*, Mentor, A Siemens Business, 2018.
20. “Unified Library Characterization Tool Leverages Machine Learning in the Cloud”, *White Paper*, Cadence, 2018.
21. H.S. Poornima and K.S Chethana, “Standard Cell Library Design and Characterization using 45nm technology”, IOSR Journal of VLSI and Signal Processing (IOSR-JVSP) Vol. 4, pp. 29-33, Jan. 2014.
22. J. Rabaey et al., “Digital integrated circuit – A Design Perspective”, Second Edition, Prentice Hall, 2003.
23. Ashral bin Bahari Tambek, Ahmad Raif bin Mohd Noor Beg, Mohd Rais Ahmad, “Standard Cell Library development”, in Proceedings of the 11th International Conference on Microelectronics, 1999, pp.22-24.
24. Neil H.E. Weste, David Harris, Ayan Banerjee, “CMOS VLSI Design: A Circuits and Systems Perspective” Wesley, 1993.
25. Masakazu Shoji, “CMOS Digital Circuit Technology”, Prentice Hall, 1988. ISBN 978-0131388505
26. M. Naga Lavanya and M. Pradeep, “Design and Characterization of an ASIC Standard Cell Library Industry–Academia Chip Collaborative Project,” Microelectronics, Electromagnetics and Telecommunications, 2018.
27. H. Tang et al., “Diagnosing Cell Internal Defects Using Analog Simulation-based Fault Models”, *Proc. Asian Test Symposium*, pp. 318-323, 2014.
28. Dan Clein, “CMOS IC LAYOUT Concepts, Methodologies, and Tools”, ISBN 978-0750671941
29. F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, S. Eichenberger, et al“, "Defect-Oriented Cell-Aware ATPG and Fault Simulation for Industrial Cell Libraries and Desi”ns", *Proc. IEEE International Test Conference*, Nov. 2009.
30. F.M. Goncalves, I.C. Teixeira, and J.P. Teixeira, “Integrated Approach for Circuit and Fault Extraction of VLSI Circuits,” *Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov. 1996.
31. F.M. Goncalves, I.C. Teixeira, and J.P. Teixeira, “Realistic Fault Extraction for High-Quality Design and Test of VLSI Systems,” *in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 1997.
32. Z. Stanojevic, and D.M. Walker, “Fed–x - A Fast Bridging Fault Extractor,” *Proc. of IEEE International Test Conference*, Nov. 2001.
33. S. Venkataraman and S.D. Drummonds, “A Technique for Logic Fault Diagnosis of Interconnect Open Defect”, in Proc. *IEEE VLSI Test Symp*, 2000.
34. C.-M. Li and E.J. McCluskey, “Diagnosis of Resistive-Open and Struck-Open Defects in Digital CMOs ICs”, *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 24, no 11, pp. 1748 – 1759, 2005.
35. F. Pedregosa et al, “Scikit-learn: Machine learning in Python,” Journal of machine learning research, 12(Oct), pp.2825–2830. 2011.