# A Graph-Based Methodology for Speeding up Cell-Aware Model Generation

Gianmarco Mongelli, Xhesila Xhafa, Eric Faehn, Dylan Robins, Patrick Girard, Arnaud Virazel

# A Graph-Based Methodology for Speeding up Cell-Aware Model Generation

G. Mongelli[1,2], X. Xhafa[2], E. Faehn[1], D. Robins[1], P. Girard[2], A. Virazel[2]

[1]STMicroelectronics
Crolles, France
gianmarco.mongelli, eric.faehn, dylan.robins@st.com

[2]LIRMM - University of Montpellier/CNRS
Montpellier, France
gmongelli, xxhafa, girard, virazel@lirmm.fr

***Abstract*—The reduction in transistor size in modern Integrated Circuits (ICs) has led to an increase in manufacturing defects within standard cells, also known as intra-cell defects. Detecting these defects and localizing them through diagnosis is crucial for achieving a fast yield ramp-up and ensuring a low test escape rate. Traditional fault models, such as stuck-at and transition, do not effectively represent intra-cell defects. To address this issue, the Cell-Aware (CA) methodology was introduced. However, this technique involves time-consuming analog SPICE simulations to characterize standard cells. This paper presents a methodology, called Transistor Undetectable Defect eLiminator (TrUnDeL), based on graph theory to speed up the CA model generation process. Our methodology identifies undetectable defects for each stimulus applied at the inputs of the cell, which are then excluded from the analog simulations. We applied TrUnDeL to two libraries from STMicroelectronics (P28 and C28) to identify the undetectable stimulus/defect pairs, so that analog simulations are performed only on the remaining pairs. As a result, the CA model generation process time is reduced by a factor 3. Finally, we applied TrUnDeL to a SRAM bitcell case study and demonstrated that the obtained results are consistent with its existing CA model.**

***Index Terms*—Intra-cell defects, test and diagnosis, graph theory, cell-aware model.**

## I. Introduction

In modern industrial technologies, the number of gates in Integrated Circuit (ICs) is increasing as transistor feature sizes get smaller and smaller [1]. This leads to an increasing number of manufacturing defects (i.e., opens, shorts) inside standard cells (i.e., intra-cell defects). Finding the correct test patterns to detect these defects in ICs and then performing diagnosis to localize them is a key issue to achieve a fast yield ramp-up and ensure a low Defective Parts Per Million (DPPM).

To generate the smallest and most effective set of test patterns, Automatic Test Pattern Generation (ATPG) algorithms have been developed. For this purpose, ATPG algorithms use fault models to represent the behavior of the defect and its effect on ICs [2]. However, traditional fault models, such as stuck-at or transition fault models, model defects at the interconnections between cells (i.e., inter-cell defects), so intra-cell defects can only be detected fortuitously [3].

Cell-Aware (CA) methodology has been introduced to target intra-cell defects. The CA methodology characterizes a standard cell by building a dictionary (i.e., a defect-detection matrix) that contains the information about which defects are detectable by which stimuli [4], [5]. The CA characterization is usually performed using analog SPICE simulators, applying each possible stimulus at the cell inputs for each defect injected (i.e., pure shorts and pure opens) in the design netlist and evaluating the cell outputs. The problem of this approach is that analog SPICE simulations are time-consuming operations and the characterization of a full CA library can take up to several months [6].

To accelerate the adoption of the CA methodology in the qualification process of silicon products and its deployment on industrial ICs [7], it is essential to minimize the time and cost required for library characterizations. A machine learning-based methodology, proposed in [8], can predict whether a defect is detectable on the outputs of a cell, without the need for analog simulations. However, its drawback is that accurate predictions can only be made for cell structures that have already been evaluated in the past, while analog simulation is necessary for other cells. Another approach, proposed in [9], uses the graph theory to model standard cells (from Cadence's GPDK045 library) and reduce the computation time of CA models. Although effective (i.e., CA characterization time-cost is improved by a factor of 2), in this methodology, defects are injected one by one for each stimulus, making it not suitable for cells with large number of transistors. Moreover, the paper only explains the effects of source and drain defects while defects related to bulk and gate are not mentioned.

Therefore, the goal of our work is to develop a methodology (TrUnDeL), able to further accelerate the CA model generation process while preventing a loss of accuracy, targeting the complete set of static and dynamic defects. The list of targeted defects include open and short defects that affect source, drain, bulk and gate of the transistors. TrUnDeL is inspired by [9] and it is based on graph-theory. In the proposed solution, generic standard cells are modeled as a graph. Then, for each stimulus applied to the graph and propagated through the different nodes, additional metadata are attached to the graph. These metadata are then exploited to identify the Undetectable defects, for each stimulus previously applied, through two techniques used in sequence: *Transistor Rules*

and *Defect Injection and Propagation*. Thanks to Transistor Rules, TrUnDeL can manage several defects at a time and not only one by one as in [9].

First experiments performed on two different libraries, P28 and C28 by STMicroelectronics, show that applying our methodology before the analog simulations reduces the computation time for CA characterization by a factor of 3. For validation purpose, the obtained results are compared with existing CA models generated using analog SPICE simulations. The main outcome using TrUnDeL is that no misclassifications occur. In other words, a detectable stimulus/defect pair in the CA model is never classified as undetectable by TrUnDeL. Moreover, TrUnDeL has been applied on a SRAM bitcell case study. The obtained results are consistent with the available CA model [10].

This paper is organized as follows. Section 2 describes basics and backgrounds on CA methodology and on graph theory applications for defect-detection or circuit modelling purposes. Section 3 presents the TrUnDeL graph-based methodology. Validation results are presented in section 4 and finally our conclusions and perspectives are given in section 5.

## II. Related Prior Work

The CA methodology involves characterizing a standard cell by creating a dictionary, which is a matrix known as the Defect-Detection Matrix (DDM) [4]. To build the DDM for a cell, one analog SPICE simulation is performed for each stimulus $s$ applied to the cell inputs and each defect $df$ (pure shorts and pure opens) injected into the cell design netlist. After conducting $s * df$ analog SPICE simulations, the characterization is complete, and a status, that can be Detectable (D) or UnDetectable (UD), is assigned to each stimulus/defect $(s, df)$ pair. UD means that the cell outputs are not affected by the defect injection in the design netlist under a given stimulus. Respectively, D means that output is affected.

The DDM has rows that represent stimuli and columns that represent defects. For each $(s, df)$ pair, if the corresponding entry in the matrix is greater than or equal to 1, it indicates that this pair is D. For instance, if a $(s, df)$ pair only affects the first output, the corresponding entry in the matrix is 1. If it affects the second output, the entry in the matrix is 2. If it affects both outputs, the entry in the matrix is 3. Conversely, if the entry is 0, it means that the pair is UD. An example of a DDM for a generic two output cell is shown in Table I.

TABLE I
EXAMPLE DEFECT-DETECTION MATRIX.

| | $df_0$ | $df_1$ | $df_2$ | $df_3$ | ... | $df_{j-1}$ |
|---|---|---|---|---|---|---|
| $s_0$ | 0 | 3 | 3 | 0 | | 0 |
| $s_1$ | 1 | 2 | 0 | 0 | | 0 |
| ... | 0 | 0 | 0 | 2 | | 0 |
| $s_{i-1}$ | 0 | 0 | 0 | 0 | | 3 |

Characterizing each library cell using the CA methodology is a time-consuming process, as it requires conducting $s * df$ analog SPICE simulations. Therefore, the time required to characterize each cell depends on its number of inputs and transistors. The methodology proposed in [9] aims to reduce CA characterization time-cost by performing structural analysis, based on graph theory, of each cell of a library. Graph theory has found wide application in various fields, such as logic simulation [11], [12] , timing analysis [13] and defect-detection purposes [14]. This demonstrates its high flexibility and effectiveness.

In the methodology presented in [9], the authors divide the cells into compartments (i.e., communicating components in switch-level algorithms [12]). Compartments are substructures of cells that are characterized by an output net that is connected to VDD through a PMOS pull-up network and to GND through an NMOS pull-down network. A compartment contains transistors that have their gates controlled by external compartment nets and contains nets that control external compartment transistors. It does not contain nets that control internal compartment transistors. An example of NAND cell divided in compartments is shown in Fig. 1.

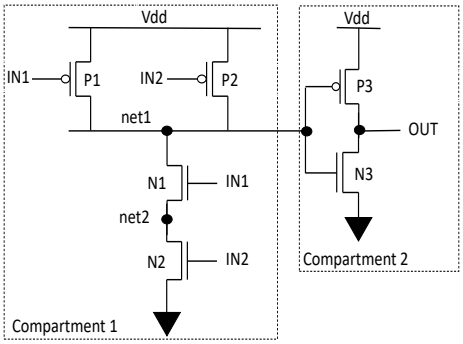


Fig. 1. Nand gate divided in two compartments.

Once compartments are identified, a graph is built for each cell and for each pattern applied. Edges represent the active transistors while nodes represent nets. If a short is injected, an edge is added. If an open is injected, an edge is removed. Finally, a $(s, df)$ pair is D if a path between VDD and GND exists for a short defect, or if an output is floating for an open defect. The solution is designed to detect both static defects, which are identified by one-cycle stimuli, and dynamic defects, which are identified by two-cycle stimuli. However, the solution only considers open defects for dynamic detection, which may not be entirely accurate as some shorts can also be detected dynamically in CA models. Thanks to this methodology, the generation time of CA models has been reduced by a factor of 2. However, the CA models generation process time can be further reduced, as the methodology only targets defects that affect the source and drain. The bulk and gate are not considered at all. If the bulk and gate were also taken into account, more $(s, df)$ pairs could be identified as UD, so more pairs could be excluded from time-consuming analog simulations. Additionally, the run-time of the methodology could be an issue, as only one defect is analyzed at a time. This is suitable for small cells with a small

number of input pins and transistors, but it could affect the efficiency of the methodology for larger cells.

## III. Transistor Undetectable Defect eLiminator (TrUnDeL)

The proposed graph-based methodology, called TrUnDeL, is based on the three-step flow shown in Fig. 2. First, the graph is computed based on the cell. Then, we apply a stimulus to the graph and propagate it through the different nodes. In the last step, detection techniques are applied to identify the U defects. These three steps are presented in detail in the next subsections.

Once TrUnDeL is applied to a cell, a set of $(s, d)$ pairs is classified as UD and will not require any analog SPICE simulations. Only the remaining pairs, classified as PD, will be simulated, reducing the time-cost for CA model generation process. A $(s, d)$ pair, classified as PD, implies that the cell outputs may be affected by the injection of a given defect under a given stimulus. After the analog simulation, all the PD $(s, d)$ tuples will be classified as U or D.

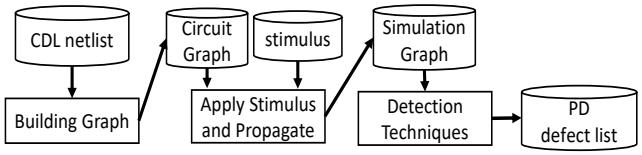


Fig. 2. TrUnDeL flow.

### A. Building Graph

The input of this step is the Circuit Design Language (CDL) netlist that describes the cell with all the information about transistors and their connections. The Circuit Graph (CG) is built by extracting the information about the nets and the transistors from this file. CG is composed of two types of nodes: i) the devices (i.e. transistors), and ii) the nets. The edges represent the connections between devices and nets. CG of the cell in Fig. 1 is shown in Fig. 3. As we can see, the transistor node P1 is connected to three net nodes (VDD, IN1, net1) through three edges (source, drain, gate). Bulk is not modeled in this example for the sake of readability.

Once the CG is built, we identify the different compartments (i.e, sub-graphs of CG) that compose it. In a compartment, we can distinguish four types of net nodes:

- internal nets: nets that are connected to transistor devices through only their sources and drains;
- input nets: nets that are connected to at least one transistor of the same compartment through its gate;
- output nets: nets that are connected to at least one transistor of another compartment through its gate;
- in/out nets: nets that are both input and output of the same compartment.

The in/out nets are required to manage loop cell structures and they are an extension to the definition of compartment given in [9], where only internal, input and output nets were defined. All additional information is attached to the graph as metadata. The division in compartments of CG is highlighted in Fig. 3.

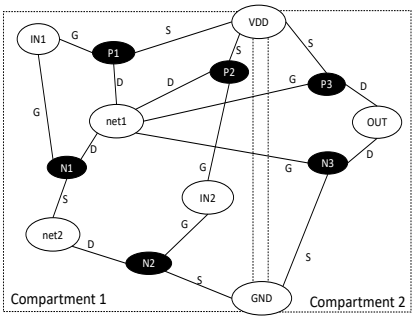


Fig. 3. CG corresponding to a nand cell.

### B. Apply Stimulus and Propagate

In this step, we apply a stimulus to CG. A stimulus is composed of one cycle in static stimuli and two cycles in dynamic stimuli. Each stimulus cycle is a set of logical ones or zeros assigned to the input nets where '1' maps to VDD and '0' maps to GND. Once the stimulus cycle is applied to the graph, it is propagated through all the nodes. Propagation starts from the devices connected to the input nets and it is performed following different propagation rules. Then, net to device and device to net propagation continues until all the nodes are covered. Then, the process starts again for the next stimulus cycle (i.e., for dynamic stimuli). At the end of the stimulus application and propagation process, every net node has a vector of logical values associated, where the logical value is '0', '1' or 'None' ('None' means that that node is isolated, i.e., connected to neither VDD nor GND). Every transistor nodes has a vector of states, where state is 'Active' (A) or 'Inactive' (I), based on the value connected to the gate. Each entry in the vector logical values and vector states corresponds to a stimulus cycle. At the end of this step, a new graph is obtained: the Simulation Graph (SG). An example of SG, with a given static stimulus applied (IN1 = '1', IN2 = '0') to CG of Fig. 3, is shown in Fig. 4.

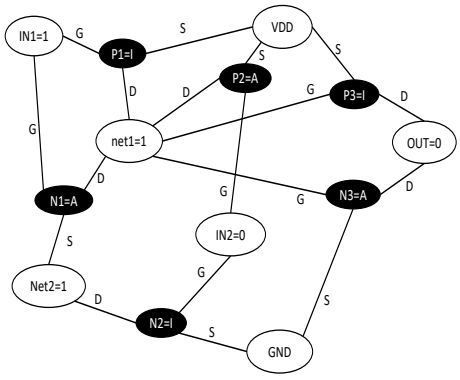


Fig. 4. SG of nand, with stimulus: IN1 = '1' and IN2 = '0'.

### C. Detection Techniques

The last step of the flow in Fig. 2 is to identify the total list of defects that can affect the cell before excluding those that we are unable to detect for the given stimulus. We start by computing the list of all defects that are possible in the cell. This initial list is composed of all shorts and opens that can affect the different transistors. The shorts (Sh) are made up of all possible combinations of Source (S), Drain (D), Bulk (B) and Gate (G) for each transistor as shown in Fig. 5. The Opens (O) are OS, OD and OG. As opens can only be detected dynamically [15], static analysis considers only the shorts while dynamic analysis considers shorts and opens.

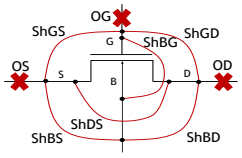


Fig. 5. Targeted defects for a transistor.

We then apply two detection techniques to exclude UD defects from this initial list: *Transistor Detection Rules* and the *Defect Injection and Propagation*. During static analysis, a one-cycle stimulus is applied. A defect is classified as UD for a given stimulus, if one of the two techniques identifies it as UD, for the unique cycle applied. During dynamic analysis, a defect is classified as UD for a given stimulus, if the defect is UD by at least one of the two detection techniques for both cycles of the two-cycle stimulus applied. By doing so, we construct the PD defect list as shown in Fig. 6.

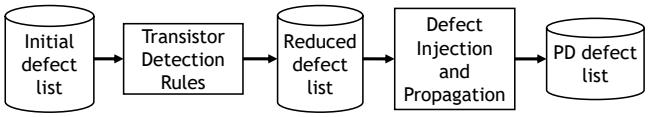


Fig. 6. Detection Techniques flow.

*1) Transistor Detection Rules:* The input of this technique is the initial defect list for a given SG. The rules are applied to all the defects affecting each transistor in one shot, proceeding with one transistor at a time. In order to do so, no further action is needed on the SG and the rules directly use the available transistor metadata contained in SG. The majority of UD defects are identified and removed from the initial defect list. A first reduced defect list can therefore be produced. Some examples of transistor level rules are shown in Fig. 7. In example *a*, since the transistor is active (i.e., G=1) the ShSD has no impact. In example *b*, since transistor is inactive (i.e., G=0), OS and OD have no impact. In the last example *c*, if G is equal to B, ShBG has no impact.

*2) Defect Injection and Propagation:* The input to this technique is the reduced list of defects obtained after transistor detection rules. The idea is to inject each defect of the reduced list into the graph one at a time, and then propagate its effects throughout the graph. If the output node is not affected by this propagation, then the defect is classified as UD. Once all the defects are injected and propagated, all the identified UD defects can be removed from the list and the final PD defect list is obtained.

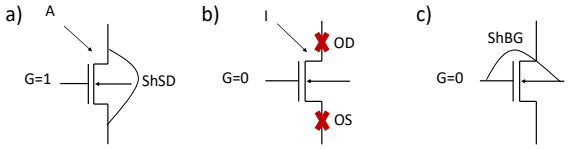


Fig. 7. Examples Transistor Detection Rules.

To inject a defect into the graph, we modify it by adding a Resistor Device Node (RDN). Then, if there is a conflict (i.e., two opposite logic values) between the two nets connected through the resistor, the propagation is performed. It is performed compartment by compartment, starting from the compartments connected to RDN. A decision tree can be applied to the current compartment of SG to verify if the injected defect affects the current compartment or not:

- if a path connecting VDD to GND and containing only active transistors exists, the defect affects the compartment;
- if the output has a path with only VDD or only GND in which all the transistor device nodes are in state 'A', the output is compared with the fault free one. If they are equal the defect does not affect the compartment.
- if the output has not a path in which all the transistor device nodes are in state 'A' with both VDD and GND, the defect affects the compartment.

If the current compartment is not affected, it means that its output does not change with respect to the fault free propagation, therefore the defect is classified as UD for the cell and the propagation is stopped. Otherwise, the defect is propagated to the next compartment where the output of the current compartment becomes the input of the next one and so on. Propagation continues until one of two conditions occur: either we reach the output of the last compartment of the cell, or the defect becomes UD in one of the compartments along the propagation path.

## IV. Validation Results

For validation purpose, TrUnDeL was applied to two libraries, P28 and C28, designed by STMicroelectronics and to a SRAM bitcell case study. Next subsections detail the different experiments performed.

### A. P28 and C28 standard cell libraries

In experimental results on P28, TrUnDeL was applied to 932 combinational cells, resulting in 110,092 defects. By the end of the analysis, 3,669,967 $(s, df)$ pairs were processed (1,215,792 static and 2,454,175 dynamic). Figure 8 highlights the comparison between classical CA models (i.e., based on analog SPICE simulations) and TrUnDeL. An important result is that no misclassifications are generated by TrUnDeL (i.e, D pair classified as UD). TrUnDeL is able to identify 66%

of UD $(s, d)$ pairs. The remaining 34% are classified as PD. These ambiguous cases are unable to be resolved by TrUnDeL because a conflict arises in the cell and, when propagated, it causes an unknown value at the cell output. Having an unknown value on the cell output means that establishing if these pairs are UD is impossible, so analog simulation is needed to determine their status. This explains the gap (23%) that can be observed when comparing the obtained results with those achieved with classical CA models.

Focusing on the $(s, d)$ pairs classified as UD (66%), 44.1% were identified only by using the *Transistor Detection Rules* technique, while the remaining 21.9% were identified by the *Defect Injection and Propagation* technique. Using both the detection techniques in sequence, TrUnDeL took about 5h, using 1 CPU, to analyze the 932 cells of P28. The first detection technique allows us to analyze the tuples more efficiently because it analyzes all the defects of a transistor simultaneously, while the second one analyzes each defect one at a time. In fact, the same 66% of UD $(s, d)$ pairs could be reached by using only the *Defect Injection and Propagation* technique, but this would be 30% slower in terms of CPU time.

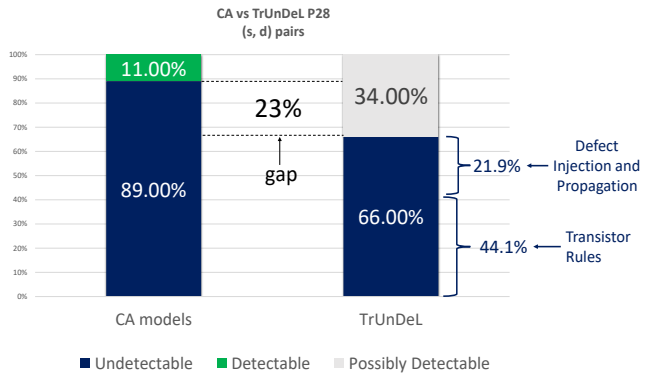


Fig. 8. Classical CA models vs TrUnDeL for P28 library cells.

Figure 9 shows the comparison between TrUnDeL and classical CA models for the C28 library. Here, 536 combinational cells are analyzed, resulting in 119,088 defects and 4,545,684 $(s, df)$ pairs (1,351,920 static and 3,193,764 dynamic). Also in this case, no misclassification pairs are generated and 77.6% of UD $(s, df)$ pairs are identified by TrUnDeL with a gap of 14.8%.

The smaller gap and the greater number of UD $(s, df)$ pairs in C28 than P28 is explained by two reasons:

- In C28, cells are bigger than in P28. So, more $(s, df)$ pairs cannot propagate their effects in the cells with more transistors, resulting to be UD at the end of the TrUnDeL process.
- In C28, the bulk of every PMOS transistors is connected to VDD. As a result, all ShBS defects of PMOS transistors which the source is connected to VDD are classified as UD, regardless of the stimulus applied. In P28, this is not the case as bulk of every PMOS transistors is connected to GND. The bulk of NMOS transistors are connected to GND in both P28 and C28.

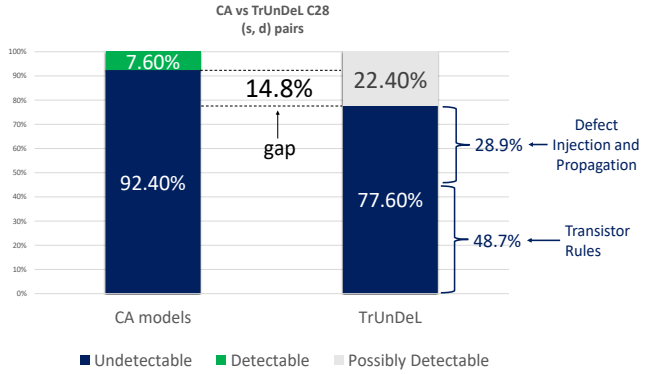


Fig. 9. Classical CA models vs TrUnDeL for C28 library cells.

TrUnDeL, using *Transistor Detection Rules* and *Defect Injection and Propagation* detection techniques took about 10h to analyze the 536 cells of C28 library, using 1 CPU. Using only the second detection technique would be 30% slower also in this case.

As an example to estimate the improvement, in CA model generation time achieved with TrUnDeL, we can refer to the experiments conducted on P28. Initially, we performed the classical CA model generation based on analog SPICE simulations on the 932 cells of P28, which took approximately 92 days and 5 hours using a single SPICE license on 1 CPU. Next, we run TrUnDeL on the same cells of P28, and the CPU time (using 1 CPU) was about 5 hours. TrUnDeL identified 66% of UD $(s, d)$ pairs that were not analog simulated. Then, analog SPICE simulations on the remaining 34% of PD pairs (using 1 SPICE license), took approximately 33 days and 12 hours. Therefore, TrUnDeL + analog simulations on PD pairs took around 1 month, resulting in a reduction in time by a factor of 3 compared to the classical CA model generation.

### B. SRAM bitcell case study

Memories are critical components in modern System-on-Chips (SoCs). Considering the higher incidence of manufacturing defects in shorter node technologies, the testing and diagnosis of these circuits becomes crucial for the overall quality of SoCs [16]. A preliminary work that uses the CA methodology to target these defects in SRAM bit-cells is presented in [10]. Here after, we evaluate TrUnDeL on the 1-bitcell SRAM case study to verify whether TrUnDeL's output is consistent with the bitcell CA model.

The bitcell with its defects is shown in Fig. 10. These defects (i.e., df1-df7) are sensitized during write operations. Consequently, BL, BLB and WL nets are considered to be inputs, while S and SB nets are considered to be outputs in the CA model of the bitcell. Table II shows CA model of the bitcell obtained with analog SPICE simulations. In order to generate the necessary stimuli to detect the considered defects, a single SRAM bitcell surrounded by the precharge circuit, the

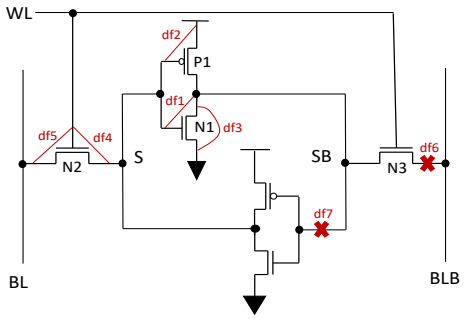


Fig. 10. SRAM bitcell with defects analyzed.

TABLE II
CA MODEL SRAM BITCELL WITH BITCELL STIMULI

| BL | BLB | WL | S | SB | df1 | df2 | df3 | df4 | df5 | df6 | df7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | D | D | D | UD | UD | UD | UD |
| 1 | 0 | 1 | 1 | 0 | D | UD | UD | D | UD | UD | UD |
| R | F | 1 | R | F | D | D | D | D | UD | D | D |
| F | R | 1 | F | R | D | D | D | D | D | UD | UD |

sense amplifier and the write driver circuit was considered. The input signals of the circuit are DATA, WR_EN (WRite ENable), WL (Word Line), SA_EN (Sense Amplifier ENable) and Pch_EN (Precharge ENable) while the output signals are S, SB and SA_O (Sense Amplifier Output). The stimuli able to detect the seven defects in Fig. 10 were generated using an Automatic Test Pattern Generator (ATPG), and they are shown in Table III.

TABLE III
ATPG MEMORY STIMULI TO DETECT THE CONSIDERED DEFECTS.

| Stimuli | Data | WR_EN | WL | SA_EN | Pch_EN | S | SB | SA_O |
|---|---|---|---|---|---|---|---|---|
| stimulus1 | 0 | 1 | 1 | X | 1 | 0 | 1 | X |
| stimulus2 | 1 | 1 | 1 | X | 1 | 1 | 0 | X |
| stimulus3 | R | 1 | 1 | X | 1 | R | F | X |
| stimulus4 | F | 1 | 1 | X | 1 | F | R | X |

The purpose of this validation case study is to use the stimuli obtained by the ATPG to identify the UD defects using TrUnDeL on the SRAM bitcell. Then, TrUnDeL's output, shown in Table IV, is compared with the bitcell CA model, shown in Table II.

TABLE IV
TRUNDEL'S OUTPUT ON A BITCELL.

| Stimuli | df1 | df2 | df3 | df4 | df5 | df6 | df7 |
|---|---|---|---|---|---|---|---|
| stimulus1 | PD | PD | PD | PD | PD | UD | UD |
| stimulus2 | PD | UD | UD | PD | UD | UD | UD |
| stimulus3 | PD | PD | PD | PD | PD | PD | PD |
| stimulus4 | PD | PD | PD | PD | PD | PD | PD |

TrUnDeL is consistent with bitcell CA model, as no misclassification occurs, like in standard cell cases. Additionally, TrUnDeL is able to identify 25% of UD $(s, d)$ pairs with a gap of 5 pairs (i.e., pairs highlighted in gray in Table IV). These experiments demonstrates that the proposed methodology is suitable not only for standard cell libraries but also for SRAM memories in generating effective CA models.

## V. CONCLUSIONS AND PERSPECTIVES

A new methodology, called TrUnDeL, has been developed to reduce the CA model generation time for standard cell libraries. This methodology, based on graph theory, is able to generate effective CA models without any misclassifications compared to classical CA models generated by analog SPICE simulations, on combinational cells. Moreover, we have experienced TrUnDeL on a SRAM bitcell case study. Compared to existing CA models, our methodology is able to efficiently generated CA models that will be helpful for SRAM intra-cell test and diagnosis purposes. Next step will be to apply and adapt TrUnDeL to sequential cells. Further improvements will consist in reducing the gap by incorporating more technological information (e.g., transistor strength, timing, defect size) as metadata in the graph.

## REFERENCES

[1] M. G. Bardon et al., "Extreme scaling enabled by 5 tracks cells: Holistic design-device co-optimization for FinFETs and lateral nanowires," 2016 IEEE International Electron Devices Meeting, San Francisco, CA, USA, pp. 28.2.1-28.2.4.
[2] Kyoung Youn Cho et al., "Gate exhaustive testing," IEEE International Conference on Test, 2005., Austin, TX, USA, 2005, pp. 7 pp.-777.
[3] S. Eichenberger et al., "Towards a World Without Test Escapes: The Use of Volume Diagnosis to Improve Test Quality," 2008 IEEE International Test Conference, Santa Clara, CA, USA, 2008, pp. 1-10.
[4] Z. Gao et al., "Defect-Location Identification for Cell-Aware Test," 2019 IEEE Latin American Test Symposium (LATS), Santiago, Chile, 2019, pp. 1-6.
[5] P. Maxwell et al., "Cell-aware diagnosis: Defective inmates exposed in their cells," 2016 21th IEEE European Test Symposium, Amsterdam, Netherlands, pp. 1-6.
[6] Z. Gao et al., "Application of Cell-Aware Test on an Advanced 3nm CMOS Technology Library," 2019 IEEE International Test Conference (ITC), Washington, DC, USA, 2019, pp. 1-6.
[7] R. Guo et al., "Efficient Cell-Aware Defect Characterization for Multi-bit Cells," 2018 IEEE International Test Conference in Asia (ITC-Asia), Harbin, China, 2018, pp. 7-12.
[8] P. d'Hondt et al., "A Learning-Based Methodology for Accelerating Cell-Aware Model Generation," 2021 Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, pp. 1580-1585.
[9] F. Lorenzelli et al., "Speeding up Cell-Aware Library Characterization by Preceding Simulation with Structural Analysis," 2021 IEEE European Test Symposium (ETS), Bruges, Belgium, 2021, pp. 1-6.
[10] X. Xhafa et al., "On Using Cell-Aware Methodology for SRAM Bit Cell Testing," 2023 IEEE European Test Symposium, Venezia, Italy, pp. 1-4.
[11] J. P. Hayes, "An Introduction to Switch-Level Modeling," in IEEE Design & Test of Computers, vol. 4, no. 4, pp. 18-25, Aug. 1987.
[12] R. E. Bryant, "A Survey of Switch-Level Algorithms," in IEEE Design & Test of Computers, vol. 4, no. 4, pp. 26-40, Aug. 1987.
[13] A. Rjoub and A. B. Alajlouni, "Graph modeling for Static Timing Analysis at transistor level in nano-scale CMOS circuits," 2012 16th IEEE Mediterranean Electrotechnical Conference, Yasmine Hammamet, Tunisia, 2012, pp. 80-83.
[14] Kuang-Wei Chiang and Z. G. Vranesic, "On Fault Detection in CMOS Logic Networks," 20th Design Automation Conference Proceedings, Miami Beach, FL, USA, 1983, pp. 50-56.
[15] James Chien-Mo Li and E. J. McCluskey, "Diagnosis of resistive-open and stuck-open defects in digital CMOS ICs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 11, pp. 1748-1759, Nov. 2005.
[16] S. Borkar et al., "Microarchitecture and design challenges for gigascale integration," in MICRO, vol. 37, pp. 3–3, 2004