Accelerating Cell-Aware Model Generation for Sequential Cells using Graph Theory

Gianmarco Mongelli [1,2] Eric Faehn [1] Dylan Robins [1] Patrick Girard [2] Arnaud Virazel [2]

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

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

***Abstract*— The Cell-Aware (CA) methodology has become essential to detect and diagnose manufacturing intra-cell defects in modern semiconductor technologies. It characterizes standard cells by creating a defect-detection matrix, which serves as a reference that maps stimuli to the specific defects they can detect. Its limitation is that CA approach needs a number of time-consuming analog simulations to create the matrix. In [1] a graph-based methodology able to reduce the number of simulations to perform, called Transistor Undetectable Defect eLiminator (TrUnDeL), was presented. TrUnDeL can identify undetectable stimulus/defect pairs that are then excluded from the analog simulations. However, its use is limited to combinational cells and does not offer any guidance on handling sequential cells, which are usually the most complex cells. In this paper we present a new version of TrUnDeL that supports sequential cells analysis. Experiments conducted on sequential cells from two standard cell industrial libraries demonstrate that the CA generation time is reduced by 30% without compromising accuracy.**

Keywords—graph theory, cell-aware model, intra-cell defects, sequential cells, test and diagnosis

I. Introduction

In the latest semiconductor technologies, the Cell-Aware (CA) methodology has become increasingly important to achieve an adequate Defective Part Per Million (DPPM) rate and a fast yield ramp-up. CA is used to detect and to localize intra-cell manufacturing defects (i.e., defects inside standard cells) during test and diagnosis. This is because traditional fault models, such as stuck-at or transition fault models, generally represent defects occurring at the interconnections between cells (i.e., inter-cell defects) [2] and intra-cell defects are only detected fortuitously [3].

CA involves characterizing standard cells by creating a defect-detection matrix, which serves as a reference by mapping specific defects to the stimuli that can detect them [4]-[6]. Characterization typically uses analog SPICE simulators to apply all possible input stimuli to the cell's design netlist, injecting every potential intra-cell defect to observe its impact on the circuit functionality. The major drawback of this approach is the significant time required: analog SPICE simulations are slow, and fully characterizing a CA library can take several months [7].

To speed up the integration of the Cell-Aware (CA) approach in the qualification process of silicon products and its application to industrial ICs, it is crucial to minimize the time and cost associated with library characterizations [8]. To address this issue, a graph-based methodology called Transistor Undetectable Defect Eliminator (TrUnDeL) was presented in [1]. TrUnDeL is a flow characterized by three steps: *Building Graph*, *Apply Stimulus and Propagate* and *Detection Techniques.* It identifies undetectable defects for each stimulus without the need of analog simulations. As a result, the number of analog simulations is reduced, thereby decreasing the time required for library characterization. The limitation of TrUnDeL is that it only addresses combinational cells, as it fails when cells are sequential, resulting in some misclassifications appear (i.e., a detectable stimulus/defect pair in the CA model is never classified as undetectable by TrUnDeL).

The main difference between combinational and sequential cells is that sequential cells have feedback-loop structures, which require specific management. Additionally, sequential cells generally have a larger structure than combinational cells, so it is necessary to improve the different steps of TrUnDeL to guarantee an effective defect classification while analyzing sequential cells. In this paper, we present a new version of the TrUnDeL methodology that can analyze sequential cells addressing the issues explained above in addition to combinational cells.

We applied, in sequence, the new version of TrUnDeL and the analog simulations on the sequential cells of two industrial libraries. To validate the methodology, we compared the results obtained using improved TrUnDeL with the pre-existing CA models. The achieved experimental results indicate a cell-aware generation time reduced by 30% with no misclassifications. It means that for the analyzed sequential cells the accuracy is 100% as we do not have any misclassification.

This paper is organized as follows. Section II describes related prior work on accelerating CA model generation. Section III presents the initial TrUnDeL flow. Section IV describes the new techniques and modifications introduced in the new version of TrUnDeL. Section V details the different experimental results obtained. Section VI concludes the paper and discusses future perspectives.

II. Related Prior Work

The CA approach is used to characterize standard cells [4]. During the characterization process, the methodology creates a dictionary called Defect-Detection Matrix (DDM). The

DDM summarizes the detectability status for each defect *df* and each stimulus *s*. The detectability status can be Detectable (D) or UnDetectable (UD). D means that injecting the defect for the applied stimulus affects at least one output of the cell. On the other hand, UD means that none outputs are affected. The stimuli can be static (i.e., one-cycle patterns) or dynamic (i.e., two-cycle patterns). To build the DDM, the methodology performs a number of analog simulations equal to the product *s * df*. This is because a simulation is performed for each *(s, df)* pair. A DDM example of a 2-output cell is shown in TABLE I. In the matrix, each entry corresponds to an *(s, df)* pair. A value number is associated with each entry of the DDM. The value can be 0 if the *(s, df)* pair is UD, 1 if the *(s, df)* pair is D to the first output, 2 if the *(s, df)* pair is D to the second output, and 3 if the *(s, df)* pair is D to both outputs. Building the DDM is a time-consuming operation as it requires conducting a high number of analog SPICE simulations. The time it takes to characterize each cell is influenced by its transistors structure.

TABLE I. EXAMPLE DDM.

	df_0	df_1	df_2	df_3		df_4	…		df_{j-1}
s_0	0	0	1	0		0			0
s_1	1	0	2	1		0			1
…	0	3	0	1		2			1
s_{i-1}	0	0	0	3		0			0

To accelerate the integration of the CA approach in silicon product qualification and industrial IC applications [8], reducing the time-cost of library characterizations is crucial. A machine learning-based methodology proposed in [9] predicts defect detectability at cell's outputs without additional analog simulations. This uses a prediction model trained on existing CA models from past exhaustive simulations. However, this method only speeds up the process for previously analyzed cell structures. Cells with new structures still require traditional analog simulations.

Another methodology uses structural analysis to reduce the number of analog simulations to perform [10]. The standard cells are divided into compartments, each defined by an output net linked to VDD via a PMOS pull-up network and to GND through an NMOS pull-down network. An example of compartments is shown in Fig. 1.

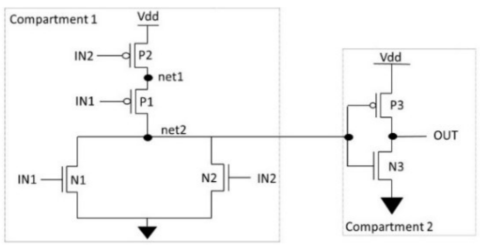

Fig. 1. OR gate divided into two compartments.

A graph is constructed for each cell and each stimulus applied, with active transistors as edges and nets as nodes. Then, the defect is injected in the circuit. Short circuits add edges to the graph while open circuits remove them. Defects are detected when a short creates a path between VDD and GND or when an open results in a floating output if their effect is propagated compartment by compartment until the output. This approach reduces the time required to generate CA models. However, further reductions are possible by addressing defects in the bulk and gate regions, which could classify more *(s, df)* pairs as UD and avoid time-consuming analog simulations. The methodology's runtime is also a challenge, as it processes one defect at a time. While this is manageable for small cells with few inputs and transistors, it may reduce efficiency for larger cells as the sequential ones.

The systematic approach, proposed in [11], uses the voltage information generated by fault-free analog simulations to identify most UD (*s, df*) pairs. For instance, if a short circuit causes two nets to have the same voltage value in a fault-free simulation, the pair is classified as UD. For all the UD pairs identified using this approach, the faulty analog simulations are not needed, resulting in a reduced number of analog simulations. For the remaining pairs, the analog faulty simulation is still required and performed as usual. However, the methodology still relies on analog simulations for the fault-free case and all the faulty cases that can still be D, which can be time-consuming for large circuits.

III. Initial TrUnDeL flow [1]

TrUnDeL is a graph-based methodology for defect detection in cells, consisting of three steps as shown in Fig. 2. The first step creates a graph from the cell netlist. The second step applies a stimulus and propagates its consequent effects throughout the graph. The last step identifies the UD *(s, df)* pairs that will be excluded from the analog simulations. At the end, only the remaining pairs, classified as Possibly Detectable (PD), will be simulated and further classified as UD or D, resulting in a reduction of the overall number of analog simulations. A PD pair indicates an output that may or may not be affected by the injection of a defect.

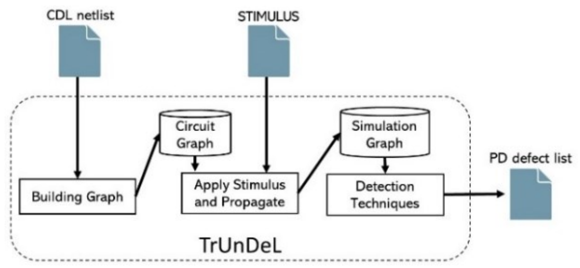

Fig. 2. TrUnDeL flow.

A. Building Graph

The Circuit Graph (CG) is constructed by extracting details from the CDL netlist, including nets and transistors. CG comprises two types of nodes: devices (such as transistors) and nets, with edges representing the interconnections between them. Metadata is attached to the edges to describe the connection types and to the transistor nodes to specify their type (Nmos or Pmos). An example representing the OR gate in Fig. 1 translated into a graph is shown in Fig. 3.

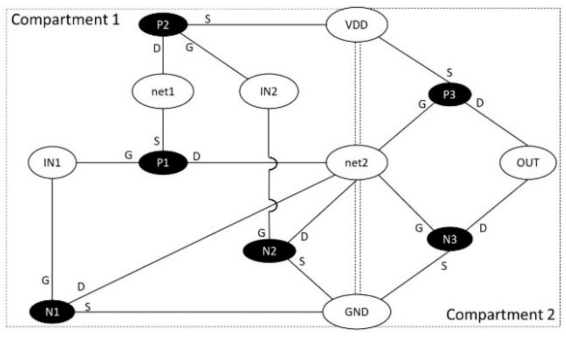

Fig. 3. OR Circuit Graph.

The bulk is omitted for the sake of clarity. After the graph extraction, CG is divided into sub-units called compartments that will be exploited later in detection techniques.

B. *Apply Stimulus and Propagate*

This step operates as a classical event-driven switch-level simulator using an iterative method [12], [13]. It begins by applying a stimulus to the input nets of the CG. A stimulus can be a single cycle for static stimuli or a pair of cycles for dynamic stimuli, with '1' representing VDD and '0' representing GND. Events are scheduled for each input net, and propagation occurs from net to device and vice versa. Each time a net changes its voltage value, a new event is scheduled. The propagation continues until no more events are scheduled. After propagation, each net node is linked to a vector of logical values ('0', '1', or 'None'), and each transistor node has a vector of switch states ('Active' or 'Inactive'). Switch states are also assigned to structure tree nodes, with parallel nodes using an OR operation and series nodes using an AND operation. The process concludes with the creation of a Simulation Graph (SG). An example of SG, with the stimulus IN1 = '0', IN2 = '0' applied to the CG in Fig. 3, is illustrated in Fig. 4.

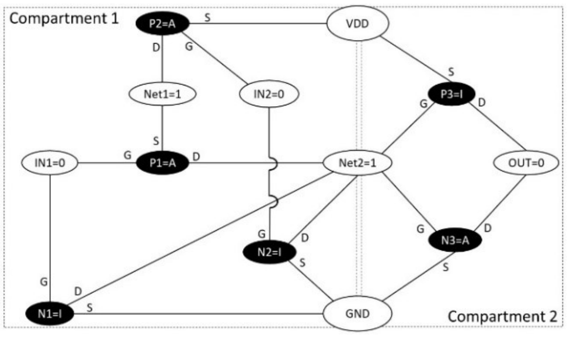

Fig. 4. OR Simulation Graph.

C. *Detection Techniques*

The final step of the flow aims at identifying PD defects impacting the cell, eliminating UD defects with a given applied stimulus. First, an initial Defect List (DL) is built, including all shorts and opens that could affect transistors. Shorts (Sh) involve combinations of Source (S), Drain (D), Bulk (B), and Gate (G) connections (e.g., ShDS for a short between drain and source), while opens (O) include OS, OD, and OG. Since opens can only be identified through dynamic detection [14], static analysis considers only shorts, while dynamic analysis evaluates both shorts and opens. Strong defects are defined as 1 Ω for shorts and 1 MΩ for opens. Two detection techniques are applied sequentially to reduce the initial DL: *Transistor Detection Rules*, and *Defect Injection and Propagation*. After each technique, the DL is reduced, and the final reduced PD DL is constructed during application of the *Defect Injection and Propagation*.

Since there can be multiple cycles in a stimulus (i.e., one cycle for static stimuli and two cycles for dynamic stimuli), it is essential to consider the impact of the defects injected for each cycle. After *Transistor Detection Rules*, if the defect is classified as UD for each cycle, the defect is classified as UD. If not, *Defect Injection and Propagation* is started. This last technique is different from the Transistor Rules. It uses fault-free information to identify UD defects. If a defect is classified as UD by this rule, it means that the fault-free behavior is not corrupted. Conversely, if the defect is still PD after this rule, it means the fault-free behavior of the simulation is corrupted, and we need to repropagate with Defect Injection and Propagation from the first PD cycle to evaluate if the defect is considered UD or PD.

Transistor Detection Rules are applied to all defects analyzing each transistor simultaneously using transistor metadata within the SG. For example, if a transistor is 'A', the ShDS defect is UD because the source and drain are already connected by the active transistor. Conversely, if a transistor is inactive, OS and OD defects are UD.

The *Defect Injection and Propagation* technique starts with a reduced list of defects obtained after applying *Transistor Detection Rules*. Each defect is injected into the graph one at a time, and its effects are propagated throughout the graph. If the output node remains unaffected, the defect is classified as UD. All UD defects are removed from the list, leaving the final PD defect list. To inject a defect, a Resistor Device Node (RDN) is added to the graph. If a conflict occurs, propagation is performed compartment by compartment, starting from the RDN-connected compartments. A decision tree is used to determine if the defect affects the current compartment. If a compartment is unaffected, the defect is classified as UD, and propagation stops. Otherwise, the defect propagates to the next compartment until either the last compartment is reached or the defect becomes classified as UD.

IV. New version of TrUnDeL

A. *New version of TrUnDeL dealing with sequential cells*

The TrUnDeL flow described in the previous subsection is effective when considering combinational cells, as it speeds up the CA model generation by 60% with no misclassifications (i.e., UD for TrUnDeL and D for CA models). However, it fails when cells are sequential, resulting in some misclassification appear. To adapt for sequential cells some modifications must be made: *Iterative Defect Injection and Propagation, Sequentialization, Initial Condition* and *Defect propagation between cycles and subcycles*. They are detailed in the following.

1) *Iterative Defect Injection and Propagation*

The methodology is applied to all the sequential cells that have mostly a configuration similar to the one shown in Fig. 5. The master part of the circuit is connected to the slave part through a switch, which is usually a Transmission Gate (TG). Managing a TG, which is bidirectional, in a compartment-by-compartment propagation with a given propagation direction (i.e., from the output of one compartment to the input of the next) can be challenging. Therefore, the first modification in *Defect Injection and Propagation* is to transform the compartment-by-compartment propagation algorithm into an event-driven iterative algorithm, similar to the one used in the *Apply Stimulus and Propagate step*.

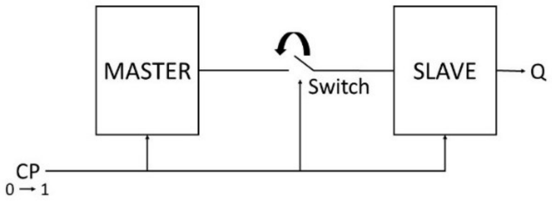

Fig. 5. General sequential cell configuration.

The difference in *Defect Injection and Propagation* is that the propagation is not fault-free; the defect effects must be considered. This means that when a conflict occurs, an X value is generated in a net node of the CG and is propagated through the net and transistor nodes of the CG. Once the steady-state response is reached for all the nets, no more events are scheduled, and the propagation stops. The output voltage is then evaluated. If it matches the fault-free value, the defect is classified as UD; otherwise, it is classified as PD.

2) Sequentialization

What primarily differentiates sequential cells from combinational cells are the feedback loops. During propagation, these loops can cause some nets to change with a small delay relatively to other nets. The example shown in Fig. 6 represents a sequential circuit composed of two compartments (C1 and C2) connected by net 'n1' and a feedback loop from the output (OUT).

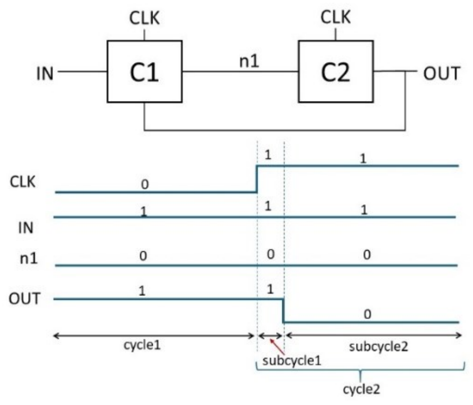

Fig. 6. Feedback loop causing a glitch.

During the rising edge of the clock (clk), OUT changes with a slight delay compared to the other nets, causing a glitch, referred to as subcycle1 in the example. In sequential defect-detection analysis, glitches must be considered, as an injected defect can cause a conflict that can affect OUT during them. We define all glitches as subcycles of a cycle. In the example, cycle2 is composed of two subcycles: subcycle1 (the glitch) and subcycle2 (the rest of the cycle). Therefore, to adapt TrUnDeL to sequential cells, detection techniques must be applied to every subcycle of each cycle.

To model the subcycles, we use the concept of switch delay of a transistor in the propagation algorithm within the *Apply Stimulus and Propagate* step and *Defect Injection and Propagation* technique. The switch delay is defined as the time a transistor takes to switch on or off. This delay is considered equal for all transistors. In the propagation algorithm, each time a transistor changes its state, a new subcycle is created to evaluate the logical values of nets and the transistor's switch state. The last subcycle corresponds to the steady-state condition where no more transistors change their switch state. This condition stops the propagation.

3) Initial condition

Sequential cells typically have a configuration like the one shown in Fig. 5. In the first cycle of the stimulus, when clk='0', the slave is disconnected from the master because the switch is open. This means that during this cycle, the slave retains its previous state, which is related to its initial condition. The slave will be set by the master only subsequently, when clk='1' and the switch is active. In the TrUnDeL flow, it was assumed that the initial condition for all the voltage values of the nets and all the switch states of the transistors were unknown. This led to some *(s, df)* pairs being misclassified (i.e., UD for TrUnDeL and D for CA models) because, in sequential cells, we always have a given previous state set to the slave. To fix this issue, it is crucial that before a new stimulus is applied, a specific initial condition of the analyzed cell is set and the slave is configured to the correct initial state.

The information related to the initial condition is provided by the CA model in the DDM through output names followed by a '-' (e.g., Q-). This represents the voltage value held by the slave before applying the new stimulus. In the new version of TrUnDeL, we extract a stimulus to set the slave from Q- and apply it through the *Apply Stimulus and Propagation* step. Then, we apply the new stimulus. Applying the initial condition generates additional cycles and subcycles compared to those already generated by the new stimulus. However, the detection techniques are always applied only to the cycles and subcycles generated by the new stimulus. The initial condition is used solely to set the slave to the desired memory state. The propagation during the initial condition application is considered fault-free.

4) Defect Propagation between cycles and subcycles

In sequential cells, it is also important to consider how defect effects propagate through the circuit between cycles and subcycles during *Defect Injection and Propagation* phase. If this aspect is not considered, the *Defect Injection and Propagation* technique can generate some misclassifications. This is because when a defect is injected and a conflict is generated somewhere in the circuit, the defect might propagate its effects to at least one output only in one of the subsequent subcycles or cycles, rather than in the one where the conflict originated.

An example of defect propagation between cycles and subcycles is shown in Fig. 7. Let us suppose that we inject a defect in the master. When clk is '0' and the switch is open, even though the defect generates a conflict in the master, referred to as 'X' in the example, its effects are not propagated because the switch is open. Conversely, when clk becomes '1' and the switch is on, the conflict generated during the previous cycle is now propagated to output Q. Therefore, the defect in question cannot be considered UD.

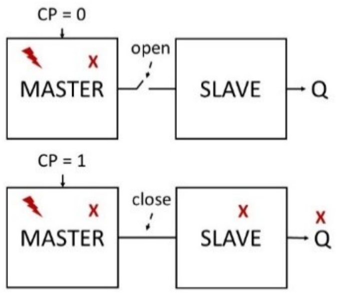

Fig. 7. Defect propagation between cycles and subcycles.

To manage this aspect, each time a conflict is generated, it is tracked in the subsequent subcycles or cycles to verify if it propagates to the output in any of those subcycles or cycles.

B. New Detection Techniques

Sequential cells are generally larger than combinational cells. In *Detection Techniques*, *Defect Injection and*

Propagation is the most expensive method, as it analyzes one defect at a time. This means that for each defect injected, it performs propagation throughout the entire CG. Consequently, the runtime for *Defect Injection and Propagation* increases with the size of the CG. Therefore, the idea is to identify UD pairs using alternative techniques able to analyze multiple defects at a time instead: *Graph Detection Rules* and *Compartment Detection Rules*.

To use the two new techniques, it is essential to perform an extraction of the structural information of the compartments. First of all, compartments are divided in three different categories. *Standard compartments*, that follow the definition in Section II. *Pass compartments,* that represent pass transistors or Transmission Gates (TGs). *In-out compartments,* that differentiate from the standard compartments by having nets that can simultaneously be output of the compartment and gate of a transistor within the compartment.

Once the compartments are classified in these three categories, the last phase of the building graph step is to extract the structural information from the standard compartments that will be then exploited in the next steps of the flow. A standard compartment consists of an NMOS network and a PMOS network connected via a common output. A generic network can be a parallel network, composed of branches, or a series network, composed of sections. Each subnetwork can further consist of other subnetworks in parallel or series. The smallest possible network consists of only one transistor. We extract all information about parallel and series networks in the form of a tree, known as a structure tree. A structure tree is constructed for each PMOS and NMOS network within every standard compartment. An example of a generic NMOS network and its corresponding structure tree is shown in Fig. 8. The network is composed of two branches (i.e., B1 and B2). B1 is composed of only one transistor. B2 is composed of two sections (i.e., Sa and Sb) in series, each one composed of only one transistor. When the necessary structural information has been extracted, we can proceed with the new detection techniques.

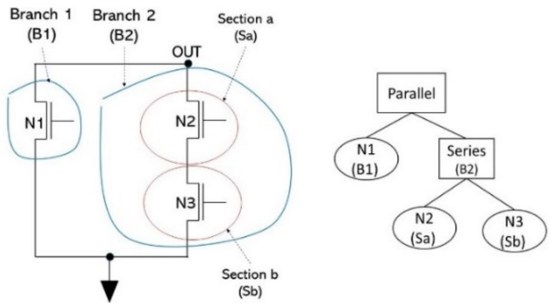

Fig. 8. NMOS network and its corresponding structure tree.

1) Graph Detection Rules

The *Graph Detection Rules* can analyze defects in cells composed of pass compartments, which can be either conducting or non-conducting after a stimulus is applied. A non-conducting pass compartment can completely disconnect a part of the cell from the output, making any defect in this disconnected part UD. In the example shown in Fig. 9, two standard compartments are disconnected by the non-conducting pass compartment, so all the defects of standard compartment 1 are UD.

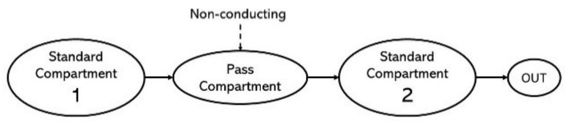

Fig. 9. Graph Detection Rules example.

2) Compartment Detection Rules

The *Compartment Detection Rules* are applied at compartment level and use the extracted structural information, evaluating one compartment at a time to assess more defects simultaneously. For example, if a stimulus is applied to the NMOS network in Fig. 8, resulting in N1 being 'A', then B1 will also be 'A'. Regardless of the state of B2, the output will always be connected to GND because B1 is active. Defects such as OS, OD, OG, ShBG, and ShDS in each transistor of B2 are UD because they only affect the switch state of N2 and N3, and thus the state of B2. Since the output depends solely on B1, these defects in B2 do not influence the output, making them UD. The final *Detection Techniques* flow is shown in Fig. 10.

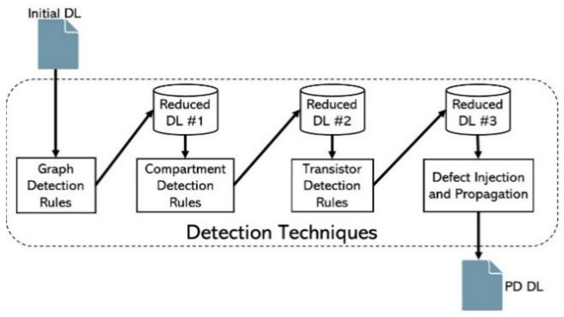

Fig. 10. Detection techniques new version of TrUnDeL.

V. Validation Results

To validate the new version of TrUnDeL, we applied it on the sequential cells of two industrial cell libraries and only transistor-level defects were considered during result analysis. Results on sequential cells of P28 are shown in Fig. 11. The new version of TrUnDeL was applied on 45 sequential cells. In total, 16,955 defects were analyzed, resulting in 146,387 *(s, df)* pairs (i.e., 76,728 static and 69,659 dynamic). Fig. 11 shows the comparison between classical CA models, which rely on analog SPICE simulations, and the new version of TrUnDeL. The bar on the left shows the percentage of UD and D defects using classical CA models. The bar in the middle represents the new version of TrUnDeL using Transistor Detection Rules (TR) and Defect Injection and Propagation (DIP). The bar on the right represents the new version of TrUnDeL using Graph Detection Rules (GR), Compartment Detection Rules (CR), TR and DIP. The methodology can identify 41.5% of the UD *(s, df)* pairs. The remaining 58.5% are categorized as PD. The new version of TrUnDeL cannot resolve these ambiguous cases due to conflicts within the cell that lead to an unknown value at the output (i.e., X). This unknown output value makes it impossible to confirm whether these pairs are UD without performing analog simulation. The gap of 31.6% observed in the results, when compared to those from classical CA models, is attributed to this limitation.

Focusing on the *(s, df)* pairs classified as UD for TrUnDeL, for both TR + DIP and GR + CR + TR + DIP, 41.5% of UD pairs are identified. Adding GR and CR techniques does not change the overall percentage of UD pairs identified by TrUnDeL but reduces the number identified by the more time-consuming DIP methods. Specifically, adding GR and CR decreases the number of UD pairs identified by DIP from 9.1% to 5.7%. In P28 library, GR do not give any contribution on the identification of UD pairs because the condition explained in Fig. 9 never happens. A key result is that, for P28, the new version of TrUnDeL does not produce any misclassifications (i.e., no D pairs are incorrectly classified as UD). This means that the accuracy for the considered cells in P28 is 100%.

For sequential cells of C28 library, the validation results are shown in Fig. 12. In total, 92 cells are analyzed, resulting in 46,262 defects and 513,178 *(s, df)* pairs (243,840 static and 269,338 dynamic). Here the gap is 34.0 %. As in P28 case, also here we obtained no misclassifications, so we do not have loss of accuracy for the analyzed cells of C28. The contribution of each phase of TrUnDeL is 0.7% for GR, 20.8% for CR, 18.5% for TR and 8.0% for DIP. Adding each phase contribution results in 48% UD *(s, df)* pairs identified. Again, adding GR and CR decreases the number of UD pairs identified by DIP from 12.3% to 8%.

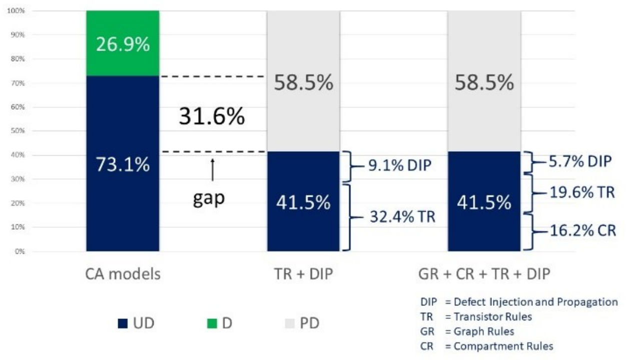

Fig. 11. Classical CA models vs TrUnDeL for P28.

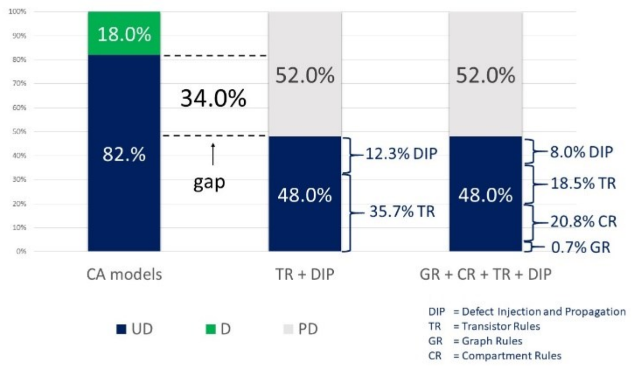

Fig. 12. Classical CA models vs TrUnDeL for C28.

Comparing P28 and C28 results, the number of UD (s, df) pairs is higher in C28 because C28 cells are generally larger. Additionally, in C28 cells, the bulk in PMOS transistors is connected to VDD, unlike in P28 cells where it is connected to GND. This results in more UD ShBS defects, as the source connected to VDD would be shorted with the bulk also connected to VDD.

We also analyzed the runtime of the new version of TrUnDeL using DIP only, TR + DIP and GR + CR +TR + DIP. The runtime results are presented in TABLE II. All the runtime estimations are done using 1 CPU and 8 MB of memory. As we can see, using DIP only is expensive in terms of runtime. Then, adding TR before, reduces the TrUnDeL runtime by 30%. Applying all the detection techniques before DIP results in a TrUnDeL runtime reduction around 40% compared to DIP only.

TABLE II. NEW VERSION OF TRUNDEL RUNTIME.

	C28 runtime	P28 runtime
DIP	67h 20m	20h 15m
TR + DIP	43 30m	14h
GR + CR +TR + DIP	39h	12h 30m

To estimate the enhancement in CA model generation time using TrUnDeL, we referred to the experiments performed on P28. Initially, classical CA models were generated through analog SPICE simulations on 45 sequential cells. The generation time took approximately 4 days and 3 hours using a single SPICE license on one CPU. Subsequently, we applied TrUnDeL to the same P28 cells. The CPU time (using one CPU) for the complete TrUnDeL process was about 12 hours and 30 minutes. The new version of TrUnDeL successfully eliminates the need of analog simulations for 41.5% of the UD (s, df) pairs for the sequential cells. The remaining 58.5% of PD pairs for sequential cells were analog simulated (using a single SPICE license), taking about 2 days and 10 hours. Therefore, combining TrUnDeL with analog SPICE simulations for the PD pairs took approximately 2 days and 23 hours, effectively reducing the time for generating the CA models by 30% compared to the classical CA model generation process.

VI. CONCLUSION AND PERSPECTIVES

In the latest semiconductor technologies, CA methodology was introduced to detect and to localize intra-cell manufacturing defects during test and diagnosis. A first graph-based methodology (i.e., TrUnDeL) was developed to speed up CA model generation process. Its limitation is that it addresses only combinational cells. In this paper, we described a new version of TrUnDeL that can manage also sequential cells implementing different modifications: *Iterative defect Injection and Propagation, sequentialization, initial condition* and *defect propagation between cycles and subcycles.* The methodology speeds up the CA models generation process for the sequential cells by 30%. Another key result is that the new version of TrUnDeL can generate CA models for sequential cells without any misclassifications (i.e., 100% of accuracy for the sequential cells analyzed in P28 and C28 technologies).

Next step will be to further reduce the gap, identifying more UD pairs to exclude them from the analog simulations with new detection techniques that will exploit physical aspects of the transistors (e.g., strength, width, length). Another perspective is to extend the TrUnDeL analysis to layout-aware defects within the cells (bridge, open via and open contact).

References

[1] G. Mongelli, et al., "A Graph-Based Methodology for Speeding up Cell-Aware Model Generation," International Symposium on On-Line Testing and Robust System Design, Rennes, France, 2024, pp. 1-6

[2] S. Eichenberger et al., ”Towards a World Without Test Escapes: The Use of Volume Diagnosis to Improve Test Quality,” *IEEE International Test Conference*, Santa Clara, CA, USA, 2008, pp. 1-10.

[3] Kyoung Youn Cho et al., ”Gate exhaustive testing,” *IEEE International Conference on Test*, Austin, TX, USA, 2005, pp. 7 pp.-777, 2005.

[4] F. Hapke et al., "Cell-Aware Test," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 9, pp. 1396-1409, Sept. 2014.

[5] Z. Gao et al., ”Defect-Location Identification for Cell-Aware Test,” *IEEE Latin American Test Symposium*, Santiago, Chile, pp. 1-6, 2019.

[6] P. Maxwell et al., ”Cell-aware diagnosis: Defective inmates exposed in their cells,” *21th IEEE European Test Symposium*, Amsterdam, Netherlands, pp. 1-6., 2016.

[7] Z. Gao et al., ”Application of Cell-Aware Test on an Advanced 3nm CMOS Technology Library,” *IEEE International Test Conference*, Washington, DC, USA, pp. 1-6, 2019.

[8] R. Guo et al., ”Efficient Cell-Aware Defect Characterization for Multi-bit Cells,” IEEE International Test Conference in Asia, Harbin, China, pp. 7-12, 2018.

[9] P. d'Hondt et al., ”A Learning-Based Methodology for Accelerating Cell-Aware Model Generation,” *Design, Automation & Test in Europe Conference & Exhibition*, Grenoble, France, pp. 1580-1585, 2021.

[10] F. Lorenzelli et al., ”Speeding up Cell-Aware Library Characterization by Preceding Simulation with Structural Analysis,” *IEEE European Test Symposium*, Bruges, Belgium, pp. 1-6, 2021.

[11] C. Chen et al., "Improving Efficiency of Cell-Aware Fault Modeling By Utilizing Defect-Free Analog Simulation," *2023 International Symposium of Electronics Design Automation,* Nanjing, China, 2023.

[12] J. P. Hayes, ”An Introduction to Switch-Level Modeling,” in *IEEE Design & Test of Computers*, vol. 4, no. 4, pp. 18-25, Aug. 1987.

[13] R. E. Bryant, ”A Survey of Switch-Level Algorithms,” in *IEEE Design & Test of Computers*, vol. 4, no. 4, pp. 26-40, Aug. 1987.

[14] J. C.-M. 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. 20.