# A Fast and Efficient Graph-Based methodology for Cell-Aware Model Generation

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

# A Fast and Efficient Graph-Based methodology for Cell-Aware Model Generation

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*—As modern Integrated Circuits (ICs) feature ever-smaller transistors, the prevalence of manufacturing defects within standard cells (intra-cell defects) has increased. Detecting and localizing these defects is crucial to guarantee a fast yield ramp-up and to maintain a low-test escape rate. Unfortunately, traditional fault models such as stuck at and transition fail to adequately represent intra-cell defects. The Cell-Aware (CA) approach was introduced to tackle this problem, but it requires time-consuming analog SPICE simulations for standard cell characterization. To speed-up the CA model generation process, this paper presents a methodology based on graph theory called Transistor Undetectable Defect Eliminator (TrUnDeL). This methodology identifies undetectable defects for each stimulus applied to the inputs of the cell, subsequently excluding them from the analog simulations to perform. TrUnDeL uses rule-based and propagation-based techniques and was trialed on combinational and sequential cells of 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 was reduced by a factor of 3 compared to a standard SPICE-based generation process.**

**Keywords—Intra-cell defects, test and diagnosis, graph theory, cell-aware model**

## I. INTRODUCTION

In advanced industrial technologies, as transistor feature size becomes increasingly smaller, the complexity of Integrated Circuits (ICs) grows with more gates being added [1]. Consequently, the occurrence of manufacturing defects such as opens and shorts within standard cells (i.e., intra-cell defects) is increasing. Identifying precise test patterns that can detect these defects in ICs and having a diagnosis process to pinpoint their location is crucial for quickly improving yield ramp-up and maintaining a low Defective Parts Per Million (DPPM) rate.

Automatic Test Pattern Generation (ATPG) algorithms are designed to create the most compact and efficient possible set of test patterns. ATPG algorithms use fault models to represent the behavior of the defect and its effect on ICs [2]. Traditional fault models like stuck-at or transition fault models typically represent defects occurring in the connections between cells (i.e. inter-cell defects). Defects within the cells themselves (i.e., intra-cell defects) are detected more by chance than by design [3].

The Cell-Aware (CA) approach has been developed to address intra-cell defects. This method involves the characterization of standard cells through the creation of a defect-detection matrix, which is essentially a reference that maps which defects can be detected by specific stimuli [4], [5]. Characterization typically involves the use of analog SPICE simulators to apply every potential input stimulus to the cell's design netlist, in which we inject every possible intra-cell defect, in order to observe the effect on the functionality. The drawback of this method is the extensive time requirement: analog SPICE simulations are slow, and fully characterizing a CA library may take several months [6].

To accelerate the integration of the CA approach in the qualification process of silicon products and its application on industrial ICs [7], it is essential to reduce the time-cost related to library characterizations. A methodology based on machine learning, proposed in [8], can predict the detectability of a defect at a cell's outputs without additional analog simulations. This is achieved by using a prediction model trained on existing CA models, generated in the past by performing exhaustive analog simulations. The limitation of this method is that the speed-up only applies to cell structures that have already been analyzed previously (i.e., cells that had at least one cell in the training data set with an identical or highly similar transistor configuration). In other words, cells with structures that never appeared in the cells of the training dataset still require the same analog simulations as before. Another approach, proposed in [9], uses a different strategy to decrease the computational generation time of CA models. The standard cells from the Cadence GPDK045 library are modeled using graph theory. With a structural analysis of the graph representations of the cells, the methodology decreases the computational generation time of the CA models, doubling the efficiency in terms of CA characterization time and cost. However, it injects defects one by one for each stimulus, making it not suitable for cells with large number of transistors. Additionally, the paper only mentions source and drain defects, omitting those related to the bulk and gate.

Our research aims to develop a methodology, called TrUnDeL, that further speeds up the generation of CA models, without compromising accuracy. TrUnDeL targets static and dynamic defects, including open and short defects affecting the source, drain, bulk, and gate of transistors. TrUnDeL draws inspiration from the approach in [9]: in our approach, we model generic standard cells as graphs. When a stimulus is applied, it propagates through the nodes, and we attach additional metadata to the graph to help identify undetectable defects for each stimulus by employing four detection techniques: Graph Detection Rules, Compartment Detection Rules, Transistor Detection Rules and Defect Injection and

Propagation. Also, unlike the method in [9], TrUnDeL is capable of handling multiple defects in parallel.

Initial tests conducted on the combinational and sequential cells from two distinct standard cell libraries, P28 and C28 from STMicroelectronics, demonstrate that applying TrUnDeL before the analog simulations reduces the computational duration of CA characterization by a factor of 3. For the purpose of validation, we compare the results with those from existing CA models created through analog SPICE simulations. It is critical for TrUnDeL to have the highest possible accuracy which means that the number of misclassifications must be reduced to the minimum (i.e., a detectable stimulus/defect pair in the CA model is never classified as undetectable by TrUnDeL). The results show that for combinational cells we reached an accuracy of 100% for both P28 and C28 libraries (i.e., no misclassifications). For sequential cells we reached an accuracy of 97.9% for C28 and 94.2% for P28 (i.e., a percentage of misclassifications exists).

This paper is organized as follows. Section II describes related prior work on CA methodology and on graph theory applications for defect-detection or circuit modelling purposes. Section III presents the TrUnDeL graph-based methodology. Validation results are presented in section IV. Finally, conclusions and perspectives are given in section V.

## II. Related Prior Work

Graph theory's versatility and efficiency are well-documented, with applications in various fields. For example, [10]-[11] present different approaches for switch-level simulations used in different areas applicable with graph theory, such as logic simulation, hardware verification and fault simulation. Most of the switch-level programs partition a circuit into a set of communicating components. A component consists of a set of transistors connected by source and drain terminals, and the nets corresponding to the different connection between the transistors. The barrier between the different components are the input nodes because no effect can be transmitted from one node to another through an input. For illustration, an OR cell segmented into components is shown in Fig. 1. Once the components are identified, the simulation evaluates a component each time its inputs change (i.e., steady state response).

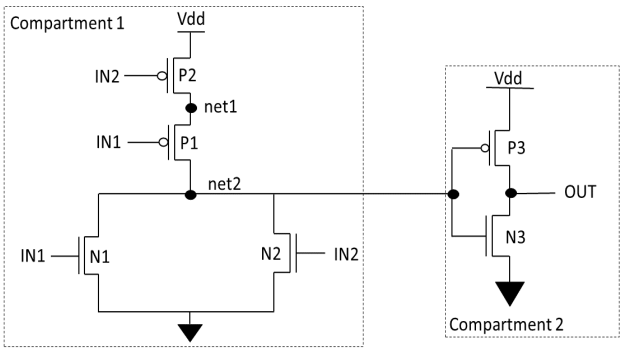


Fig. 1. OR gate divided into two components or compartments.

Switch-level simulators can also use iterative or direct methods. The first one is based on an event-driven simulator in which each time a node changes its value state, an iteration step starts. An iteration step is an operation in which the value of the event net is propagated through the transistors to the neighboring nodes. Then, the algorithm combines the new value of the node with its old value. The iterations continue until the algorithm reaches a condition called "convergence" (i.e., no more events are scheduled). Finally, the direct methods are based on systems of Boolean equations that are used to calculate the steady state response of the simulated circuit. This method is more effective for small circuits or small fractions of a circuit.

Another example given in [12] proposes a graph model of a circuit for static timing analysis. Its goal is to calculate all the static timing parameters, such as critical time and timing constraints. It uses a graph-based transistor level model of the circuit. This method uses two directed graphs to describe the circuit. In both graphs, the vertices are transistors while edges represent wires. However, this model is not easily usable for our purpose because once the graph is constructed, identifying which terminals are represented by each edge is difficult. This is important because in our method, all combinations of defects involving source, drain, gate and bulk must be considered.

Graph theory has also been used for a long time for defect detection. For example, [13] describes a method to detect open and short defects in combinational CMOS networks by applying specific test sequences to a graph-based model of the circuit. An open defect is a break in a connection between two nets while a short defect is an unwanted connection between two nets. In this graph model, transistors are represented as labeled edges, in which the label represents the conducting state of the transistor (0 or 1). Nets are represented as nodes. Two complementary graphs, G and G′, are extracted from a CMOS circuit, one representing the PMOS pull-up network and the other representing the NMOS pull-down network. A transmission function is assigned to each graph. Each time an open defect is injected, the label of the corresponding edge is set to 0. An open fault is detected when it is part of a conducting path activated by the defect and the applied stimulus. Moreover, if an open is detectable on a transistor t in G, then the corresponding short on t ′ in G′ is detectable too. However, this method does not consider defects affecting the bulk and gate of the transistors.

The approach suggested in [9] aims at reducing the time needed for CA characterization by applying structural analysis through graph theory to each library cell. The CA methodology is used to characterize a standard cell by building a dictionary in the form of a matrix, called Defect-Detection Matrix (DDM) [4]. To construct the DDM for a given standard cell, a single analog SPICE simulation is performed for each input stimulus *s* and for every potential defect *df* (consisting in pure shorts and pure opens) injected into the cell's design netlist. After completing *s* * *df* analog SPICE simulations, the characterization process is finished. Then, a status is assigned to each pair of stimulus/defect *(s, df)*: either Detectable (D) if the cell's outputs are affected by the defect under the stimulus, or UnDetectable (UD) if the outputs are not affected. The DDM includes both static and dynamic stimuli. Static stimuli consist of one-cycle patterns, whereas dynamic stimuli consist of two-cycle patterns.

The DDM is structured with rows corresponding to stimuli and columns to defects. Within this matrix, any entry of 1 or higher for a specific *(s, df)* pair means that the pair is designated as 'D'. For instance, considering a cell with two outputs, a *(s, df)* pair that affects only the first output has its corresponding entry matrix equal to 1. For a *(s, df)* pair that affects only the second output, the corresponding entry matrix is 2. Finally, if the *(s, df)* pair affects both outputs, the

corresponding entry 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 0

EXAMPLE DDM.

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

Using the CA methodology to characterize each cell in a library is a time-consuming operation as it requires conducting a series of $s * d$ analog SPICE simulations. The time it takes to characterize each cell is influenced by its number of inputs and transistors.

In the methodology presented in [9], the authors divide the cells into compartments. Compartments are subunits within the circuit defined by an output net linked to VDD via a PMOS pull-up network and to GND through an NMOS pull-down network. Each compartment is composed of transistors with gates driven by nets external to the compartment and includes nets that control transistors in external compartment. However, it excludes nets that control transistors within the same compartment. This definition of compartments is very similar to the one of the communicating components explained in [11]. The difference can be noticed if, for instance, we suppose that the circuit in Fig. 1 has a pass transistor or a Transmission Gate (TG) between the NOR and the inverter. For the definition in [11], the pass transistor or the TG would be incorporated in the first compartment with the NOR. On the other hand, the definition provided by [9] does not suggest how to handle a pass transistor or a TG.

After identifying compartments, a graph is constructed for each cell and the corresponding pattern. In this graph, active transistors are represented by edges, and nets by nodes. Injecting a short circuit adds an edge to the graph, whereas injecting an open circuit removes an edge. A *(s, df)* pair is considered as D if a short circuit creates a path between VDD and GND, or if an open circuit results in a floating output. This method is designed to detect static and dynamic defects. However, this solution focuses only on detecting open defects dynamically, which might not capture the full picture since some shorts can also be dynamically detected in CA models. Thanks to this approach, the time to generate CA models is reduced. Nonetheless, there is potential for further reduction in generation time since the method currently only addresses defects impacting the source and drain, neglecting the bulk and gate. Including these could allow for more *(s, df)* pairs to be classified as 'UD', eliminating them from time-consuming analog simulations. Moreover, the methodology's runtime can be challenging as it processes one defect at a time. This is manageable for small cells with few inputs and transistors but may reduce the method's efficiency for larger cells.

## III. TRANSISTOR UNDETECTABLE DEFECT ELIMINATOR (TRUNDEL)

TrUnDeL is a graph-based methodology that involves three sequential stages as shown in Fig. 2. The flow starts with building the graph of the cell based on the information extracted from the corresponding netlist. The second step applies a stimulus to the input nodes of the graph. Then, it performs the propagation through the different nodes. The last step exploits different detection techniques to identify the UD defects, to remove them from the initial Defect List (DL) and produce as output the Possibly Detectable (PD) DL.

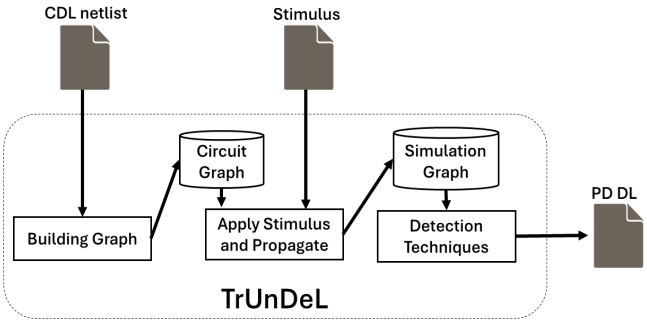


Fig. 2. TrUnDeL flow.

When TrUnDeL is run on a cell, a set of *(s, df)* pairs is identified as UD, eliminating the need for analog SPICE simulations for them. The remaining pairs, categorized as PD, will undergo simulation, reducing the time-cost for the CA model generation process. PD *(s, df)* pair means that an output can be affected by the injection of a defect under a given stimulus. These PD (s, df) pairs are then simulated by analog simulations and further classified into UD or D categories.

### A. Building Graph

This step has the Circuit Design Language (CDL) netlist as input. The netlist contains all the information about transistors and their interconnections. The Building Graph phase is divided into three sub steps: *graph construction*, *compartments identification* and *standard compartments structural info extraction*.

#### 1) Graph Construction

In this step, the Circuit Graph (CG) is built by extracting the details on the nets and transistors from the CDL netlist. It consists of two kinds of nodes: i) the devices, such as transistors, and ii) the nets, with edges representing the interconnections between devices and nets. A metadata is attached to the edges to understand the typology of the connection between the device nodes and the net nodes. Another metadata is attached to the transistor nodes to specify the type (i.e., N or P). For example, a net driving a gate of a transistor is connected to the device through a gate edge. CG of the cell in Fig. 1 is shown in Fig. 3. In this example, the transistor N1 is connected to three net nodes via a gate (G) edge, a source (S) edge and a drain (D) edge. The bulk connection is omitted here for the sake of clarity.

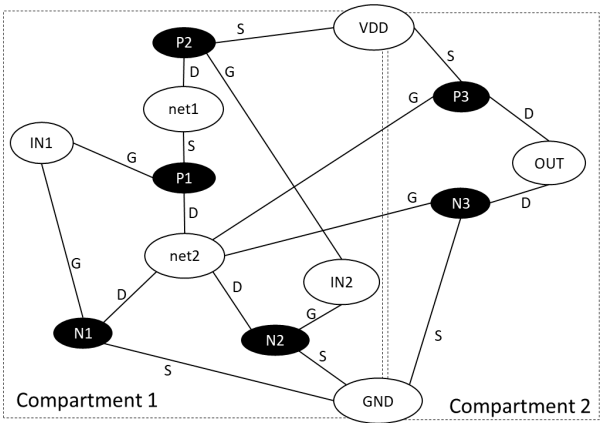


Fig. 3. CG corresponding to an OR cell.

*2) Compartments Identification*

After constructing the CG, we identify the compartments composing CG (i.e., sub-graphs of CG). We can distinguish three types of compartments: the standard compartments, the pass compartments and the in-out compartments. The standard compartments follow the definition previously explained in [9]. For instance, in Fig. 1, the cell is composed of two standard output compartments (i.e., a NOR + an Inverter). These types of compartments are characterized by a single output and have three net categories:

- *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 nest:* nets that are connected to at least one transistor of another compartment through its gate.

The first extension of the concept of a compartment is the pass compartment. For us, all pass transistors and TGs are considered as stand-alone compartments. This second type of compartment includes the input net category while another one is added: bidirectional net. This net category is added to take care about the bidirectionality of the transistors. In fact, in some cases a bidirectional net can behave as an input while other times it can behave as an output, depending on the state of the nets and transistors. The bidirectional nets are the source and drain of the transistors of a generic pass compartment. A pass compartment with its net categories is shown in Fig. 4 in the example *a*. The last type of compartment is the in-out compartment. It includes the internal net category as in standard compartment, but it includes a new net category, called in-out nets. This type of net can be at the same time output of the compartment or gate of a transistor inside the compartment, differently from the concept expressed in [9] where a net cannot drive a transistor inside the same compartment. An example of an in-out compartment is shown in Fig. 4 in case *b*.

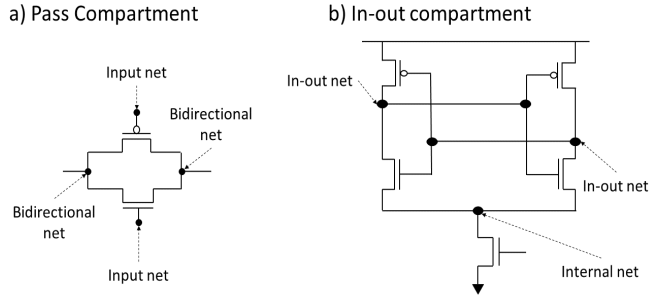


Fig. 4. Example of a pass compartment and an in-out compartment.

*3) Standard Compartments Structural Info Extraction*

Once compartments are identified, structural information is extracted from each standard compartment. Before explaining how we extract this information, defining some concepts about the structure of a standard compartment is essential. A standard compartment is composed of an NMOS network and a PMOS network that are connected via a common output. More specifically, a generic network can be either a parallel network, composed of subnetworks that we call branches, or a series network, composed of subnetworks that we call sections. Each subnetwork can be composed of other subnetworks either in parallel or in series, and so on. The smallest possible network is composed of only one transistor.

An example of parallel network is shown in Fig. 5. It 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.

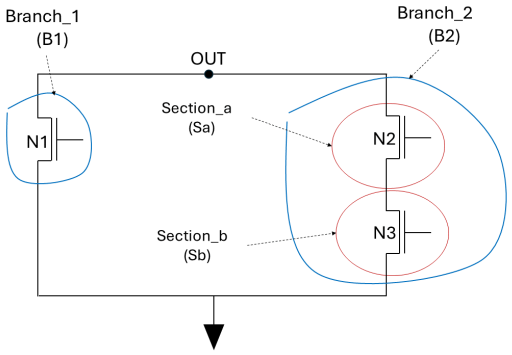


Fig. 5. Example of a parallel network.

Another example in Fig. 6 shows a complementary case with respect to the one analyzed above. This case is a series network composed of two sections (i.e., S1 and S2). S1 is composed of one transistor and S2 is composed of two branches (i.e., Ba and Bb) in parallel, each one composed of only one transistor.

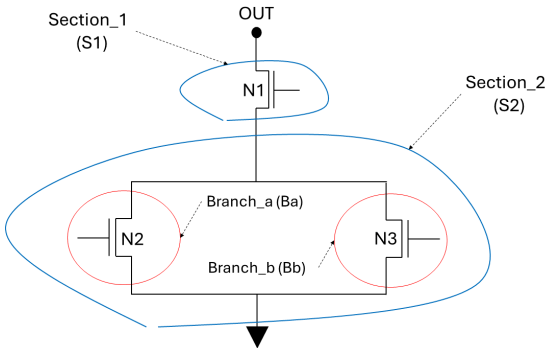


Fig. 6. Example of a series network.

Moreover, we describe a compartment as complementary when its NMOS network is the complement of its PMOS network. Two networks are considered complementary when they have opposite structures, and the inputs drive corresponding opposite transistors. For example, the network in Fig. 5 is complementary to the one in Fig. 6. We extract all the info about parallel and series networks in a tree form, called structure tree. The structure tree is built for each PMOS and NMOS network of every standard compartment. For instance, Fig. 7, shows the structure trees of the networks in Fig. 5 and Fig. 6, respectively.

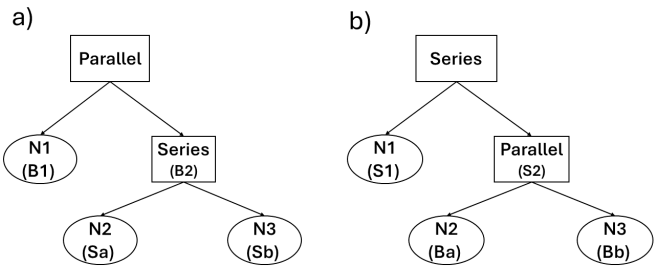


Fig. 7. Structure trees of the two networks in Fig. 5 and Fig. 6.

To determine whether a standard compartment is complementary, we examine its corresponding PMOS and NMOS networks. This involves analyzing the structure trees

of both networks, comparing each node, and ensuring that the inputs are connected to the gates of matching transistors in both networks. Through this method, we can confirm the complementarity of the standard compartment.

### B. *Apply Stimulus and Propagate*

This step works as a classical event-driven switch level simulator based on an iterative method. The step starts by applying the stimulus to the input nets of CG. A stimulus consists of a single cycle for static stimuli and a pair of cycles for dynamic stimuli. Every cycle within a stimulus represents a sequence of logical '1's or '0's that are assigned to the input nets. Here, a '1' corresponds to the voltage supply level (VDD), and a '0' corresponds to the ground level (GND). An event is scheduled for each input net and propagation starts from the device nodes driven by them. Propagation is performed from net to device and device to net, raising a new event each time a net changes its state value. The propagation continues, until the algorithm reaches the convergence state (no more events are scheduled). In dynamic stimuli case, the process starts again to evaluate the second stimulus cycle.

Upon completion of the stimulus application and propagation, each net node is linked with a vector of logical values. These values can be '0', '1', or 'None', with 'None' indicating that the node is isolated, meaning it is not connected to either VDD or GND. Similarly, every transistor node has a vector of switch states, which can be 'Active' (A) or 'Inactive' (I), determined by the value connected to its gate. Each entry within the vectors of logical values and states corresponds to a specific stimulus cycle. Moreover, a switch state is also associated to each node of the structure trees. For parallel nodes, the switch state is equal to the *or operation* between its child nodes. The switch state of the series nodes is equal to the *and operation* between its child nodes. The switch state of each transistor node is equal to its own switch state.

Finally, a strength is assigned to each net value. The strength can be a value between 1 and 4. We assign 4 to power nets (i.e., VDD, GND), 3 to input nets, 2 to the nets driven by transistors and 1 to all the nets that have the current cycle value depending on the previous cycle. At the conclusion of this step, a different graph, known as the Simulation Graph (SG), is produced. For instance, referring to CG depicted in Fig. 3, an SG is illustrated in Fig. 8 when a specific stimulus is applied (IN1 is set to '0', IN2 is set to '0'). Each transistor node has a given switch state while each net node has is associated with a voltage value and a specific strength (i.e., *Sth* in the figure).

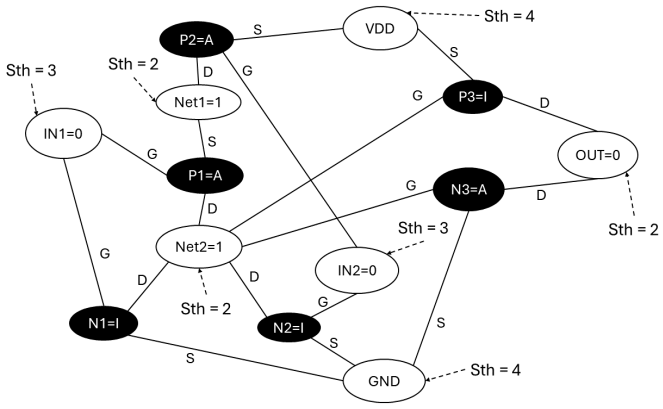


Fig. 8. SG of OR, stimulus: IN1 = '0', IN2 = '0'.

### C. *Detection Techniques*

The goal of the final stage, depicted in Fig. 2, is to identify a comprehensive set of potential defects impacting the cell (i.e., PD defects), therefore eliminating those UD with a given applied stimulus. At first, we build the initial DL of the cell, comprising every short and open that could affect the different transistors, as shown in Fig. 9. Shorts (Sh) consist of every combination of Source (S), Drain (D), Bulk (B), and Gate (G) connections for each transistor. Opens (O) include OS, OD, and OG. Since opens are only identifiable through dynamic detection [14], static analysis considers only the shorts, whereas dynamic analysis evaluates both shorts and opens.

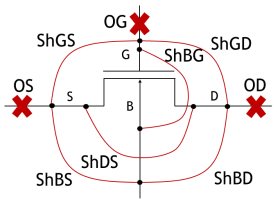


Fig. 9. Targeted defects for a transistor.

We apply in sequence four detection techniques to eliminate UD defects from the initial DL: *Graph Detection Rules, Compartment Detection Rules, Transistor Detection Rules* and *Defect Injection and Propagation*. After each technique, the initial DL is reduced. During the last technique, the final reduced list is constructed (i.e., PD DL). In static analysis, a single-cycle stimulus (i.e., static stimulus) is applied. A defect is classified as UD for a given static stimulus when any one of the four detection techniques classify it as UD, during this single cycle. In dynamic analysis, a two-cycle stimulus (i.e., dynamic stimulus) is applied. A defect is classified as UD for a given dynamic stimulus when any one of the four detection techniques classify it as UD, for each cycle of the two-cycle stimulus. This process allows us to build the list of PD defects. The detection techniques process is presented in Fig. 10.

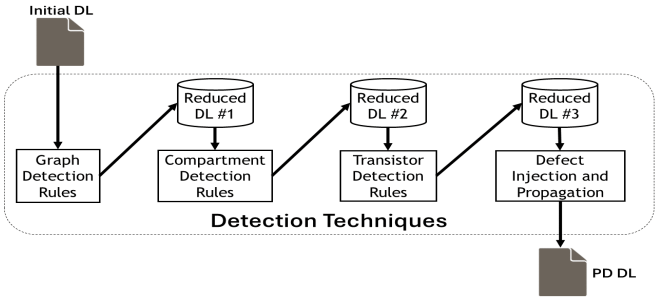


Fig. 10. Detection Techniques flow.

#### 1) *Graph Detection Rules*

These rules can be applied to all the circuits that are composed of pass compartments. After the stimulus is applied, the pass compartments can be in two opposite states: either conducting or non-conducting. Sometimes, a non-conducting pass compartment can completely disconnect a part of the circuit from the output. It means that any defect injected in this disconnected part of the circuit is UD. An example of these rules is shown in Fig. 11. The represented cell is composed of three compartments. Specifically, standard compartment1 is connected to standard compartment2 through a pass compartment. Moreover, only standard compartment2 is connected to the output. Suppose that the pass compartment is in non-conducting state. It means that standard compartment1

is completely disconnected from standard compartment2, therefore it is disconnected from the output. So, all the defects injected that affect only standard compartment1 are UD, as they cannot affect the output. Potentially, it means that in specific conditions all the defects belonging to different compartments can be classified as UD in one shot.

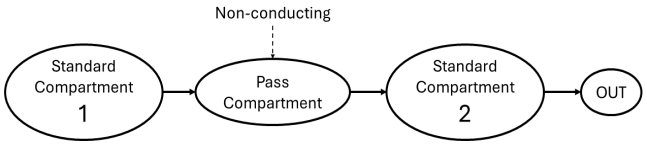


Fig. 11. Example graph rules.

*2) Compartment Detection Rules*

These rules are applied at compartment level, considering one compartment at a time. This technique can evaluate more defects inside a compartment simultaneously. To identify UD defects, some compartment rules exploit the structural info about each compartment, extracted during the stage A.3). For example, suppose a stimulus is applied to the NMOS network of Fig. 5 for which the transistor N1 is 'A'. As we can see in the corresponding structure tree of Fig. 7 (i.e., case *a)*, B1 is in parallel with B2. B1 is active, as N1 is active. It means that the output is connected to GND whatever the state of the parallel branch B2. So, OS, OD, OG, ShBG and ShDS of each transistor of B2 are UD. In fact, these defects can only change the switch state of N2 and N3, consequently the state of B2. Therefore, considering that the output depends only on B1, these defects are UD. The formal corresponding rule says that if a branch *Bx* has an active branch in parallel, OS, OG, OD, ShBG and ShDS of each transistor of *Bx* are UD. In the case of series sections, the methodology is opposite to the parallel branches case.

Another example of compartment rule can be explained by looking at the complementary standard compartment in Fig. 12.

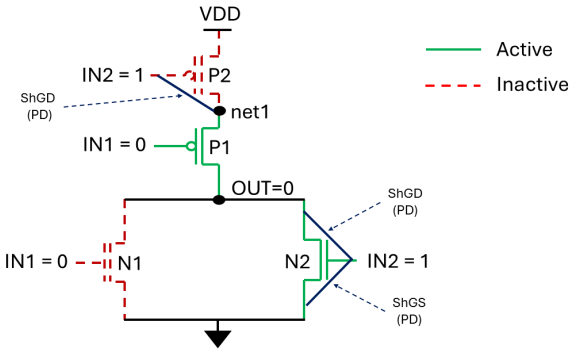


Fig. 12. Standard compartment rule for ShGS and ShGD.

In this case, the output (OUT) voltage value of the compartment is equal to '0'. For the NMOS network the only PD defects are ShGS and ShGD of the active transistors connected to OUT. So, ShGS and ShGD of N2 are PD. The reason is that these defects connect the gate to the OUT through an active path. For N2 it means that IN2, with a voltage value = '1', is connected to OUT, with a voltage value = '0'. This causes a conflict, and the defects are considered PD. For the PMOS network, the only PD defects are ShGD of the inactive transistors either directly connected to the OUT or connected to the OUT through consecutive active transistors. In PMOS network, only ShGD of P2 is PD because it creates again a conflict between IN2 and OUT. On the other hand, if the voltage of OUT is equal to '1', the case is opposite to the example just explained.

In the last example, we consider the case of a pass compartment, specifically a TG. The structure is composed of two transistors in parallel, one PMOS and one NMOS. Suppose that the TG is conducting (i.e., PMOS and NMOS switch states are 'A'), and voltage values of the bidirectional nets are equal to '1'. In this case, OS, OD, OG and ShDS of the NMOS transistor are UD. In fact, if one of these defects is injected, the state of the NMOS transistor could switch off and the only remaining 'A' transistor would be the PMOS. However, the PMOS transistor is good in conducting '1' so, the injection of one of these defects does not affect the behavior of the TG. On the other hand, only the ShDS of the PMOS transistor is UD. The opens injected in the PMOS transistor could switch off its switch state and the only remaining 'A' transistor would be the NMOS one. NMOS transistor is not good in conducting '1', so we cannot be sure that the opens are UD too.

*3) Transistor Detection Rules*

This technique applies rules to all defects that impact each transistor simultaneously, addressing each transistor one by one. This process requires no additional steps for the SG, as it uses the existing transistor metadata within the SG. Fig. 13 illustrates some rules at the transistor level.

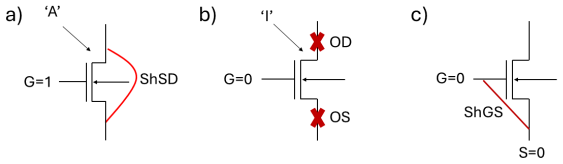


Fig. 13. Examples transistor detection rules.

For instance, in example *a*, an active transistor (i.e., G=1) means that ShDS defect is irrelevant if injected. In example *b*, an inactive transistor (i.e., G=0) indicates that OS and OD defects have no impact. Finally, in example *c*, G and B are equals, so ShGS has no impact.

*4) Defect Injection and Propagation*

This is the last step of the detection techniques before obtaining the final PD DL. It injects each defect of the reduced DL into the graph subsequentially. Then, it propagates the defect effects throughout the graph. Finally, it evaluates the output voltage value. If it is equal to the fault free one, the defect is UD, otherwise the defect is PD. After all the defects have been injected and their effects propagated across the graph, it is possible to eliminate all UD defects from the DL, culminating in the creation of the final PD DL.

The first step of this technique is to inject the defect as a Resistor Device Node (RDN) inside the graph. Then, the propagation starts from the compartments connected to RDN, and it continues compartment after compartment. In order to verify if the current compartment is affected by a resistor defect injection in the CG, a decision tree is applied:

- If there is a pathway from VDD to GND composed only of active transistors, then the defect affects the compartment.
- If the output is connected through a path exclusively to VDD or GND, where all transistor device nodes are

active, the output is evaluated against the fault free one. A match indicates that the defect does not affect the compartment.

- If there is not a pathway, where all transistor device nodes are active, connecting OUT to VDD, and OUT to GND, then the defect is considered to affect the compartment.

When the current compartment remains unaffected, it means that its output is consistent with the fault free one. In this scenario, the algorithm classifies the defect as UD and propagation stops. Conversely, if the compartment is affected, the defect injection can have an impact on the subsequent compartment, with the output of the current one becoming the input for the next. This propagation persists until either the final cell compartment's output is reached, or the defect is classified UD in any compartment along the propagation path.

## IV. Validation Results

This section presents the validation results obtained when applying TrUnDeL to combinational and sequential cells of two standard cell libraries designed by STMicroelectronics (i.e., C28 and P28). The following subsections detail the different stages of the conducted experimental procedures.

### A. Combinational cells of C28 and P28

In the C28 library case, TrUnDeL was applied on 536 combinational cells. In total, 119 088 defects were analyzed, resulting in 4,545,684 *(s, df)* pairs (i.e., 1,351,920 static and 3,193,764 dynamic). Fig. 14 illustrates the comparison between classical CA models, which rely on analog SPICE simulations, and TrUnDeL. The graph on the left shows the percentage of UD and D defects using classical CA models. The graph in the middle represents TrUnDeL using *Transistor Detection Rules* (TR) and *Defect Injection and Propagation* (DIP) only. The last graph on the right represents the complete TrUnDeL solution, that uses in sequence *Graph Detection Rules* (GR), *Compartment Detection Rules* (CR), TR and DIP.

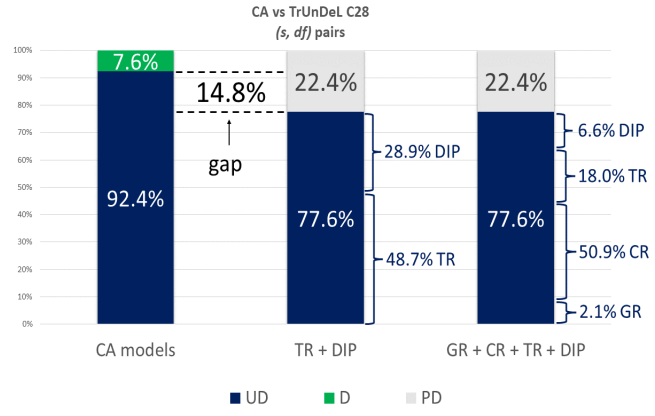


Fig. 14. Classical CA models vs TrUnDeL for C28 combinational cells.

A key result is that TrUnDeL does not produce any misclassifications (i.e., no D pairs are incorrectly classified as UD). TrUnDeL can identify 77.6% of the UD *(s, df)* pairs. The remaining 22.4% are categorized as PD. 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 14.8% 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, we can notice that for both TR + DIP and GR + CR + TR + DIP the percentage of UD identified pairs is 76,6%. Adding the two additional techniques (i.e., GR and CR) does not change the overall percentage of UD pairs identified by TrUnDeL. However, it decreases the number of UD pairs identified with TR and DIP technique, as some are now identified by CR and GR instead. This shift is particularly significant for DIP, which is the most time-consuming method applied and it can significantly impact TrUnDeL speed if overused. Specifically, adding GR and CR reduces the number of pairs classified as UD for DIP by 4 times (6.6 % compared to 28.9 %). In summary, the complete TrUnDeL identifies 2.1% of UD with GR. CR can identify most UD pairs (i.e., 50.9%) while TR identify 18.0% of them. The rest, amounting to 6.6%, are detected through DIP.

In P28 case, TrUnDeL was applied on 932 combinational cells. 110 092 defects were evaluated, resulting in 3,669,967 *(s, df)* pairs (1,215,792 static and 2,454,175 dynamic). Also in this scenario, TrUnDeL does not generate any misclassified pairs and identifies 66.0% UD *(s, df)* pairs, resulting in a 23% gap. The comparison between the classical CA models and TrUnDeL is shown in Fig. 15.

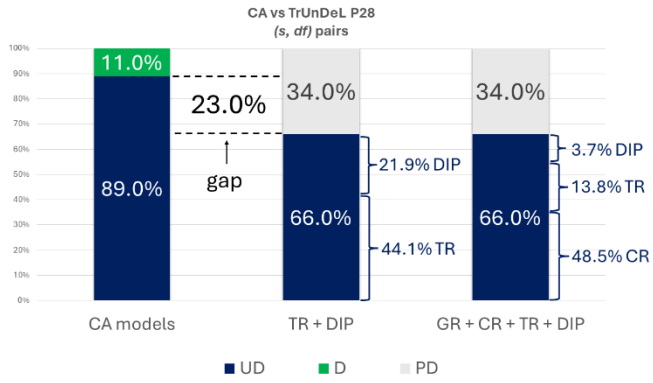


Fig. 15. Classical CA models vs TrUnDeL for P28 combinational cells.

In this case, 48.5% of UD *(s, df)* pairs are identified only using CR. The remaining pairs are identified by TR (13.8%) and DIP (3.7%). The percentage of pairs identified as UD with DIP is reduced by 5 times, using the complete TrUnDeL.

As we can see, comparing the result in C28 and P28, the GR give their contribution to the C28 library cells only. This happens because the GR conditions, as the one explained in Fig. 11, never happened to the P28 library cells.

The results indicate that a substantial portion of *(s, df)* pairs can be classified as UD by TrUnDeL. This means that we can exclude these pairs from the time-consuming analog simulations. Only the remaining PD pairs will be analog simulated for further categorization as either UD or D. As a result, we obtain a CA model generation time reduction by a factor of 3 compared to classical CA model generation process, as explained in detail in the subsection F.

### B. Sequential cells of C28 and P28

For sequential cells of C28, the validation results are shown in Fig. 16. In total, 95 cells are analyzed, resulting in 47,810 defects and 586,021 *(s, df)* pairs (269,568 static and 316,453 dynamic). Here the gap is 29.8 %. Differently from the combinational cells case, in sequential cells we obtained a percentage of misclassifications (i.e., 2.1%). The contribution of each technique is 0.7% for GR, 20.3% for CR, 17.0% for

TR and 14.7% for DIP. Using GR and CR decreases the usage of DIP by 2 times.

Results on sequential cells of P28 are shown in Fig. 17. We analyzed 45 cells, resulting in 16,955 defects and 146,387 *(s, df)* pairs (76 728 static and 69 659 dynamic). The gap for these cells is 29.8%. A percentage of misclassifications also exists here (i.e., 5.8%). Here, the contribution of each technique is 18.7% for CR, 16.8% for TR and 20.8% for DIP. Again, the percentage of UD defects identified by DIP is reduced. As in combinational case, the GR technique does not have a contribution in P28. The reason is the same explained in subsection A.

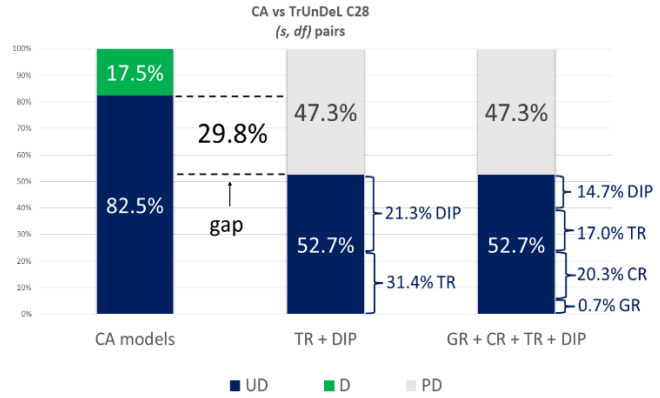


Fig. 16. Classical CA models vs TrUnDeL for C28 sequential cells.

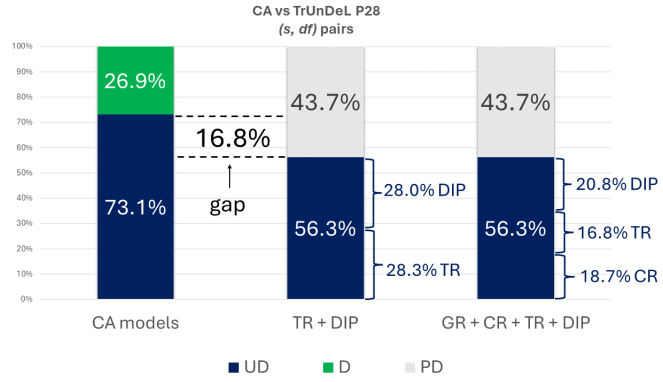


Fig. 17. Classical CA models vs TrUnDeL for P28 sequential cells.

A key difference between combinational and sequential cells is the real number of cycles for static and dynamic stimuli. A static stimulus, in a combinational cell is composed of a single cycle. In a sequential cell case, a static stimulus is composed of two cycles, as the clock (clk) goes from '0' to '1' to simulate the rising edge. Similarly, with dynamic stimuli, the concept remains consistent. In sequential cells, the clk rising edge occurs twice, resulting in a stimulus comprising four cycles.

As in combinational cell case, in sequential cells the percentage of UD *(s, df)* pairs identified by TrUDeL is big and only the remaining PD pairs will be simulated. Consequently, the CA model generation time is reduced by a factor of 3 compared with classical CA model generation process (subsection F).

### C. *Observation about the gap in C28 and P28*

As already explained in subsection A, the gap depends on the ambiguous cases due to conflicts within the cell that lead to an X value at the OUT. The propagation of the conflict to the OUT in a cell mainly depends on two factors:

- The number of transistors inside the cell.
- How these transistors are distributed in the different compartments inside the cell.

It is more probable that *(s, df)* pairs will not be able to propagate their effects in cells containing more transistors, and thus are ultimately classified as UD by the TrUnDeL process. However, cells with the same number of transistors can have a different probability to propagate or not the defect effect until the OUT, considering how the transistors are distributed inside the different compartments. To better understand this concept, we can compare the examples *a* and *b* in Fig. 18. The number of transistors is equal in both circuits (i.e., 6). In case *a*, four transistors are inside compartment1 and two in compartment2. In case *b*, the transistors are equally distributed in the three compartments. In case *a*, if IN1 = '0' and IN2 = '1'. ShDS of N3 propagates an 'X' to the output of compartment1. However, this 'X' does not affect compartment2, as IN2 switches on N2 and switches off P2, so OUT = '1'. It means that the defect is masked at compartment2. In case *b,* if IN = '0', the effect of the defect is propagated until the OUT through the three compartments. So, we see how two circuits have the same number of transistors but the defect in one case is not propagated until the OUT and in the other one it is propagated. This depends on how the transistors are distributed in the different compartments of the circuit.

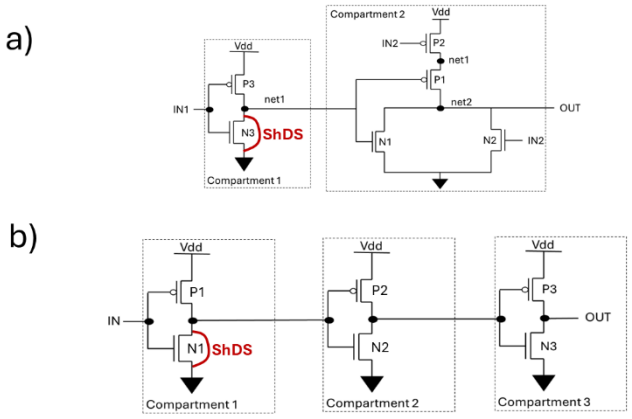


Fig. 18. Different transistors distribution in two circuits.

Comparing C28 and P28, in the case of combinational cells, we see that C28 has a narrower gap and a higher number of UD *(s, df)* pairs. This is attributed to C28 having generally bigger cells (i.e., bigger number of transistors) with a distribution of the transistors inside the compartments that increases the probability of *(s, df)* pairs not propagating their effects until the OUT. On the other hand, in sequential cells, although cells in C28 are still bigger than in P28, the gap in C28 is bigger than in P28. In fact, most of the UD *(s, df)* pairs that are identified in C28 belong to the ambiguous case in which X is propagated until the output of the circuit. This is due to the distribution of the transistors in the different compartments that decreases the probability of having a defect effect masked during the propagation.

### D. Discussion about misclassifications on sequential cells

If for combinational cells no misclassifications occur, this is not the case for the sequential ones. Analyzing the different misclassifications cases, we found the reasons that explain this result. For example, Fig. 19 illustrates a high-level cell composed of two compartments (i.e., C1 and C2) with a feedback loop between the OUT and C1. In the example, *clk* rising edge is represented and IN = '1'. Subsequentially, OUT goes from '1' to '0'. It is important to consider that the OUT changes its voltage value with a small delay. This small delay causes a "mini cycle" between the main two cycles. A conflict, caused by a defect inside C1 or C2, can propagate its effect to the output during cycle1 or cycle2, but also during the "mini cycle". This is one of the cases that are not captured from TrUnDeL, that for now does not consider timing related aspects of the cells.

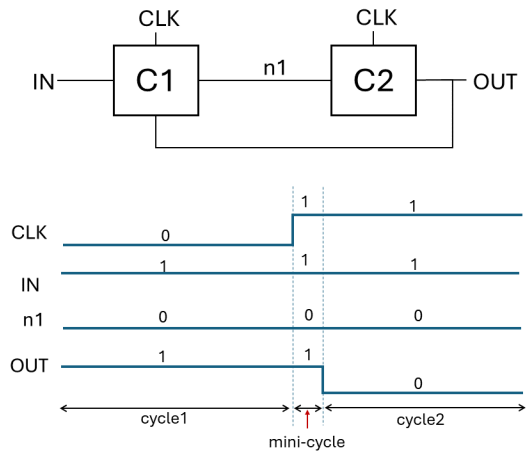


Fig. 19. Feedback loop between compartments.

Another critical case in which the timing can create an issue is the in-out compartment. In fact, this type of compartment intrinsically includes a feedback loop between the in-out nets and some transistors. This could create a similar situation in terms of waveforms to the one shown in Fig. 19. An additional mini cycle could be generated, in which a defect injection could propagate its effect until the OUT.

The last case of misclassification that we analyze depends on the initial state of the cell. In fact, in some cases, knowing the initial state of a sequential cell, before the stimulus is applied, is essential to understand if a defect is propagated until the OUT. This kind of misclassification can be solved by applying the correct initial state to the SG and consider it as a new cycle to be evaluated through detection techniques.

### E. TrUnDeL runtime

We also analyzed how TrUnDeL behaves in terms of runtime using the different detection techniques, as shown in 0All the runtime estimations are done using 1 CPU. For all the combinational and sequential cells of C28, using TrUnDeL with only DIP took 18h 48m. Adding the TR using DIP improves the runtime by 42%. If we apply the complete TrUnDeL, the runtime is improved by 58% compared to the TrUnDeL with only DIP applied. For combinational and sequential cells of P28, the improvement is confirmed. Using only DIP, TrUnDeL took about 9h. adding TR reduces the runtime by 38% and the complete TrUnDeL improves the runtime by 56%. In other words, using all the techniques in sequence reduces the runtime for *(s, df)* pairs analysis by TrUnDeL by more than one half.

TRUNDEL RUNTIME C28 AND P28.

| | C28 runtime | P28 runtime |
|---|---|---|
| DIP | 18h 48m | 9h 6m |
| TR + DIP | 10h 44m | 5h 36m |
| GR + CR + TR + DIP | 7h 42m | 4h 8m |

Then, we analyzed the runtime for a cell in function of its number of transistors. When more cells had the same number of transistors, we made the average. This is useful to understand the trend of the TrUnDeL runtime when the cells become larger. Again, the runtime estimations were performed using 1 CPU. We plotted all the points and the quadratic regression that interpolates them in the graph, as shown in Fig. 20.

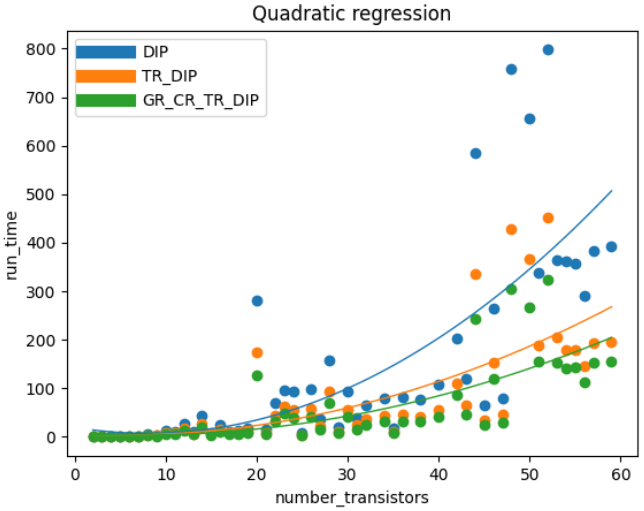


Fig. 20. Quadratic regression runtime in function of number of transistors.

The graph shows three curves. The blue one represents TrUnDeL using only DIP, the orange one represents TrUnDeL using TR + DIP, and the green one represents the complete TrUnDeL solution (i.e., GR + CR + TR + DIP). Each curve has its corresponding equation:

- DIP: $y = 0.19x^2 - 3.09x + 19.67$
- TR + DIP: $y = 0.09x^2 - 0.96x + 6.36$
- GR + CR + TR + DIP: $y = 0.07x^2 - 1.03x + 7.10$

As the number of detection techniques incorporated into the analysis grows, the coefficient of the quadratic term decreases. Consequently, with a higher number of transistors, the runtime decreases at a reduced rate when additional detection techniques are used. Specifically, we graphically see that the DIP curve is the one for which the runtime grows faster. TR + DIP curve grows much slower than DIP. Finally, the curve with the complete TrUnDeL is the one with the minor slope.

### F. CA model generation time reduction

To estimate the enhancement in CA model generation time using TrUnDeL, we referred to the experiments performed on P28. Initially, the classical CA models were generated through analog SPICE simulations on 977 cells of P28 (i.e., 932 combinational and 45 sequential). The generation time took approximately 96 days and 8 hours, using a single SPICE license on 1 CPU. Subsequentially, we applied TrUnDeL to the identical P28 cells. The CPU time (using 1 CPU) for

complete TrUnDeL was about 4 hours. TrUnDeL was able to skip analog simulation for 66% of the UD *(s, df)* pairs for the combinational cells and the 56.3% of the UD *(s, df)* pairs for the sequential cells. The remaining 34% of PD pairs for combinational cells and 43.7% PD pairs for sequential cells were analog simulated (using a single SPICE license), taking about 35 days and 1 hour. Therefore, combining TrUnDeL with analog SPICE simulations for the PD pairs took approximately 35 days and 5 hours, effectively reducing the time for generating the CA models by a factor of 3 compared with classical CA model generation process.

## V. Conclusions and perspectives

In modern ICs the number of manufacturing defects inside standard cells has increased a lot. CA methodology was introduced to detect them and diagnosis purposes. The drawback of CA methodology is that it is based on time-consuming analog simulations. In this paper, we described the different steps of TrUnDeL, a methodology that speeds up CA models generation process (by a factor of 3 for P28 used as test case). TrUnDeL can generate CA models for combinational cells without any misclassifications (i.e., 100% of accuracy for the combinational cells analyzed in P28 and C28).

TrUnDeL has also been applied on sequential cells of the same libraries, generating CA models with a percentage of misclassifications (i.e., 97.9% of accuracy for C28 and 94.2% of accuracy for P28). Analyzing the different types of misclassifications that occur during the experiments, we have seen how an important improvement is to handle the timing aspects of the cells, to increase as much as possible the accuracy of the generated CA models. Moreover, another aspect that can further improve the accuracy is to consider an additional cycle for the initial state in sequential cells. In order to further reduce the gap, the plan is to incorporate more information about the technology, the physical characteristics and the strengths of the transistors, and the defect size.

Finally, we evaluated the runtime of TrUnDeL for the two libraries, and we analyzed the runtime of TrUnDeL in function of the number of transistors inside a cell, considering the different detection techniques implemented. The results show that using the techniques in sequence leads to a more gradual rise in the runtime curve when the number of transistors within the cell increases. This makes our solution suitable also for larger circuits, such as memories.

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

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

[12] 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.

[13] 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.

[14] 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.