Trabajo de Fin de Máster
"Máster Universitario en Microelectónica:
Diseño y Aplicaciones de Sistemas Micro/Nanométricos"

Optimización del flujo de simulación de defectos y fallos en circuitos analógicos y de señal mixta

Estudiante: Valentín Gutiérrez Gil

Tutor: Gildas Leger

Septiembre 2018

Optimization of the defect and fault simulation flow in analog and mixed signal circuits

Student: Valentin Gutierrez Gil
Advisor: Gildas Leger

Trabajo Fin de Máster:
Máster Universitario en Microelectónica:
Diseño y Aplicaciones de Sistemas

September 2018

Acknowledgements

I would like to express my deepest appreciation to all those who provided me the possibility to complete this work. A special gratitude I give to my advisor, Dr. Gildas Leger, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Furthermore, I would like to with much appreciation to the members of the n-PATETIC project, namely, Adoración Rueda, Antonio López, Antonio Ginés, Eduardo Peralías and my advisor for giving me not only the opportunity but for teaching me everything I'm proud of.

Finally, an special thank to my family, whether blood or friendship, for enduring this whole journey with me.

Contents

Chapter 1

Introduction

1.1 The goal: Test validation

Historically, the golden standard for analog, Mixed-signal and RF circuit (AMS-RF) test has been functional testing. Indeed, an AMS-RF circuit is defined by its datasheet: a number of performance metrics associated with standard measurement procedures are guaranteed to be within a given range. For instance, the static Integral Non-Linearity of and Analog-to-Digital Converter can be specified to be below 1.4 Least Significant Bits and there that metric can be measured in several ways like histogram testing or servo-loop, as defined in the IEEE 1241 standard [1]. With such a definition, it seems perfectly logical to consider functional testing as a reference: since the datasheet is what is sold to the customer, performing the measurements that are associated to the specified performance parameters should, by definition, guarantee the detection of any faulty behavior.

While this is still true, there are two trends that challenge the suitability of functional test. The first and oldest one is cost: AMS-RF circuits are more and more complex and embedded in larger systems. As a result, it may be difficult to reach the primary input and output of the circuits. In addition, the number of specifications increases and some of them can involve lengthy measurements (the direct measurement of the Error Vector Magnitude in transceivers is a good example). The test time is thus on the rise and so the test cost. By the way, the test cost of a modern System-on-Chip can be dominated by the analog section in spite of the clear superiority of digital circuits in terms of area.

The second and more recent trend questions the quality of functional test itself. Indeed, functional test is performed at time zero but there may be defects or deficiencies that cause a minimum performance degradation at time zero, still within the specifications, but may evolve in time with circuit aging, or simply be stimulated by real-world usage. In safety- or mission-critical applications (like automotive, health-care, space, ...), defects are perceived as a threat even if they do not impact performance in the production line. In the automotive industry, some cases of field issues could be traced to manufacturing defects that were not actually detected by functional test.

As a response to these two trends, there exist a demand for test alternatives to functional test. Some of the proposed solutions respond essentially to the cost issue like reduced (non-exhaustive) functional test, indirect test [2–4] and its recent machine-learning update, Built-In Self-Test (BIST) approaches, etc. And other solutions rather try to mimic what happened in the digital world –where functional test quickly became prohibitive due to the ever increasing number of states to be tested. These solutions aim at detecting defect and their primary goal is not to measure the specifications but rather to guarantee that the circuit exactly corresponds to what was designed. These solutions are gathered under the term of Defect-Oriented Testing, even if they are mostly ad-hoc to a particular topology or at least circuit family.

Independently of the cause, the shift from exhaustive functional testing is a paradigm change from the point of view of validation: If exhaustive functional test is impossible, too costly or simply not considered as perfect anymore, then the test strategy has to be validated before its implementation in the production line. If it involves some design modifications (DfT or BIST) the quality must be assessed at early phases of the design. Unless some mathematical proof of test quality could obtained through formal methods, that validation necessarily involves defect simulation [5, 6].

1.2 The issue: Reducing the computational cost

In the case of an exhaustive functional test, part of its validation is implicitly done in the design phase. Indeed, the performance measurements are simulated under both process and mismatch variations and the obtained metrics are verified to remain within their specified validity ranges. By definition, an exhaustive functional test would detect any deviation out of the specifications. However, if the target is to assess the defect coverage (which in the case of exhaustive functional case is the proportion of defects that do affect performance), defect simulation has to be done.

If a Defect-Oriented Test or any alternative to exhaustive functional test has to be evaluated, then it should be validated with respect to process and mismatch variations to verify that it does not produce yield losses (i.e. false positive) or test escapes (i.e. false negatives) under normal process conditions. And obviously, a defect simulation would also have to be performed to assess the defect coverage of the proposed test. So as we can see, defect simulation seems unavoidable.

The number of possible defects sites obviously increases with the number of nodes and transistors. Defect simulation can be very challenging from a computational viewpoint, particularly if the tests to be evaluated involve long transient simulation times. If we center our attention only on the possible CMOS transistor defects the relation is straightforward. An approximation for 3 terminal models considers 3 different shorts (Gate-Source, Gate-Drain and Source-Drain) and 3 opens (Source, Gate and Drain). There are thus 6 different defects per transistor and ICs with analog transistor counts in the range of 5000 (a medium complexity) would thus lead to 30000 potential defects. Performing an exhaustive simulation of these candidates in not affordable in practice.

Solutions are thus required to evaluate (validate or discard) test strategies as fast as possible.

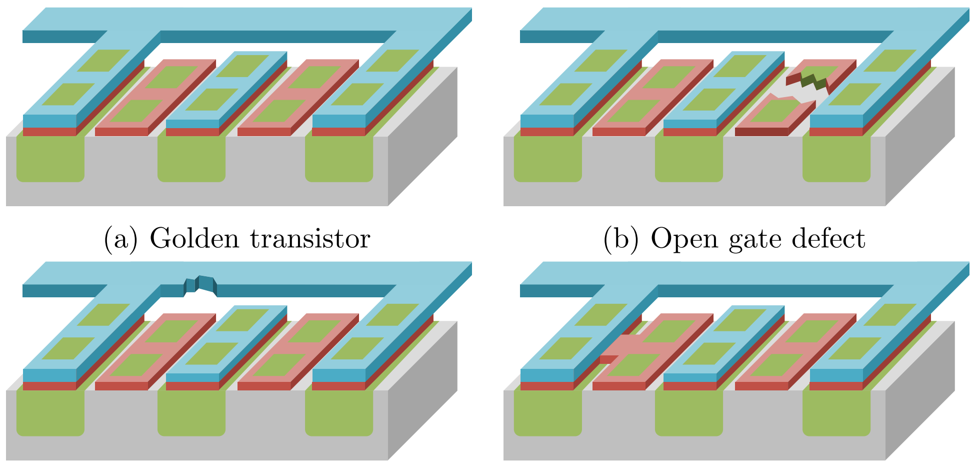
1.3 Clarification on vocabulary

With respect to the terminology, there are two terms that need to be clearly defined: Defect and Fault. In the test bibliography, these two terms are sometimes used indistinctly and this could lead to important conceptual errors in the rest of this manuscript.

Using [7] as reference, we can define a Defect as a lack or excess of material in any mask used during the fabrication process of an IC. In Fig. 1.1 we have illustrated some examples of defects. Starting from a non defective transistor (fig. 1.1a) we can add or remove material of any layer. For example, in Fig. 1.1b we removed some poly from the gate, generating an open on one of the fingers gate. On the other hand, Fig. 1.1c shows a lack of metal in the horizontal net that, a priori, would not produce any malfunction. Finally, Fig. 1.1d represents a short-circuit between one finger gate and the transistor drain/source. It is crucial to understand that Defects are only manifested in the physical layer of the IC. This is usually a well-accepted definition.

With respect to faults, there are two interpretations that are most of the time not explicitly stated in the bibliography.

The most generic definition would be that a fault is any defect that has an electrical impact of some sort on the behavior of the circuit. In other words, the fault-free and the defective circuits are not electrically equivalent.



(a) Golden transistor

(b) Open gate defect

(c) Insignificant missing metal

(d) Short circuit between terminals

Figure 1.1: Examples of possible defects on a two-finger transistor

Such a definition of a fault is most usual in the papers that deal with defect modeling and simulation, which by the way is most of the time called fault-modeling and fault-simulation. Indeed, defects as defined above do not possess intrinsic electrical characteristics and evaluating their impact on a given circuit thus requires a process of electrical modeling. For instance, a short-circuit can be modeled by the inclusion of a extra resistor with a given (parametrized) short resistance. However, such a definition entails a major drawback: the level of detail of defect modeling is always ad-hoc. For instance, the missing material in Fig. 1.1c could be considered irrelevant (and the defect would thus be discarded) or eventually modeled as a slight increase in the resistance of the metal line.

In most papers that deal with alternative test proposals, a fault is usually defined as a defect that cause the test to fail. This looks like a clear definition but it is indeed relative to the test under consideration. Since functional test is still the golden standard in the industry (despite the above-mentioned issue) the best definition of a fault is thus with respect to the functional test of the circuit. In what follows, we will thus call a fault any defect which brings the circuit out of one or more of its specifications.

For example, as we will see later, if we have a buffer with a 80dB specification for the Total Harmonic Distortion, any chip that obtains a lower value will be considered faulty. If the design has been properly centered (we thus exclude parametric yield losses), this faulty behavior is necessarily produced by a defect inside the circuit. However, not all defects have to produce a fault.

In order to clarify concepts, we will consider the Defects showed in Fig. 1.1. At first sight, we could affirm that 1.1c will not produce a fault itself, since there is not a complete open in that terminal. But if we consider the electromigration effect on that net we can not ensure that the circuit will work properly as long as a non defective one. This defect, that right now can not be considered a fault, could become one in the future.

Defect in 1.1b can be expected to affect performance, but not necessarily cause a fault since it only affects one of the fingers. The electrical effects could be treated as a parametric variation and, maybe, the result of the test would comply the specification. As before, this does not mean that the service life of the circuit is the same.

Finally, 1.1d is a short-circuit between the gate and the source or drain. This is clearly bad, since all the finger gates are usually connected together. This defect will most probably affect the behavior and cause a fault. From now on, we will maintain this nomenclature, even though the references do not mention it in the same way.

1.4 State-of-the-Art

It is not till the 80s that industry focused on DoT for AMS-RF circuits. Even though many research groups where working on the field, the biggest changes were made back then. During these years the main issues to be resolved ranged from the definition of the models to the simulation process.

1.4.1 Defect extraction

One of the solutions proposed for defects extraction was the Inductive Fault Analysis (IFA) process. The first mention of this where in [7]. The IFA procedure involves, basically, three major steps. First defects are generated base on statistical data obtained from the fabrication process. Second, the defect behavior is analyzed and the a circuit level model is produced. Finally, defects are classified in types and ranked according to their likelihood of occurrence.

Different groups [2,6,7] have quoted that considering all defects as equally probable is a poor approximation of reality that may induce a bias in the estimate of test quality metrics. For that reason it is needed to take into account the relative likelihood of the different defects RL, that is to say, the relative probability of occurrence of each defect. Intrinsically to this definition it is the fact that the sum of the RL of each defect is 1 (eq. 1.1), being N the total number of possible defects in the circuit.

$$\sum_{i}^{N} RL_i = 1 \tag{1.1}$$

This parameter is quite difficult to calculate. It is impossible to reach a realistic set of values until the layout of the circuit is defined, and normally the layout is done at the end of the design process. Defect insertion should ideally be done at layout level either following the flow of IFA [7, 8] which randomly inserts disks of missing or extra material according to the dust particle distribution statistics provided by the fab, or resorting to the proxy that consists in observing that the probability of occurrence of a short between two nodes is proportional to the parasitic capacitor between these two nodes while the probability of an open is proportional to the parasitic resistor. On an extracted netlist, the list of parasitics provides a list of potential defect sites together with their relative likelihood.

Unfortunately, test strategies must more often than not be evaluated before the layout is available (particularly if DfT methods are not considered as an afterthought). As a result, many defect simulation campaigns are carried out at schematic level. In such cases, the 6-resistor model is often considered [2, 9].

Another way to approach defect-extraction is based on the Monte-Carlo method [10]. In this method, random defects are sprinkled over the layout first. Their number varies from a few thousand to several millions. These defects can be divided into a number of groups and the amount of defects fitting into one group determines the chance a certain type of occurrence. After performing transistor-level simulations the parametric defect model is verifiable. As a result of the algorithm the parametric fault model is a large set of functional circuit instances that range from nominal to marginally functional. Therefore, the algorithm outputs the necessary data to estimate fault coverage and yield coverage metrics of a test suite.

Another approach which leads to the same results is based on the critical area principle [11]. This method determines the probability of a certain defect to occur deterministically. The critical area is defined as the layout area that is susceptible to defects. If the central point of a particle lies in this area, the defect will cause a fault in the circuit.

As a a summary, IFA uses the layout as a defect probability extractor. With it, we obtain the total number of possible defect candidates (and their associated likelihoods). The simulator afterwards decides what to simulate depending on it. Once the simulation is done, the evaluation of the test starts.

1.4.2 Defect modeling

At this point, we keep having a nomenclature problem. In the digital field the difference between fault and defect can be kept the same but they use fault models instead of the defects ones. That is, they develop fault models of a defective gate and check the performance of the whole circuit. One example of this is the stack-at fault. This defect affects the state of logic signals, either internal or external, in a logic circuit. It transforms the correct value on the defective signal line to a constant logic value, either a logic 0 or a logic 1, referred to as stuck-at-0 (SA0) or stuck-at-1 (SA1), respectively [12]. If we consider this fault we only have to put a net, or part of a net, with a fixed value. Here, the difference is clear, the fault refers to the bad performance of the affected gate and the defect could be anything that produce that fault, a short with power or ground, for example.

Defects have very complex physical characteristics and may be significantly technology-dependent [13]. The description of the physical properties encloses the relationship between process-induced defects and its effect, which allows the definition of the models. Bubbles, falling particles, flakes or irregular dips belong all to the same group are some of the well known spot defects.

In the analog field we can not set this differentiation with our definition. In [14] is given a very good sum up of the types of spot defects we can consider for an IC. These are classified in 11 groups and they involve the mistakes in any mask used during the fabrication process [7]. As mentioned in [15], most of the spot defects produced in a IC can be modeled as open or shorts. We will insert different open or shorts in the circuit but, a priori, we do not know the behavior of that defect. That is why we refer to Defect models instead of Fault models.

At the IFA process, shorts and opens are inserted in the layout and then the netlist is extracted in order to simulate but as mention before, this is not always possible, so that there are equivalent models attending to the transistors. The most extended defect model is the 6-resistor one. It consist of a possible open in each transistor pin and a short-circuit between each of them. Fig. 1.2 exemplifies that model. On the non-defective transistor each transistor has an ideal value of infinity and zero depending on if the resistor model is a short or an open, respectively. These models are good as a first approximation but there may be convergence problems between SPICE and the gate open model [9, 16, 17]. However, this point will be dealt with later in Chapter 4.

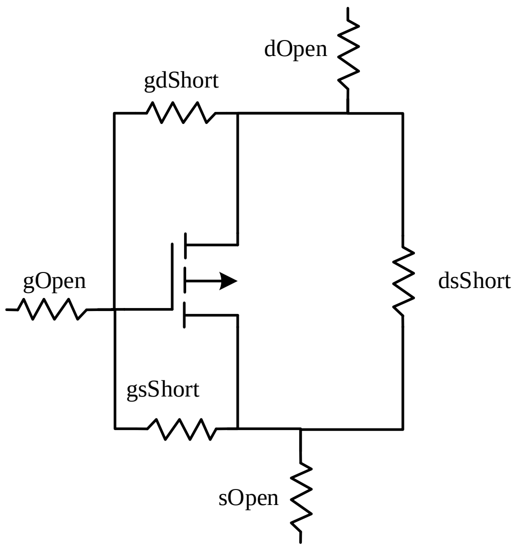


Figure 1.2: 6 transistor fault model

1.4.3 Defect simulation

Nowadays, there is only two commercial fault simulator for analog and mixed signal circuits: Tessent® DefectSim [18], from Mentor®.

Tessent® DefectSim datasheet gives a brief explanation of the program capabilities. It is able to simulate any combination of Spectre, SPICE, and HDL models, analog/digital circuitry, pre/post-layout sub-circuit netlists. The defect injection can be made from its library of shorts, opens and parametric variabilities or from a custom model defined by the user. At the end of the simulation, the program also outputs an estimated metric of the parametric yield and defect coverage of the test. Automotive industry is one the of most demanding non-defective circuits so it is not a surprise that the program generate the metrics for the ISO-26262 too.

There is another software tool called DOTSS (Defect-Oriented Test Simulation System) from NXP® which van be used to perform the simulations [10, 19, 20] but technical details cannot by found in the open litterature. The most interesting part is how it fills the Design Database. It extracts the possible defects from the layout and adapt the effect to the transistor ports to generate a netlist with the fault. Unfortunately, this tool is sold with the fabrication kit.

On the academic side, by contrast, there have been significantly more contributions. It has already been said that the number of possible defects make unmanageable an exhaustive simulation. That is why some of the first efforts center the attention in simulation time reduction. One example of this can be found in [21]. They directly use the IFA flow extracting the defects from the layout by the critical area process. Every defect is weighted using the RL in order to obtain the most significance ones. The list is reduced to the group of most likely defects.

Another method is to simplify the number of nodes. It is trivial to think if the circuit is simpler the simulation will run faster. That is what is done in [22]. They convert non-linear analog circuits into Boolean models, faster to simulate. This method was used in [23] during the simulation process. They only simulated SPICE level model if the block affected by the defect, letting the rest of the circuit as an ideal behavior.

Very close to the previous approximation is in [24] but here they go to a lower level of abstraction. They use a special model of stuck-at fault that affects not the signal path, but the transistors' logic gate. That is the stuck-open and stuck-short. This model substitute the transistor by a unique resistor between the drain and source pin, letting the gate floating. Obviously, this simulation must be done at SPICE level netlist to be able to consider resistors but they only substitute the SPICE model in the defective transistor. The rest of the circuit is in HDL language. Doing this a mixed-signal simulation must be performed and, basing on their results, is remarkably faster than the complete SPICE netlist simulation.

In [25], during the defect extraction process, the authors generate connection schemes between the different sub-circuits that compose the circuit. By doing this, they can isolate independent sections of the chip that does not interact between each other, allowing a reliable divide and conquer approach.

The direct approach to increase simulation speed when we have a big amount of them is to run them in parallel. This method, even though it could be reducibly obvious, can not always be applied. In [26] reached this task with a specific tool called CONCERT2, that can perform DC and transient fault simulations. The simulation speedup is achieved by sharing the intermediate results among the corresponding defective and fault-free circuits.

In another branch of research there are jobs that benefit from the fact that defects which cause significant deviations from the fault-free response are easier to detect than the others, so that simulation accuracy can be lowered. By risking having to repeat the simulation of the defects that do not have a significant effect on the circuit at the higher accurscy, they can reduce the total simulation time for those who do not. In [27] this approach is done with the DC simulation time. By only doing one-step Newton-Raphson iteration to calculate the operating point they not only reduce simulation time but speed up exact fault simulation based on simulation continuation.

A previous work implemented this same idea but trying to accelerate transient simulations. In [28] the speed up the transient analysis by relaxing and tighten the time step dynamically. In order to achieve that, the first perform a DC analysis of all the faults and classify in groups with expected equivalent behavioral. From echa group they simulate the most likely defects in parallel step by step and sharing data between each process to save unnecessary iterations.

Coming back with the first idea of reducing the number of defects to simulate appears the Sensitivity Analysis. With the adjoint network method, for example, the output sensitivities can be computed with respect to the complete defect list at the same time. Indeed, the adjoint network method allows to compute the sensitivity of one output parameter with respect to all component variations (including non-existing components) in only two simulations, one for the original network and one for the corresponding adjoint network [9, 29]. This, however, is limited to Linear Time Invariant circuits.

In [30] is presented a technique that enables the derivation of the constraint of an internal functional block with respect to the DC specification. Basing on it, they have implemented a defect simulator that reduces the computational effort by both, removing undetectable defects from the simulation list and performing the defect simulation only in the blocks affected.

In the past few years, the way to consider the problem has changed [6]. Since we can acknowledge that an accurate and faithful defect simulation is computationally too intensive, what is the best way to spend the limited computational resources? As mentioned in [21], a solution is to reduce the number of defects by limiting the evaluation to the most likely ones.

However this may not be representative of the real circuit. Very likely defects (associated to large transistors and/or global nets) may behave differently from unlikely defects (associated to small transistors and or local nets) with respect to test. And despite a significant likelihood difference between individual defects, the population-wise probability of occurrence may not be so different. Indeed, there may be very few though highly-probable defects and many small and unlikely defects.

With this consideration in mind, it was proposed in [15] to use likelihood-weighted defect sampling to obtain a statistically unbiased estimate of defect-coverage. This approach was further refined in [2], proposing a simpler selection algorithm and include the possibility of continuously parametrized defects (i.e. opens and shorts with a given resistance probability density function).

Since exhaustive defect simulation is intractable, the defect coverage obtained by proper defect population sampling is a statistical estimate and as such should be associated to a confidence interval.

In the next chapter, we will explain in more detail the path followed in this work: We will argue that evaluating only the defect coverage is insufficient, since not all defects have to lead to a failure. In order to obtain a more global vision of the test, this fact must be taken into account, as well as the importance that these failures would have in the event of not being detected. Consequently, we will provide a simulation framework to discard or validate a test strategy and optimize the required computational resources.

Chapter 2

The proposed Framework

2.1 A complete picture of test quality

Since we are in Zero Defects era [19] the exhaustive fault analysis, perfect by definition, has derived to more complex solutions. The imperfections in the process of Defect-Oriented Test has questioned its quality and an evaluation of it is needed. Generally, this problem has been solved by reporting Defect Coverage but this metric presents some problems.

Defect Coverage is defined as the amount of detected defects divided by the total number of considered defects, Eq. 2.1. Using this definition as the only option to evaluate a Defect-Oriented Test only can lead to conclusive results if the value is close to 1 or 0, allowing the validation or exclusion of the test.

$$DC = \frac{Detected \quad defects}{Simulated \quad defects} \tag{2.1}$$

$$FE = \frac{Undetected \quad faults}{Simulated \quad faults} \tag{2.2}$$

For intermediate values of the metric it is necessary more information in order to fully assess the test quality. That is why the fault analysis is included in our test. The classical expression of this metric has practically the same definition as de Defect Coverage. Fault Coverage is the ratio between the detected faults and the total amount faults. The complementary metric, Fault Escape, is defined as one minus the Fault Coverage. It is the amount of undetected faults divided by the total of faults, Eq. 2.2.

These two metrics give a more complete vision of the test quality, but there are cases where these metrics can still exclude some valuable info. That is the case when we have with equal Defect Coverage and Fault Escape.

Here, the decision between two test should be done taking into account the importance of the defects that have passed the test. Indeed, a circuit that fails to meet the specification only marginally is undoubtedly better than one with a catastrophic failure. In order to solve this problem we have provided a new FoM that pretends to solve this problem.

2.1.1 Modified Defect Coverage

The classical expression of the Defect Coverage is in Eq. (2.1). This expression does not consider the Relative Likelihood of happening of each defect. An extended proposal is that of [15]. It consists of substituting the numerator and denominator by the sum of the RL weights of each defect, Eq. (2.3).

$$DC^{*} = \frac{\sum_{i}^{N} D_i \cdot RL_i}{\sum_{i}^{N} RL_i} \tag{2.3}$$

Where D_i switches its value between 1 and 0 if the defect i is detected or not, RL is the array with all the weights and N the total number of considered defects. This metric has resulted to be useful for the non-exhaustive defect simulation, as it will be studied in chapter 4.

2.1.2 Modified Fault Escape

It has already been discussed that perform an exhaustive fault test is not affordable due to the amount of possible defects in an IC. In addition, fault simulations are presumably much longer to simulate than a defect simulation. For the sake of brevity, we call fault simulation the simulation of a defect to assess its impact on performance. It is thus using a functional test setup. Conversely, we call defect simulation the simulation of the defect in the Defect-Oriented Test setup. In order to lower computational burden, we propose to simulate the performance of only those defects that have not been detected by the test. Indeed, we do not really care if the detected defects produce a fault or not since a defect is always a threat and should be discarded so no interesting information is lost.

But the classical metrics cannot be computed since the denominator would be a biased approximation. To clarify this problem we only have to observe two tests. One of them lets go of a failure and the other a fault and a defect. In the first case we would have a Fault Escape of 100% and in the second a 50%. Essentially, both test are equally bad with regard to fault escape but the metrics are different. For that reason we proposed to reference the metric not to the total number of simulated escaped defects but to the total number of defects, as with the Defect Coverage.

$$FE* = \frac{Undetected \quad faults}{Simulated \quad defects} \tag{2.4}$$

The basic approximation is in Eq. (2.4). It has the same form as the original Defect Coverage, but now the numerator only have the undetected faults. As before, this metric can be reshaped for adding the RL, Eq. (2.5).

$$FE^{*} = \frac{\sum_i^N F_i \cdot RL_i}{\sum_i^N RL_i} \tag{2.5}$$

Here F_i is set to 1 if the defect has passed the test and it is a fault, it has a null value in any other case, if it has not passed the test or it is not a fault.

2.1.3 Severity

Severity is a metric that pretends to summarize how far are the undetected faults from the specification. The problematic with this metric is that it has to be referenced to the test, and not to the faults, since the exhaustive simulation has not been performed, Eq. (2.6).

$$Severity = \frac{\sqrt{\sum_i^N F_i \cdot RL_i \left(FV_i - Spec\right)^2}}{\sum_i^N RL_i} \tag{2.6}$$

FV is an array with the performance value measured and $Spec$ the specification value. The expression is justifiable if we understand the objective of it.

- With $(FV_i - Spec)$ we measure the distance to the specification.
- F_i is the same array as before. It values 1 if it correspond to a escaped fault and 0 in any other case.
- The sum of the squares, $\sum_i^{Ne} (...)^2$, ensure that every fault will increase the metric.
- The root allows to resize the value to units of the specification.

- By dividing by the total of defect likelihood, $\sum_i^N RL_i$,we normalize the value for comparison with other tests.

However, this metric is quite unintuitive so it only can be used for test comparison, not for absolute results of how good is the test. We're working on another version of it.

2.2 Sequential defect simulation with early stopping

The approach to carry out the simulation using these metrics is a sequential simulation of defects. The idea is setting Defect Coverage and the Fault Escape target and perform simulations till it is possible to ensure that the test reaches or not these values. At first sight this could derive in an exhaustive test, so some considerations must be taken into account.

For the defect selection we will use the "Likelihood-sampling adaptive defect simulation" [2]. This method consists of a random defect selection with the Relative Likelihood as weight. Every time a defect is selected refreshes an array that contain the number of occurrences of each one. On this way, the same defect can be selected more than one time, but the simulation is done only once. Once the simulation is finished, that array is normalized and it contains the likelihood of each defect that has produced the simulation.

Metrics, in turn, are calculated after every simulation. The first problem of not doing an exhaustive test is that metrics confidence is in question. This makes that every metric must be calculated and showed with a level of uncertainty. The process of metrics calculation from subpopulations, what is called estimation, has been widely explored in other research fields and the Confidence Interval calculus is widely extended.

Fig. 2.1 reflects the flow diagram developed. In order to explain it we are going to assume that we have already obtained a list of defects that are saved in the `listOfDefects` variable. Every defect listed in that array will have an ideal Relative Likelihood associated in the corresponding position of the `RLideal` variable. These two variables are considered global for allowing any function to access them. Apart from that, the rest of variables initialization has been separated in four different blocks to clarify the meaning of them.

- First block corresponds to the user input variables:
  - `desiredDC` and `desiredFE` correspond with the desired Defect Coverage and Fault Escape, respectively.

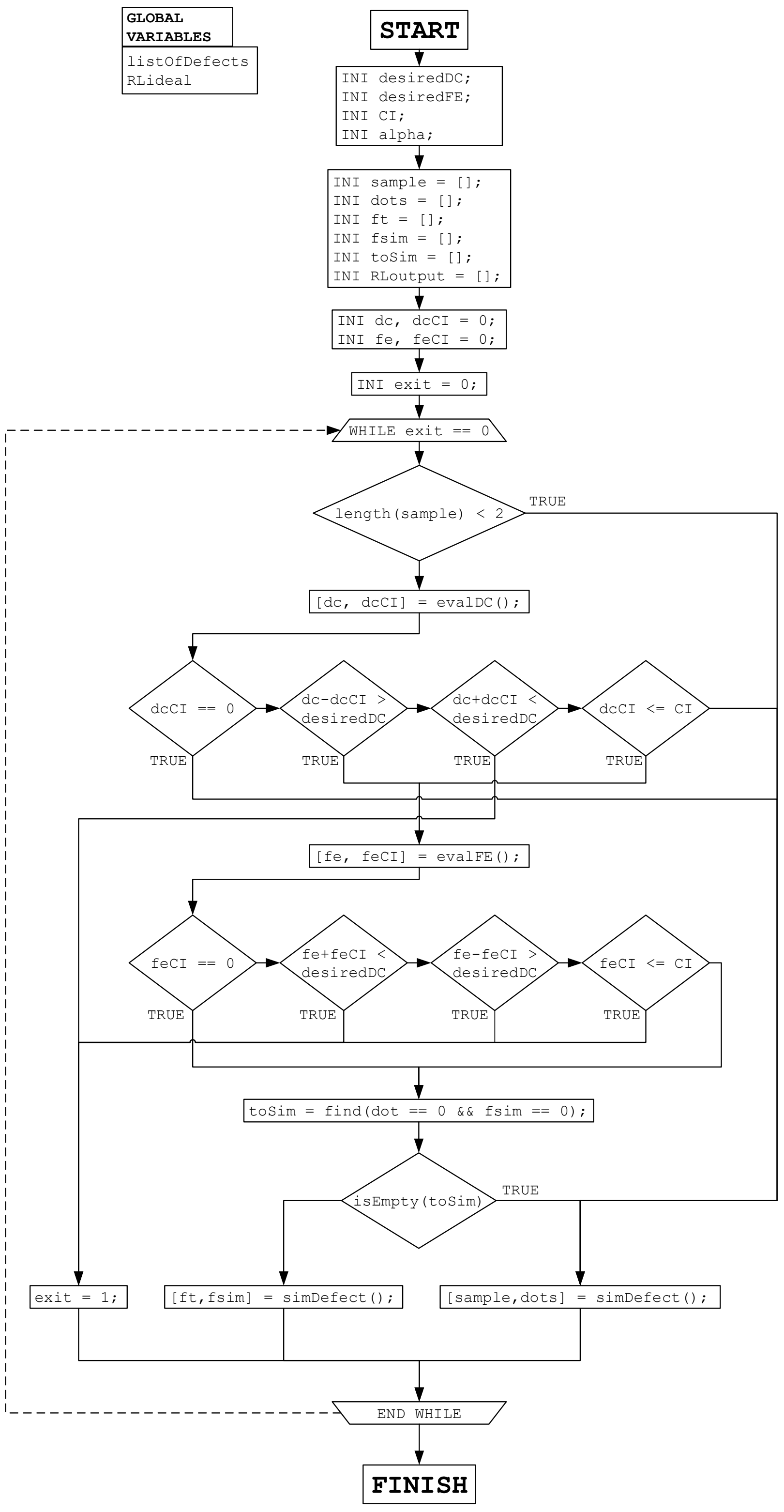


Figure 2.1: Sequential simulation flow diagram.

  - `CI` is the absolute value of the Confidence Interval width. It is used to stop the simulation in the case the early stopping can not be done.
  - `alpha` determine the reliability of the calculated results.

- Second block corresponds to the simulation loop variables. In the arrays are declared the simulation history.
  - `sample` contains the defect index of the selected defects. For every defect simulation, a defect is selected following the likelihood-sampling. If the selected defect index is not in the array it is pushed at the end, if not, this arrays is not updated. So that, the length of the array will correspond with the total number of different defects simulated.
  - `dots` saves the results of the Defect-Oriented Test simulated. For every defect on `sample` saves a 1 if the defect has been detected and a 0 otherwise. Every time a defect is pushed in `sample` a 0 is pushed in this array at the same position. On this way, the detection info saved in `dots(i)` correspond with both the `i`-th defect simulated and the `sample(i)` index of the list of defects.
  - `ft` the equivalent to the previous one but for the fault simulation. Maintaining the reference, we push a 0 at the end with every new defect selected. In the case the fault simulation of the `sample(i)` defect is performed, the `ft(i)` is updated to 1 if the defect `sample(i)` is a fault.
  - `fsim` contains the list of performed fault simulations. Like `ft`, a 0 is pushed with every new defect selected but this array update the `fsim(i)` value if the fault simulation of the `sample(i)` is performed, independently of the result. It will be useful to select the non-simulated escapes from the sample, as we will see soon. With this definition, the sum of all the elements of this array will be the number of defects used for the Fault Escape calculation.
  - `RLoutput` contains the number of times a defect has been selected. Intuitively, a 1 is pushed at the end with every new defect and every time the `sample(i)`-th defect is selected, it increments the `i`-th position by one.

- The third initialization contains the test evaluation itself. The estimated Defect Coverage and Fault Escape, `dc` and `fe`, with its respective Confidence Interval, `dcCI` and `feCI`. The last block only contains a variable to stop the loop.

Once the variables are correctly initialized the simulation loop starts and maintain looping until the variable `exit` is not null. For clarity, some variables have been omitted from the functions call but the will appear in the explanation of each of them.

After loop initialization, two different defects are simulated, if this is not done there are none statistics to calculate. `simDefect()` function, Code 2.1, first perform the likelihood-sampling for defect selection from the `listOfDefects` variable with the `RLideal` as weight. The next step is to find out if the defect selected has been already simulated or not. If it has been simulated we update `RLoutput`. If not, we update all the arrays. The `performDOTSimulation`() function performs the injection of the defect in the circuit netlist and simulate the Defect-Oriented Test that is being evaluated. The output of the function will be a 1 if the defect is detected and 0 if not and `dots` is updated accordingly.

```
function [sample,dots,RLoutput,ft,fsim] = simDefect(sample,dots,
    RLoutput,ft,fsim)

  defect = randsample(1:length(listOfDefects),1,true,RLideal);

  if ismember(defect, sample)
    RLoutput(find(sample == defect)) = ...
    RLoutput(find(sample == defect)) + 1;
  else
    sample(end+1) = defect;
    RLoutput(end+1) = 1;
    dots(end+1) = performDOTSimulation(defect);
    ft(end+1) = 0;
    fsim(end+1) = 0;
  end

end
```

Code 2.1: `simDefect()` function Matlab® code

After two simulations the first estimator can be calculated. The Defect Coverage estimation and its Confidence Interval are obtained from the function `evalDC()`, Code 2.2. In the code exposed only the estimator calculation has been included. The acquisition of the `alpha` level of accuracy of the metric will be explained in next section. We will only clarify that the Confidence Interval estimates the error in the measurement done due to the calculation from a subpopulation. From it, we obtain a upper and lower limit. The result returned from the `CIcalc`() function is the distance to the center of that interval. In this way, we can ensure that the metric is its value plus-minus the `CI` value with an `alpha`-level of confidence.

```
function [dc,dcCI] = evalDC(dots,RLoutput,alpha)

  RL = RLoutput/sum(RLoutput);
  dc = sum(dots.*RL)/sum(RL);

  dcCI = CIcalc(alpha,dots,RL);

end
```

Code 2.2: `evalDC()` function Matlab® code

The Confidence Interval will be useful for the early stopping. That is the objective of the four `if`s statements in cascade. The first check if the Confidence Interval is valid or not. This could happen if the sample does not have enough diversity. As we will see, this is quite usual for the first few iterations. The second checks if the lower limit is higher than the desired value. If this is the case, we can ensure the test will be accepted so we can start to simulate escapes. Third block checks if the test can be rejected, that is, is the upper limit of the Confidence Interval is lower than the desired value. The last one will stop simulating defects in the case the Confidence Interval is narrow enough.

```
function [fe,feCI] = evalFE(ft,ftsim,dots,RLoutput,alpha)

  RL = RLoutput/sum(RLoutput);
  simF = find(fsim==1);

  fe = sum(ft(simF).*not(dots(simF)).*RL(simF))/sum(RL);

  feCI = CIcalc(ft,ftsim,dots,RLoutput,alpha);

end
```

Code 2.3: `evalFE()` function Matlab® code

If the decision taken is to perform a fault simulation, Fault Escape is estimated first (`evalFE()`, Code 2.3). With this, we follow the same comparison as before, but with some different exits. The first one is the same, if cannot trust the Confidence Interval we ask for another fault simulation. Otherwise, we check the values. With this three conditionals we finish the loop whether or not we accept the test or if the Confidence Interval is narrow enough.

`simFault()` function, Code 2.4, performs the fault simulation. It reads the samples in the same order they have been selected until an undetected defect that has not been simulated yet is found. In this way, the same simulation method used for defect selection is virtually repeated for the fault simulation and the metric calculations remains the same.

```
function [ft,fsim] = simFault(sample,dots,RLoutput,ft,fsim)

  noSim = find(fsim == 0);

  for i = 1:length(noSim)
    if dots(noSim(1)) == 1
      fsim(noSim(i)) = 1;
    else
      ft = performFTSimulation(sample(noSim(i)));
      fsim(noSim(i)) = 1;
      break;
    end
  end

end
```

Code 2.4: `simFault()` function Matlab® code

The fault simulations have an especial consideration, they are only performed if there are escapes that has not been simulated yet. So that, we first look for among the escapes, that is `dots==0`, the defects that have not been simulated, `fsim==0`. If the array is empty we perform a defect simulation first to have another try. This loop is done this way do to the fact that we could accept a test that meets Defect Coverage objective but the number of samples is insufficient to ensure that the Fault Escape is low enough.

The gains in computation time comes from two fronts:

- The early stopping criterion, which allows validating or discarding the test as soon as the metrics estimates are confident enough.
- The fact that we simulate for performance only the escapes of the test instead of all the defects. Considering that performance test setups usually require much more longer simulation time than defect oriented ones, this can account for important savings.

Chapter 3

Statistical evaluation of test metrics

3.1 Likelihood random sampling

In order to solve the problem of the quantity of possible defects to simulate statistical test is performed. Statistics is the science of collecting, organizing, and interpreting experimental data to draw conclusions about unknown populations observed through experiment [5]. This approach can be directly applied test evaluation for reducing simulation effort.

By following the IFA process we obtain the already mentioned Relative Likelihood. As mentioned before, this metric give us information about how likely is a defect to happen relative to the rest. It is rational to think that the mentioned metrics in chapter 2 will be more affected by the defects with high RL than any other, so this defects may be the most important to simulate.

In [2] this fact is discussed. They present the discussion between using Likelihood random sampling or simulating the most likely defects. In essence, the second group argue that since spot defects are not part of any stochastic process, the inference of any defect property can not be done taking into account another defect. Talking about defect coverage, the fact that one defect is detected gives absolutely no clue if another one will be detected or not.

On the other hand, pro-Likelihood Random Sampling group claims that in a complex mixed-signal circuit there can be a majority of defects with low likelihood and fewer defects with large likelihood. The global probability of occurrence of these two populations may be similar. If there could be a reason for which the test coverage may be dependent on likelihood, simulating only the most likely defects would induce a bias.

Such a bias is what they obtain in one of the scenarios they use to demonstrate the point. By introducing a strong imbalance in the fault coverage dependent on the likelihood of defect, the simulation of most likely defects produces a very important bias, what it is not surprising since the test was designed to produce that bias. What appears to be more interesting is that the confidence interval of likelihood sampling is almost equal to the confidence interval of the most likely simulation. This means that, even in the hypothesis of independence between likelihood and coverage, there is no significant penalty in terms of computational effort to perform likelihood random sampling instead of simulating only the most significant.

Likelihood random sampling was firstly proposed in [15] but it was refined in [2]. The process of defect selection and simulation can be sum up in the next items.

- Once the Relative Likelihood has been extracted from the circuit netlist, defects are selected randomly using it, what means that a high probable defect will be more likely selected, even more than one time.
- There is substitution in sampling, what is, the same defect can be selected more than one time.
- During the selection, an array, that we have called Output Relative Likelihood, is filled with a start value of 1 for every selected defect.
- If a defect is selected again, it increments its value by one in the Output Relative Likelihood, but it is not simulated again.
- The number of defects in the sample it is not defined by the total number of simulations but for the number of deferent defects selected.

It can be demonstrated that the Output RL array will be the same as the original RL is the number of simulations tends to infinity, but this is not necessary. The Output RL can be used to calculate the defects even though it has not the same values. The effect of the overrated defects is compensated by the overrated value of the less significant ones.

$$dc^* = \frac{\sum_i^n d_i \cdot RLo_i}{\sum_i^n RLo_i} \tag{3.1}$$

$$fe^* = \frac{\sum_i^n f_i \cdot RLo_i}{\sum_i^n RLo_i} \tag{3.2}$$

Equations (3.1) and (3.2) shows the estimators of the Defect Coverage and Fault Escape with the obtained output Relative Likelihood, RLo. The arrays d_i and f_i are the same as the originals but with the subpopulations, that is why the sum finish at n instead of N. With this process a faster convergence of the metrics is reached, what implies a direct reduction on simulation costs, but something more is necessary. By estimating the measurement value with a subpopulation of the defects a Confidence Interval of that result must be given, and this is what we talk in next section.

3.2 Estimate of confidence intervals

In general, a ratio is the division of two magnitudes. In the statistical bibliography, specifically in [31], the ratio is defined as the proportional relationship between two means within the same population. What is the same, if we have a population N and two subpopulations of N called Y and X we define the ratio R as

$$R = \frac{Y}{X} = \frac{\overline{Y}}{\overline{X}} \tag{3.3}$$

A trivial estimator of this ratio can be the showed in equation (3.4). From now on, we will use uppercase letters when we are talking about a magnitude referring to the whole population and lowercase letters when we are talking about an estimate. On this way, the mean $\bar{y}$ is the mean of the same magnitude as $\bar{Y}$ but measured in the subpopulation n instead of N.

$$r = \frac{y}{x} = \frac{\bar{y}}{\bar{x}} \tag{3.4}$$

The statistical problem comes from determining how much we can rely on the estimation. For that, we first need to determine the sampling method. A widely extended sampling method is the Simple Random Sampling. It consist in the randomly selection of sampling discarding the already selected. On this way, we can determine the Mean Square Error as equation (3.5).

$$MSE(R) = \sigma^2_{R,1} = \frac{1 - FPC}{n \cdot \overline{X}^2} \frac{\sum_i^N (Y_i - R \cdot X_i)^2}{N - 1} \tag{3.5}$$

With $FPC = n/N$, called the Finite Population Correction Ratio. But the problem of evaluating the Confidence Interval (CI) of the ratios comes since, a priori, we do not know the rue value of the population parameters. For that reason, the calculus should be treated carefully. For example, since our metrics can only reach values between 0 and 1 they can be treated as a binomial distribution. In this specific case, the confidence interval can be as easy as (3.6), [32].

$$CI_{binomial} = z_{\alpha/2} \cdot s_{R,binomial} + \frac{1}{2 \cdot n} \tag{3.6}$$

$$s^2_{R,binomial} = (1 - FPC)\,\frac{r(1-r)}{n-1} \tag{3.7}$$

The term s is the estimator of σ. Only on this case, the ratio variance can be reduced to (3.7). The term $\frac{1}{2 \cdot n}$ is the correction for continuity. The problem of this approximation is that it is unreliable for the limits values 0 or 1. So that, it is needed a CI that contemplates that problem, even though generates a bigger interval.

The solution part of considering as correlated numerator and denominator. On this way, the calculus of the confidence interval is reduced to calculate the variances and covariances of both terms and determine the ratio CI with them. For that, it is recommended to obtain the ratio variance from the means variance. This can be done from (3.5), as showed in [31], resulting in equation (3.8)

$$\sigma^2_{R,2} = \frac{1 - FPC}{n \cdot \overline{X}} \left(\sigma^2_{\overline{Y}} + R^2 \cdot \sigma^2_{\overline{X}} - 2 \cdot R \cdot \sigma_{\overline{YX}}\right) \tag{3.8}$$

$\sigma^2_{R,2}$ is the variance of the ratio in expresión of the variances of its terms. $\sigma^2_{\overline{Y}}$ and $\sigma^2_{\overline{X}}$, and the covariance between them, $\sigma_{\overline{YX}}$. But this values are unreachable with an exhaustive test, so that an estimator of the variance is used. This results in calculate the variances as follows:

$$s^2_{\overline{y}} = \frac{\sum_i^N (y_i - \overline{y})^2}{n-1} \tag{3.9}$$

$$s^2_{\overline{x}} = \frac{\sum_i^N (x_i - \overline{x})^2}{n-1} \tag{3.10}$$

$$s^2_{\overline{yx}} = \frac{\sum_i^N (y_i - \overline{y})(x_i - \overline{x})}{n-1} \tag{3.11}$$

$$s^2_r = \frac{1 - FPC}{n \cdot \overline{x}} \left(s^2_{\overline{y}} + r^2 \cdot s^2_{\overline{x}} - 2 \cdot r \cdot s_{\overline{yx}}\right) \tag{3.12}$$

It is important to differentiate that our intention is not to estimate $E(Y/X)$ but $E(Y)/E(X)$. In fact, if X and Y are normally distributed, the first quantity does not even exist [33]. A widely extended CI calculation is by the Fieller's theorem [34, 35], that obtains the limits of the range expressed in (3.13).

$$(\alpha_L, \alpha_U) = \frac{1}{1-g}\left[r - \frac{g \cdot c_{\overline{yx}}}{s_{\overline{x}}^2} \mp \frac{t_{d,\alpha} \cdot s_r}{\overline{X}}\sqrt{\Gamma}\right] \tag{3.13}$$

$$\Gamma = s_{\overline{y}}^2 - 2 \cdot r \cdot c_{\overline{yx}} + r^2 \cdot s_{\overline{x}}^2 - g\left(s_{\overline{y}}^2 - \frac{c_{\overline{yx}}^2}{s_{\overline{x}}^2}\right) \tag{3.14}$$

$$g = \frac{t_{d,\alpha}^2 \cdot s_{\overline{y}}^2}{\overline{X}^2} \tag{3.15}$$

The term $t_{d,\alpha}$ refers to the α-level deviate from the Student's t-distribution based on d degrees of freedom, with $d = n - 1$. The use of the Student's t-distribution approximation instead of the Normal distribution is due to the possibility of not having enough samples. Anyways, the difference from 50 samples or more is insignificant.

In order to use this expression, it is necessary that both means of the ratio follow a bivariate normal distribution. The second condition is that Γ, eq. (3.14), need to be positive so as not to obtain an imaginary value. In addition to that, the parameter g allows to determine the meaning of the range. When g is very close to 1, the confidence interval is infinite and if it is greater than 1, the overall divisor outside the square brackets is negative and the confidence interval is exclusive.

Even though Fieller's theorem is the most extended method we will see it does not work correctly with values close to 0, and the problem of continuity is not solved with it. In order to decide which equation use to perform the analysis we have prepared a virtual environment.

3.2.1 Checking confidence intervals

We will perform the analysis of a circuit with 9000 transistors. Each of them with 10 different possible defects. That makes a total of 90000 different simulations. The RL is randomly assigned making more weighted some of them. The histogram of the array through the form showed in Fig. 3.1.

From the full population, we will set a Defect Coverage by randomly selecting defects till the desired value is reached. This definition will work as the exhaustive simulation results. Our aim with this experiment is to obtain the real CI of the Defect Coverage. In order to accomplish this task we will

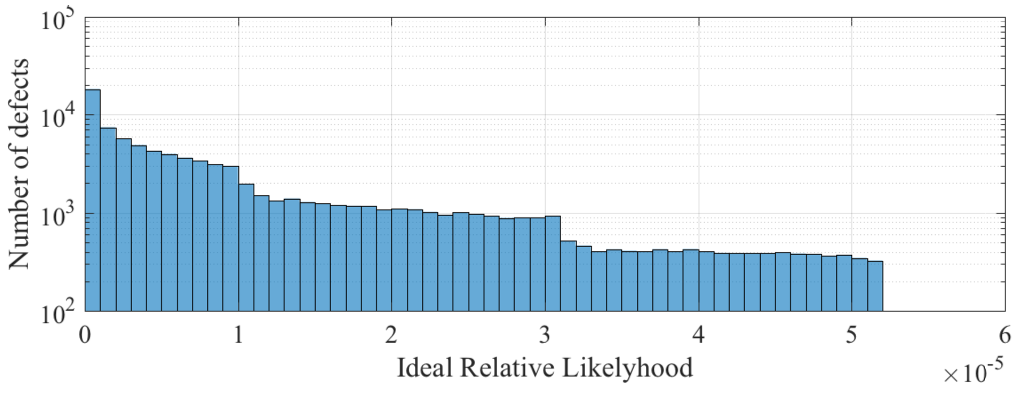


Figure 3.1: Virtual example, RL histogram

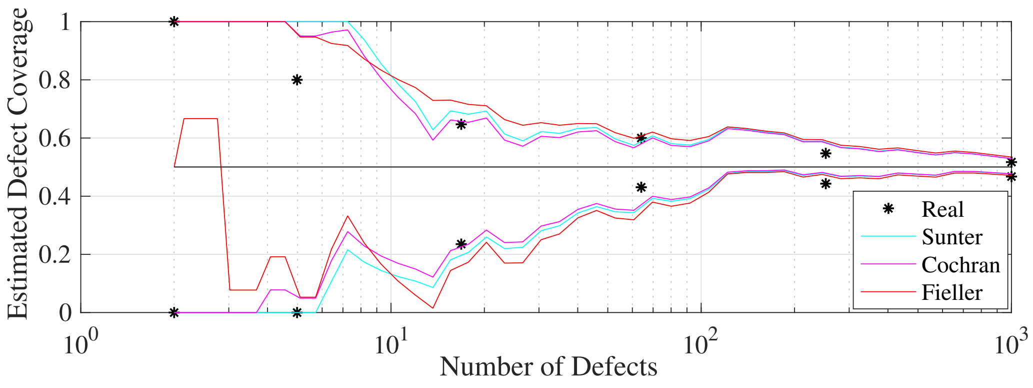


Figure 3.2: CI narrowing with the number of samples, $dc \approx 0.5$

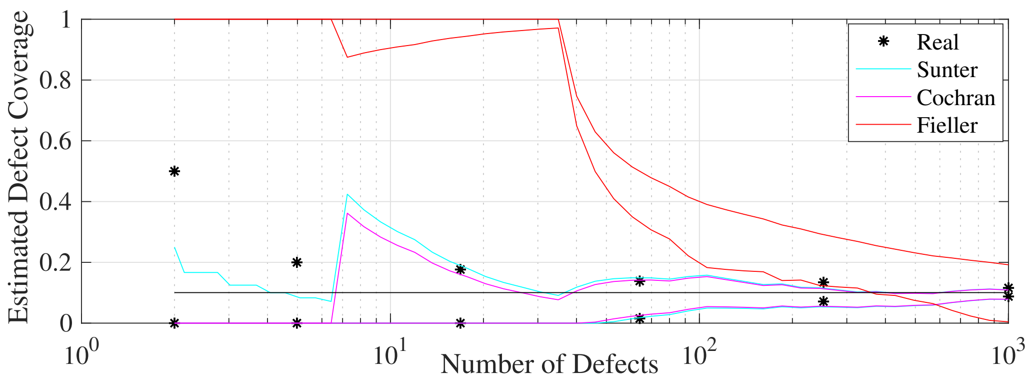


Figure 3.3: CI narrowing with the number of samples, $dc \approx 0.1$

perform a virtual sampling 100 times for different amounts of defects. Once this is done, we will have a histogram of the obtained Defect Coverage for each number of defects and we will be capable of setting a empirical Confidence Interval, the measurement error. The theoretical Confidence Interval should follow this results or, at least, not narrowing too fast.

Fig. 3.2 and 3.3 shows the results for a Defect Coverage of 0.5 and 0.1, respectively. The Sunter CI is calculated as Eq. (3.6). Cochran is calculated using the same equation but calculating the typical deviation as a ratio, Eq. (3.8). Finally, the Fieller range is calculated with Eq. (3.13).

In general, Confidence Interval are not reliable for low number of samples, at least 7 are necessary. Besides, all three seem a good approximation when Defect Coverage has an intermediate value since the real value is always inside the theoretical. Regardless, the Fieller's Theorem does not give good results with values close to zero, as expected, so this option is discarded. The difference between Sunter and Cochran depends on the continuity summand. In our case, we are going to use Cochran because the bad behavior with low number of samples is easy to detect, it only requires to compare equality between the two extremes.

Chapter 4

Description of the developed tools

4.1 Defect models

It may not be clear the name of fault models when what we want to perform is a defect simulation. This name comes from the digital world where fault models refer to the effect that any defect could have in the circuit, for example, the stuck at 0 or stuck at 1 faults. If we consider this fault we only have to put a net, or part of a net, with a fixed value. Here the fault refers to the bad performance of a gate and the defect could be anything that produce that badfunction, a short with power or ground, for example. So in reality the stuck at 0 and stuck at 1 faults can be seen as a functional fault at gate level.

In the analog field this difference is not as clear. In [14] is given a very good sum up of the types of spot defects we can consider for an IC. These are classified in 11 groups and they involve the mistakes in any mask used during the fabrication process [7]. As mentioned in [15], most of the spot defects produced in a IC can be modeled as open or shorts. Here is the point, we will insert different open or shorts in the circuit, but the defect can be dust, bubbles or any mistake produced during the fabrication.

It is widely accepted modeling spot defects as resistances. This model is close enough in the case of shorts but with opens appears some problems. A big resistor is settled in the place of the open in order to guarantee los currents but the simulator convergence is not limited. In addition, for the transient analysis it could be interesting to add a capacitor in parallel with the resistance. This approach could be applied for the electrical modeling, independently of where it happens.

For an post-layout approximation the resistors would have to substitute

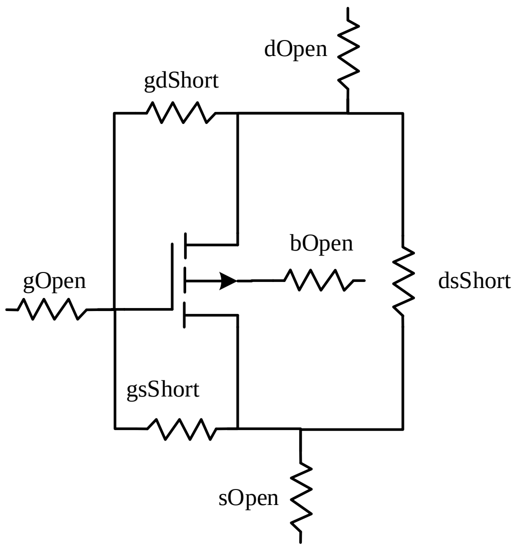


Figure 4.1: 6+1 transistor fault model

the suitable parasite. If it is an open on a net only needs to increase the parasitic resistor value. On the other hand, if it a short between two needs, the parasitic capacitor has to be replaced by a low valued resistor. However, when the injection is done directly on the nominal schematic the defects are usually injected in devices. So that, models are slightly different.

Fig. 4.1 shows the classical 6 resistor fault model. Starting from a nominal case in which the Open resistors have a ideal value of zero whereas the Shorts are set to infinity. In the case a short between any terminal is chosen to simulate the resistor is set to a determined *Rshort* value, that can be fixed or variable depending on the simulation. In the case of the opens we follow the same pattern, but with an *Ropen* parameter. In order to increase accuracy, we have added the baseOpen defect, that could happen in the case the transistor has its own well.

In addition to that, we have considered three more open for the case an only finger fails of the entire transistor. This is a typical example of defect that may affect or not the performance, since it could be considered as a parametric variation of the transistor, and not a complete failure.

The calculation of the Relative Likelihood in this case is done from the device parameters itself. To sum up, the biggest the transistor the most likely to be defective since there are more are where the dust particle can fall. But it is not accurate treating opens as likely as shorts. In fact, opens are more likely than shorts. In our case we will consider five times more. For the fingers open, since only affects to one part of the transistor we have change

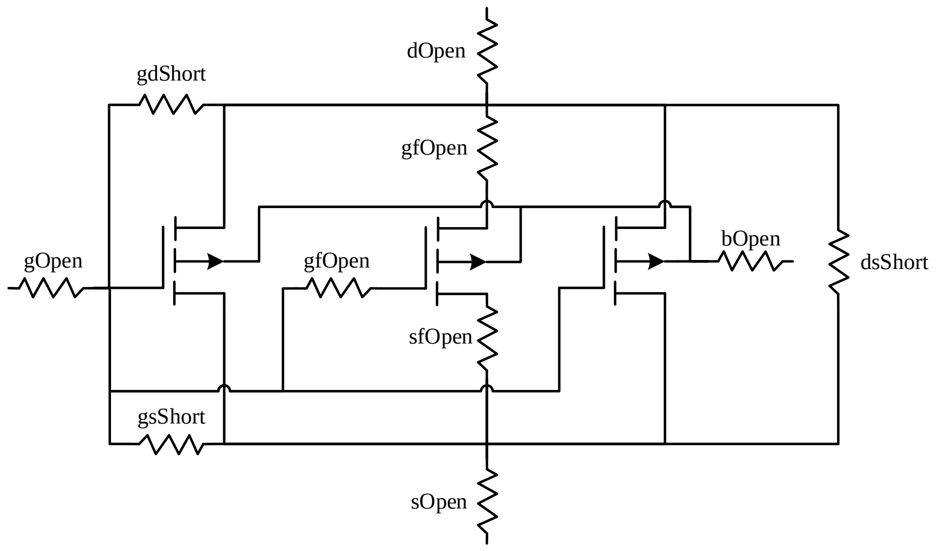


Figure 4.2: 6+1 transistor fault model with one-finger open option

the factor by three. Finally, baseOpen defect is the least likely to happen, so its value is divided by half. In the end, the Relative Likelihood array will be the area of the transistor the belong multiplied by the mentioned factors. In order to make the values relative to the rest, the vector is normalized with the sum of all elements.

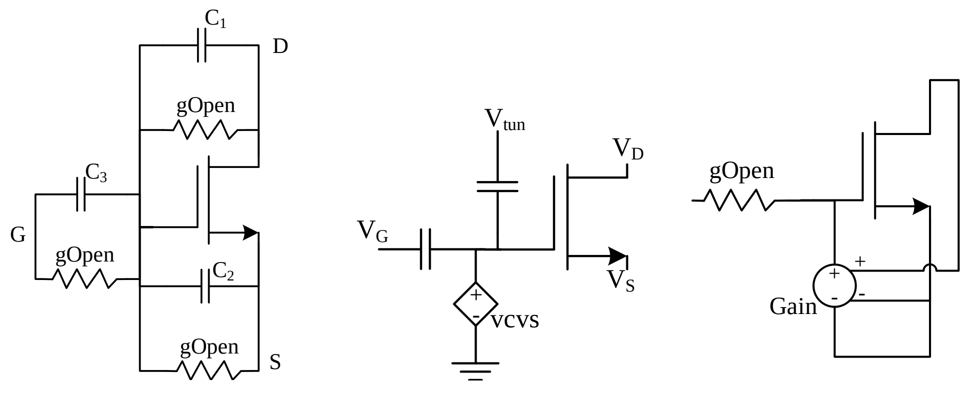
4.1.1 Convergence error with Open-Gate defect

The problem of modeling the open-gate defect by a large resistor is that the DC analysis converges at the same point with or without input resistance. For that reason some modifications of the model should be done.

There are several example of Floating Gate Transistor (FGT) models [16, 17, 36] but due to how the DC analysis with Spectre works, some of them are not implementable. Any model for which the gate voltage depends on resistor ratios will generate wrong results.

For example, in [16], Fig. 4.3a, voltage is induced by two so-called leakage resistors between the gate and both drain and source. Another resistor, with the same value, is located at the gate. Every resistor has a capacitor in parallel which value is the total parasitic capacitance between the nodes they are located. The ones connected with the drain and source are calculated by the transistor parameters and the other considers scaling effect caused by the charge sharing in the polysilicon.

This could work if we had enough information for determining the resistor values. The gate resistor can be set at the *Ropen* value, but the other two



(a) From [16] (b) From [36] (c) From [17]

Figure 4.3: FGT model examples

not, since is reasonable to think that these values depends on the transistor itself.

Looking at [36], the gate voltage is decided by a model of the tunneling and injection current, injected with voltage-controlled current sources. The initial value, for the DC analysis, is settled by a voltage-controlled voltage source with a high output resistance, as high as possible. The sources have a gain dependent on the transistor parasitic capacitors. The calculated voltage is settled by a buffer at the gate of the transistor in Fig. 4.3b.

Recently, [17] has shown that an open-gate transistor can be modeled in DC by a transistor having the same electrical properties but operating but in the sub-threshold region. That sub-threshold region is forced by a voltage-controlled voltage source which gain depends on, again, parasitic capacitors. In that paper, the ratio between V_{GS} and V_{DS} is inside the margin showed in Eq. (4.1). Even though this premise is debatable, we will use this model as reference for the DC analysis.

$$\frac{C_{gdo}}{WLC_{ox}} \leq \frac{V_{GS}}{V_{DS}} \leq \frac{3C_{gdo}}{2WLC_{ox}} \tag{4.1}$$

On the other hand, for the transient analysis the model should be different. The gate voltage in the transient analysis is set by the parasitic capacitors in its leakage current and, reading the BSIM3v3 documentation [37], the parasitic capacitors are more than modeled for the transient analysis. So that, we will consider as correct modeling the gate open by a large resistor for transient defect simulations. This model should be conservative in the sense that it will account for dynamic effects, the gate voltage is actually low-pass filtered, but will not disrupt the operating point.

4.2 Injection

The process of injecting defects starts from the netlist obtained from Cadence Virtuoso®. But to understand why and how it is needed to explain the netlist itself.

```
simulator lang=spectre
global 0
parameters ...

include "..."

// COMMENT
subckt NAME LIST_OF_TERMINALS
paramters ...
  INSTANCES
ends NAME

NETLIST_TOP_LEVEL

SIM_CONFIGURATION
```

At the beginning of the document we need to add the language used in the document. For Spectre® is a must. It only gives the possibility of putting comments (// ...) before of it. The rest of the structure it is more a good practice to avoid mistakes than a mandatory structure.

The label *global* determine the global nets. In the case there are not global nets declared it appears a 0, that will be used as voltage reference during the simulation. Second line correspond to the *parameters*. This can be written any time you want to initialize a new variable and can contain more than parameter in the same line. The only condition is that before using a variable it must be previously declared with this label. It is important to point out that every variable declared in the document will be considered as global, what is to say, a modification of the parameter will affect all the netlist, even if the edition is done after the use of the variable.

Next part are the *include "..."* sentence. This command only add the content of the file between commas to the netlist. It does not matter the format of the document, even no format works, because it will read it as a text file. The only condition is that the text file has a Spectre® code structure, that is, starting with *simulator lang=spectre* and having no syntax errors. However, there are some conventions in the text file format for as an habit of order.

- Files that finish with ".mdl" correspond to models, transistors, technology capacitors, resistors or inductances, etc... Some layout extractors,

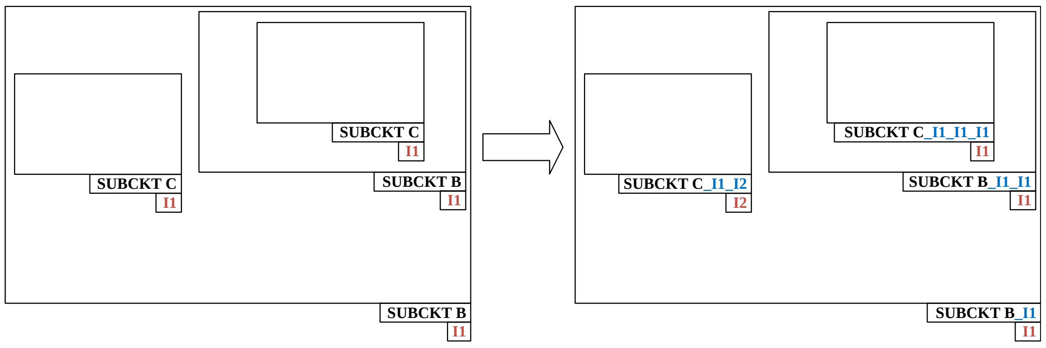


Figure 4.4: Flattening process

for example, use this format to add the parasites.

- Files finishing in ".inc" normally contain parameters. Model variables, Monte-Carlo variables, etc...

- ".lib" files correspond to the libraries. Here sections are defined. Starting from the nominal, and finishing with corners and Monte-Carlo.

After the models we have the subcircuits declaration. This is the part where we need to work more. By default, subcircuits are only declared one time, independently of the number of times it has been called. An edition of one parameter inside the subcircuit will affect to all its calls. That is why we need to flatten the netlist before working with it.

The netlist top level is where the testbench is defined. In there, the upper level of the circuit is called and the necessary environment to test the circuit is set. The only modification that is expected to be done here is editing the name of the upper subcircuit.

Finally, it is the simulation configuration. Tolerances, analysis, saved voltages, saved currents and temperature is set among these lines. We will see later that we can use the same analysis that is declared here using SpectreMDL® in order to ensure the same result we obtained with ADE L.

The basic process of flattening is showed in Fig. 4.4. With an input netlist, the program starts by reading all the declared sub-circuits at the beginning of the document. Afterwards, it reads recursively the document to save the dependencies of that sub-circuits and extract a tree with every sub-circuit instance. The name of the sub-circuit is edited depending on how deep is the sub-circuit instanced. For example, if the call is made at the first level it will only have its own name with the instance string immediately afterwards. If it is a second, third and so on level, the complete list of instances need to reach the sub-circuit is written at the end of the name. On this way, we achieve a netlist with non-repeated instances.

That is the only way to guarantee that there will not be double sub-circuit declaration. Of course, it is more efficient to only flatten the specific sub-circuit we are going to edit, but we have made tests and for Spectre® it is transparent or, at least, the simulation time is much more higher. The good point of doing this way is that the measurements done for the input netlist are perfectly compatible with the output, since the measurements depends on the ID and net name inside the sub-circuit, not on its name.

At the end of that process, the netlist is read again in order to obtain the necessary parameters to elaborate a list of Relative Likelihood. In order to calculate the area of the transistor we need the width, height and multiplicity of it but, for the sake of versatility, this requires a more complex process. First of all, the global parameters are read. It is common that the sizes are between this list. Then, we start by reading sub-circuits in order, tanking into account that more parameters would be declared locally. Once a potentially defective device is reached, the three model parameters are extracted and the parameters declared on them are substituted by the real value. At this point, the code is capable of substituting any dependency of parameters, so we have not had to set any requirements to the designer.

Once the netlist is flattened, we are sure that the defect we inject in the netlist will only appear at the block we want to, and not any other. On this part only rest to decide how the defect is gonna be inserted.

On our case, we have edited the transistor models in such a way it only requires add one entry parameter to the transistor call in the netlist. All the transistors models have been replaced by our custom models, keeping functionality with Cadence Virtuoso® simulation environment. By default, the models does not have the defect parameter in the netlist, so that Spectre® will use its default value, which correspond with the nominal behavior. Once a defect is selected, we follow the root of the defective device until we reach it. The defect injection only requires to add the defective parameter at the end of the call line.

4.3 Simulation

Simulation part has been prepared to be performed in Matlab®. A set of functions have been written in order to perform all the IFA process. The functions are hybrid codified between Matlab® code and PERL®. The use of the last one has resulted to be really useful because of the regular expressions. With it use, text edition, the netlist, becomes much more easy to perform.

During the flattening the data for the RL calculation is extracted, that is, width, length, multiplicity and number of fingers. Furthermore, some extra information is prepared to be read, as the complete route from the top to the transistor, the name of the original sub-circuit where it was placed and the list of possible defects for each transistor.

Relative Likelihood can be calculated after the flattening, using Matlab®, depending on the case. A first approximation is the mentioned in the first section. We calculate the area and multiply it by a constant depending on the defect, that is, 5 for opens, 3 for finger opens, 0.5 for the base open and 1 for shorts. This will make a more realistic approach to Relative Likelihood once the array is normalized by the sum of all the values.

Defect selection is done attending the the obtained Relative Likelihood. The highest it is the most probable to be selected, and the defect is not discarded once it is selected in order to accomplish the Likelihood-sampling explained before. In addition, the code is capable of discriminating in the defect selection by sub-circuit. That is, we can simulate the same defect in different test-benches or even in different circuits. Once the defect is selected, the injection and simulation process has been reduced to fill the entry data of a function. That function perform the simulation using SpectreMDL®. The software use Spectre in background to simulate netlist.

This software, apart from needing a valid initialization of Cadence in the same environment, requires an extra file that defines the models. The structure of this files is defined in the documentation [38] so that we will only mention some functions.

SpectreMDL® is a scripting language that you can use to simulate a given netlist from Virtuoso®. Between its features, it includes the creation of measurement aliases that can be easily reused in different circuits, efficiently run simulations in batch mode and the parameterization of measurement aliases.

The use of this code allows us to fasten the simulation process. It has his own protocol to perform measurements and save it in the format you indicate. Some measurements examples are a periodic sampling during a transient a analysis, performing a FFT from a specified signal, saving the DC operating point or calculate the settling time of a determined times of periods. Anyways, the database generated is perfectly compatible with the *Results Browser* of Virtuoso® and with the Matlab® toolbox to read this kind of files.

In order to clarify concepts we will explain a basic process simulation of a defect:

- Flatten netlist: With the models information and the netlist itself a

PERL® reads the netlist a create a new one with an unique declaration for every sub-circuit. As a result, we obtain the new netlist and the width, length, multiplicity and number of fingers of each transistor.

- Relative Likelihood: Likelihood computing is done immediately after. Depending on the case, a matrix with the number of considered defects as columns and the number of devices as rows is filled with the area of the devices. The columns which correspond to open defect are multiplied by 5, the finger opens by 3, the base open by 0.5 and the shorts are not edited. Finally, the matrix is normalized with the sum of all the elements.

- Simulation loop: With the list of candidates and its ideal Relative Likelihood the simulation loop begins.

  - Selection: A defect is selected considering its Relative Likelihood and the sub-circuit it belongs. If the defect has already been simulated the output Relative Likelihood position which correspond to it is incremented by one. If not, a new element is pushed at the end.
  - Defective netlist generation: With the defect info, a new folder is created to perform the simulation. The defective netlist is saved in that folder and SpectreMDL® is invoked with its configuration file.
  - Results: Depending on the simulation the results to read may variate. For example, for the DC operating point the results would be the interesting DC nodes but for a transient simulation it could be the final value, the sampled wave or the settling time. Either way, the results can be extracted from the simulation database by configuring the SepctreMDL® script or using the Matlab® functions provided by Cadence®. The important thing is that the results must be read now, since the simulations files are quite big and a complete simulation could fill the free space on disk.

The most important advantage of the flattening process is its versatility. The code accept almost every netlist generated by Cadence® so no extra conditions have been imposed to the designer. This point allows the simulation on different setups wether it is different setup or complete different circuits.

4.4 Collateral developments and applications

During the code development, other application ideas appeared. Some of them are already coded and others are planned to be performed in the future. One example of this is the Monte-Carlo analysis, that seemed to be useful to run it in Matlab® taking absolute control over it. Another interesting point is the capability of using the same structure of defect injection to simulate Single-events in transistors. This highlights the versatility of the code produced.

4.4.1 Monte-Carlo analysis

Adapt the Monte-Carlo to be performed in Matlab® is not a trivial task, since every technology, even though they use the same base process, produce the variations using the Spectre® random numbers generation [39]. But our intention was to take control of the analysis as much as possible, allowing us not only to read the variations obtained but to repeat the simulation in a different setup with the exact same variables.

For this purpose, the custom models have the information of how many variables needs to be generated to perform an analysis and with what distribution they need to be generated. During the flattening process this information is extracted to a Matlab® readable format so as it can generate and save them.

This extension has certain advantages over the traditional method. On one hand, we can apply the same process and mismatch variations to a single block in different setups. In Cadence Virtuoso ADEGXL® this could be done for process but not for mismatch. On the other han, since the variable values are assigned in Matlab®, we ca use deterministic values for diagnosis purposes, or perform any kind of adaptive process. This is of particular interest to examine the tails of the performance in high-yield applications or to validate machine-learning indirect test.

4.4.2 Single Event

As a proof of the toolbox versatility single events simulations has been implemented. Single Event Transients have become a great threat to the reliability in space applications. They consist on the effects of the localized charges injection that accompanies the impact of a radiation particle in the active zone of the semiconductor crystal. The work done with the toolbox has already been accepted in DCIS 2018, [40].

In order to perform Single Event Transient simulation for large circuits we re-used the defect injection platform. The concept is very similar to the defect simulation: the Relative Likelihood of a Single Event Transient in a given MOS transistor is proportional to the source or drain area. We thus extract that information from the netlist and produce 2 Single Event Transient models for the MOS with a double exponential circuit source connected between the body and the drain (or source) terminals. This model has 4 entry parameters (the total charge, rise and fall time and the injection instant) that can be farther varied during simulations.

Chapter 5

Cases of study

5.1 First case of study: Resistive buffer

The results exposed in this section have been published for the conference IOLTS 2018 [41]. The purpose of this case of study was do demonstrate the need for the complete test picture in order to assess test quality.

5.1.1 Circuit description

The circuit under test is an unity gain differential closed-loop amplifier with resistive feedback (RFB-AMP) that is shown in Fig. 5.1. This buffer was designed to cope with the demanding requirements imposed by the input stage of a high-performance switched-capacitor ADC. More concretely, the design goal was to achieve an effective resolution of 13 bits for a $2V_{pp}$ signal range, withstanding a switched capacitive load of $2pF$ at $50MHz$.

The core of the resistive buffer is a the two-stage op-amp with Miller's compensation shown in Fig. 5.2. In addition, three common-mode feedback (CMFB) circuits are used for stabilizing the voltages at the input, the first and the second stage outputs of the op-amp. Such an independent control of the common-modes was deemed necessary to optimize linearity performance. In the example of Fig. 5.1, continuous-time architectures with active auxiliary op-amps at the input and output are shown for this purpose but discrete-time CMFB circuits can be used based on switched-capacitor (SC) implementations [42].

Despite its apparent simplicity, this resistive buffer is a high-performance block that operates at the limits of the technology capabilities. Thus, the design margins are quite limited and the parametric yield is not 100%, as we will see later.

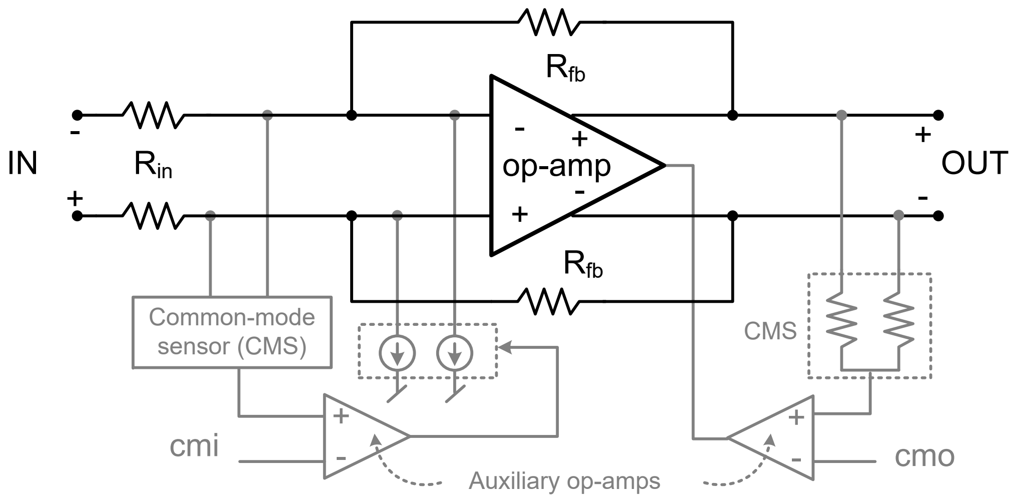


Figure 5.1: Simplified schematic of the DUT based on an unit-gain differential closed-loop amplifier with resistive feedback (RFB-AMP).

Table 5.1: Key electrical specifications of the DUT in a 1.8V UMC 180nm process.

Characteristic	**Specification**
Amplifier DC Gain	79dB
Amplifier GBW	196MHz
R_{in}	500Ω
R_{fb}	500Ω
Buffer THD	80dB

Table 5.1 summarizes the main specifications of a transistor-level implementation in a 1.8V UMC 180nm process, herein used as DUT. The challenges in the design were solved using the methodology in [43] which splits the process in two steps: 1) generation of a set of candidates based on an analytically-derived linear model (3-poles and 3-zeros) to assure a dominant critical damped response, 2) selection of the final solution using transistor-level simulations with a realistic SC load model to take into account the actual non-linear transient evolution.

The total number of potential defect sites after flattening process is of only 304. This is a manageable number to consider an exhaustive defect simulation if fixed resistors are used.

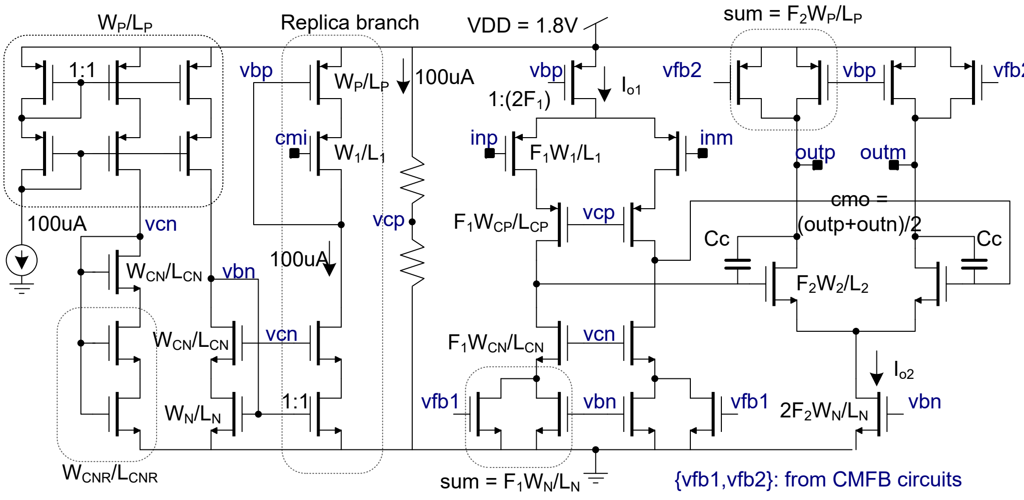


Figure 5.2: Transistor-level implementation of the differential two-stage op-amp.

5.1.2 Test description

The difficulties found in the design are also found during testing, specially when process and mismatch variations are considered. Characterizing the performance of such a buffer requires a dynamic test consisting in applying an at-speed highly-pure sinusoidal signal at the input of the circuit, and acquiring and processing its output response in order to compute dynamic performance figures such as the THD, ENOB, etc. The high performance of the circuit imposes demanding requirements on the test equipment. And even if the appropriate equipment is available, the buffer would probably be integrated within a complex system and its primary inputs and outputs pins would not be accessible.

In this paper, we will consider as a defect-oriented approach the measurement of the DC voltages of the amplifier internal nodes. Such an approach has been followed by many proposals in the past and practical solutions have been proposed to acquire such DC voltages in an embedded circuit, like the IEEE 1149.4 analog test bus, or more recently resorting to local one-bit digitizers with an external reference [44].

In order to process these DC values and output a pass-fail decision we decided to perform a Machine Learning Indirect Test. Fig 5.3 shows the process of training and using a regression model to THD for the DC values obtained during a Monte-Carlo simulation of both process and mismatch. Using this model directly on the fault simulation results would not be appropriate as pointed out in [45]. However, we can consider it in combination with a

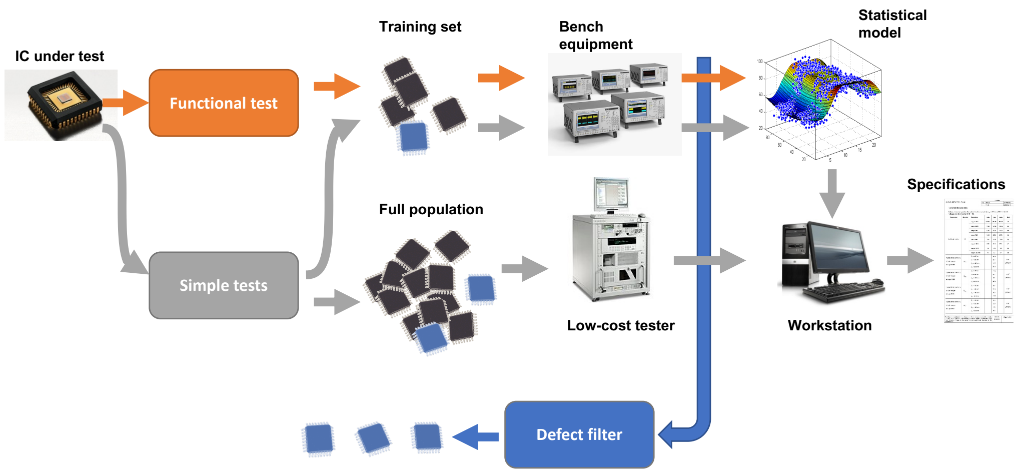


Figure 5.3: Machine Learning Indirect Test Process

Defect filter. Indeed, in a standard Machine-learning indirect test flow (also known as Alternate test), the defects that manage to pass the filter would be submitted to the regression model and they may output a performance below specification and be diagnosed as defective, increasing the overall test coverage.

In our case, we are going to implement two different defect filters, with two options for each of them.

Independent pass-fail limits

Consider independent pass-fail limits on each node. The thresholds are set by considering the maximum and minimum values obtained during a Monte-Carlo simulation of both process and mismatch. This can be considered as a simple and straightforward approach. For such a solution, the inclusion of embedded window detectors can be envisioned [46].

It is reasonable to think that narrow intervals should make better defect detectors than large ones. For this reason, we also computed the acceptance windows considering only the circuits that met the $80dB$ THD specification in the Monte-Carlo simulation.

Joint probability density

Evaluate the joint Probability Density Function (PDF) of the DC values obtained during a Monte-Carlo simulation of both process and mismatch using kernel methods. Set the threshold on probability as the minimum value

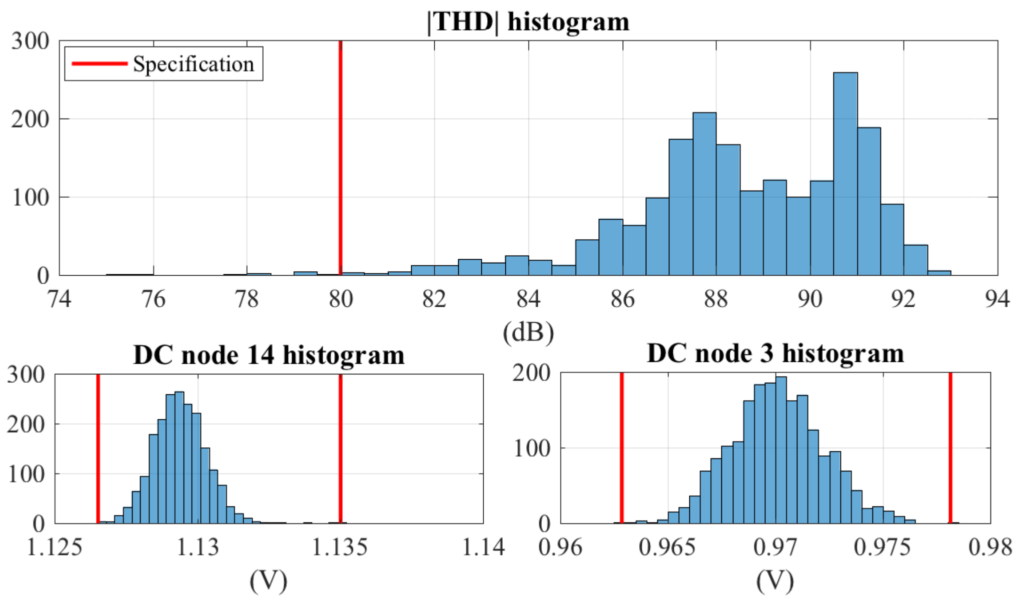


Figure 5.4: Results of the Monte-Carlo process and mismatch simulation. THD histogram and DC node voltage excursions.

obtained by the circuits that fulfill the specification. This approach is the one proposed as a defect filter for alternate test approaches in [45].

The probability of occurrence of the DC values obtained by fault simulation can then be easily computed using the trained PDF and compared to the threshold. Unlikely sets of DC values are tagged as defects.

This approach is a refinement of the previous one in the sense that it accounts for the possible correlations between the different DC nodes (i.e. if an existing correlation is broken it may be the signature of a defect, even if the individual nodes are in their validity range) and it also account for the reduced probability of two extreme events at a time.

5.1.3 Results discussion

In order to obtain the necessary data to compare with the defects the nominal point it is not enough. For that, we have performed a Monte-Carlo analysis of 2000 points including mismatch and process variations.

Fig. 5.4 shows the obtained results from the Monte-Carlo analysis. The first graphic represents the THD histogram. According to it, we could set a $80dB$ specification for the THD, which would give a parametric yield of 99.5%. On the other hand, the second graphic shows the DC voltage excursion of each node. At this point, we could consider to display the intervals for

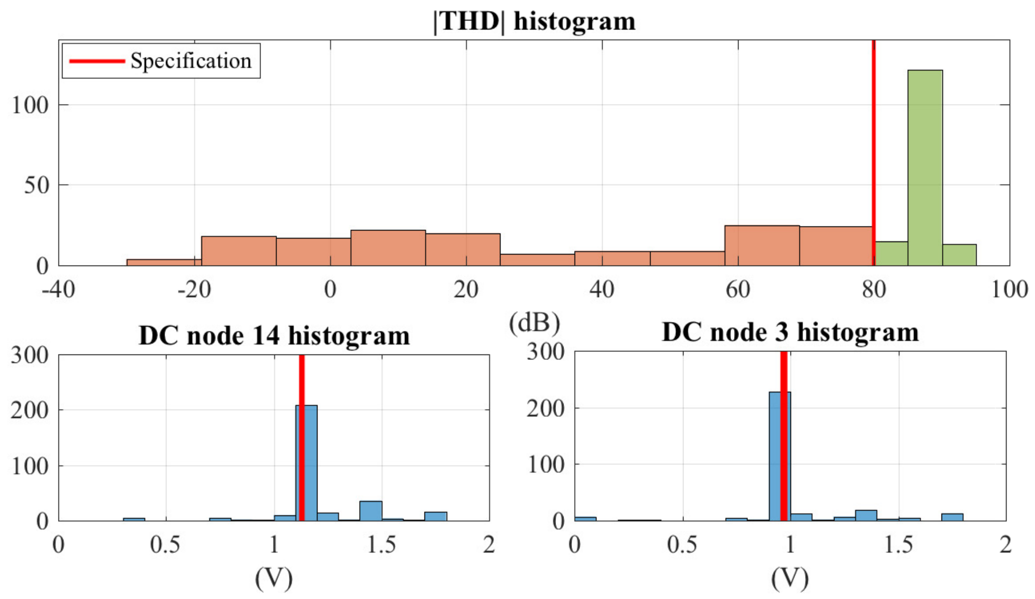


Figure 5.5: Results of the defect signature generated. THD histogram.

circuits with more than $80dB$ of THD but, as will be shown later, this does not produce a significant effect on the results.

Once obtained the Monte-Carlo data, the fault simulation can be carried out. For this purpose, we exhaustively simulated the list of candidate defects. We thus obtain a null variance, attending to the equation (3.8), hence it will be assumed for all the results.

As commented above, the most computationally efficient way to estimate the fault escapes would be to simulate the performance test setup only for the defects that have passed the test. However, we have not done that because we want to check different test methods and it is not possible to know beforehand which defect will fail which test. We have thus simulated the performance for each defect in the candidate list, since the was affordable for this small circuit.

As before, Fig. 5.10 shows the data obtained from the defect signature generated. At first sight, it confirms that not all the defects have to be a fault. Actually, 49.01% of the circuits meet the specification of $80dB$, so any test (even no test) would obtain at least a modified Fault Escape of 50.99%.

Let us now consider different test methods, that will make use of the available data.

Independent pass-fail limits results (IPF, IPF80dB)

Using all the Monte-Carlo data without the mentioned restriction of $80dB$, we obtained a defect coverage of 63.16%. The point is that only the 8.04% of the undetected defects represent an actual fault, that is, have a THD less than $80dB$. It comes that the modified fault coverage reaches 97.04%. So despite a moderate defect coverage score, such a test could be considered good enough since it will catch most of the actual faults. In addition, the severity of the escaped faults is $0.432dB$. This test will be referenced as *IPF* henceforward.

When stretching the pass-fail margins by setting a minimum THD of $80dB$ in the Monte-Carlo data the results do not improve. This method will be called *IPF80dB*, in order to differentiate it from the previous one.

Joint probability density results (JPD)

With respect to the *IPF* method, this one implicitly takes into account the correlations between nodes so it will increase the number of restrictions on the possible values. In this way, we expected to detect defects that, even though being inside the individual nodes margins, do not fulfill the correlations. However, we found no significant difference with the previous method.

The defect coverage reaches the value of 61.18%, with a fault coverage of 96.71%. It actually slightly worsens the results of *IPF*, what it is noticed in the severity of $0.436dB$. But a closed look at the results shows that it corresponds to only one more escape than before, located quite close to the specification boundary, as can be seen in Fig. 5.6. There are no changes either when we apply this method considering as an input only the circuits with more than $80dB$ THD to compute the joint probability density function.

These slightly worse results are probably due to the kernel estimation process which by construction somewhat widens the tails of the distribution while the IPF method consider the extreme cases of the Monte-Carlo simulation as an absolute limit. What is clear is that, at least for our case of study, the correlations between DC voltages bring no relevant information to defect detection.

Machine Learning (ML, IPF80dB+ML, JPD+ML)

For this test we train a neural network in order to predict the THD with the DC test signatures generated on the Monte-Carlo analysis. This is what is usually done in machine-learning indirect test approaches [3, 47, 48].

At first, we tried to directly submit the DC signatures of defective circuits to the performance predictor. Poor results were expected since the necessity

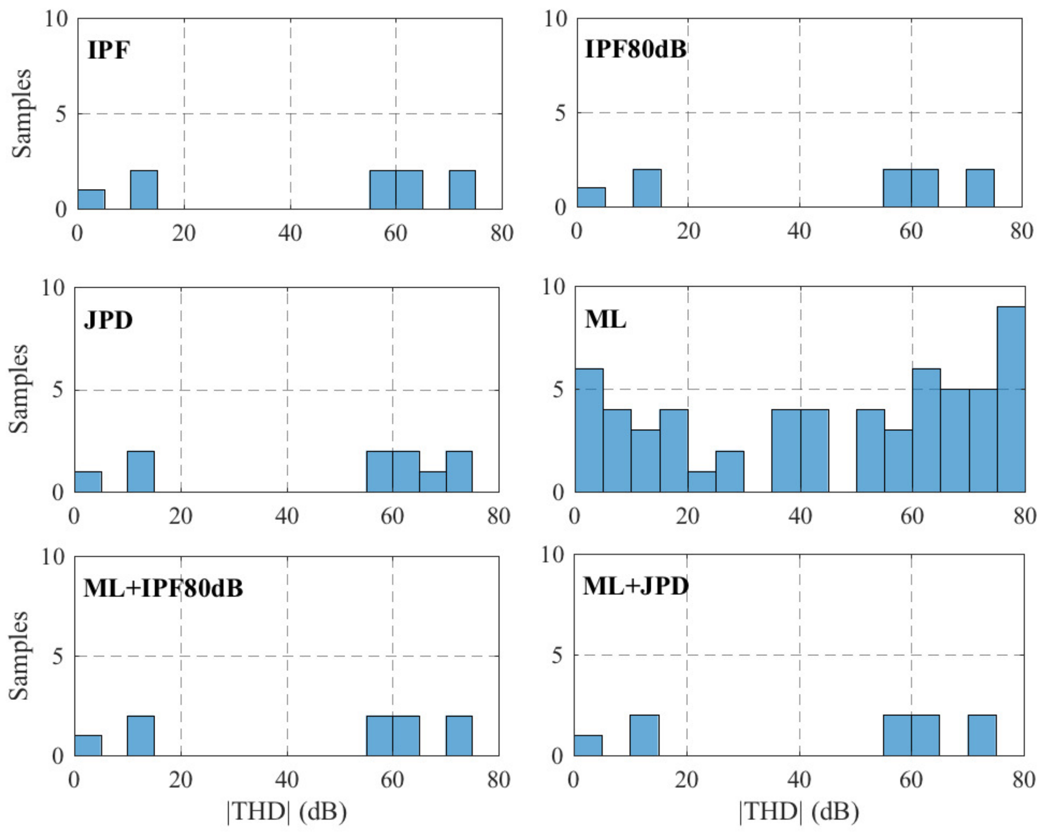


Figure 5.6: Undetected faults THD histogram.

Table 5.2: Results comparison table.

Test	**Defect coverage**	**Fault coverage**	**Severity**
noTest	0%	50.99%	3.4222dB
IPF	63.16%	97.04%	0.4318dB
IPF80dB	63.16%	97.04%	0.4318dB
JPD	61.18%	96.71%	0.4335dB
ML	21.71%	70.39%	1.752dB
ML+IPF80dB	63.16%	97.04%	0.4318dB
ML+JPD	61.51%	97.04%	0.4318dB

of a defect filter is acknowledged in the literature, as commented above. This test is not enough to be used as a defect detector by itself. Its performance is too low: it only detects 21.7% of the defects. The modified fault coverage is 70.4%, which is not so far from the result of performing no test at all. And the severity is $1.75dB$, significantly higher than the previous methods. This means that the fault escapes are not close to the specification, as can also be seen in Fig. 5.6.

Considering the complete machine-learning indirect test flow, the circuits that pass the defect filter would then be submitted to the trained performance predictor (i.e to the Neural Network model in our case). It may thus occur that the model detects some additional defects. So we tried to use the trained model on the defect escapes of *IPF80dB* and *JPD* methods.

The results show very marginal improvement. Actually, the additional machine-learning step does not improve the defect detection statistics of the stand-alone *IPF80dB*. If used in conjunction with *JPD* we reach the *IPF80dB* modified fault coverage and severity, but not its defect coverage.

Table 5.2 summarizes the results obtained for the different methods and Fig. 5.6 shows the histogram of the undetected faults in each case. It appears quite clearly that the different tests provide similar results, except the Machine-Learning test which is clearly inadequate.

From these results we can conclude that the proposed complete test picture is of interest to assess test quality more accurately. In particular, we can see that the test with a moderate Defect Coverage does nor perform that bad taking into account the complete information.

5.2 Second case of study: Source follower

This second case of study considers a more complex circuit and it is aimed at validating the statistical approach.

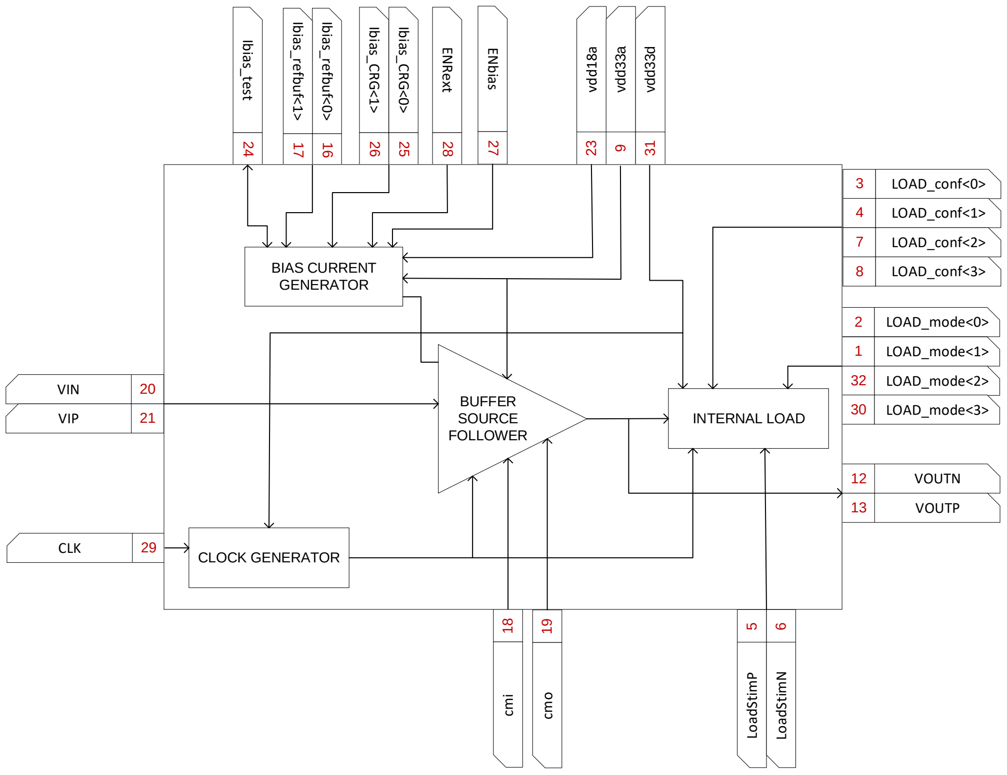


Figure 5.7: Block diagram of the chip.

5.2.1 Circuit description

It is a differential buffer differential buffer based on source-followers designed to be in cascade with the previous circuit in order to reduce the kickback generated during the sampling of the ADC with a higher load or as a standalone buffer. Unlike the previous one, the design goal was to achieve an effective resolution of 14 bits for a $2V_{pp}$ differential full-scale signal, withstanding a switched capacitive load of $10pF$ at $100MHz$ over the first Nyquist bandwidth (up to 50MHz).

In order to achieve this level of accuracy, the source followers have to be linearized and the biasing scheme ensures a very accurate reference current as well as a constant source-drain voltage. Design constraints were very demanding with regards to common mode output and bandwidth so that that the circuit is polarized at $3.3V$ in order to achieve saturation in all the output range with a common mode output of $1V$.

In this case we will not simulate only the circuit core, but the whole chip in its prepared test-bench. The chip is divided in the blocks showed in Fig. 5.7. It has three different polarization voltages in order to separate biasing, digital

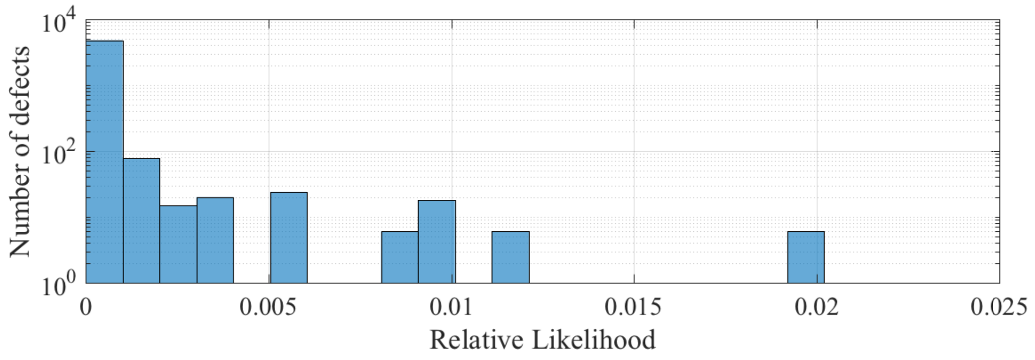


Figure 5.8: Relative Likelihood histogram extracted from the Buffer Source Folloer chip.

and analog parts. Thinking on the final aim which was to perform a BIST an internal switched load was included that can vary to total capacitance from disconnected to $10pF$.

This circuit has been integrated and will soon be test un the laboratory. Despite the developed toolbox allows the defect injection only in the buffer, in order to have enough samples to validate the statistic approach, we will consider the injection in the entire chip. Netlist flattening and defect extraction process output a lis of 4962 candidates with a Relative Likelihood distribution showed in Fig. 5.8.

5.2.2 Test description

Test-bench needs of a transformer followed by a bandpass filter tuned at the input frequency to generate the $2V_{pp}$ with enough resolution and the output is read by a ADC, so that the internal load is turned off during the test.

As before, in order to access performance test-bench simulation requires a long transient of 256 samples for performing a FFT. The input signal is a full-scale sine-wave close to $50MHz$. In this case, instead of the THD, the SNDR is calculated.

Here again, we will consider as a Defect-Oriented Test the measurement of internal DC nodes. We will use the Independent pass-fail limits used in the previous circuit. The test pass-fail will be the maximum and minimum results obtained during the Monte-Carlo simulation of 1000 points. From the computational perspective it only requires the simulation of the DC operating point, so the simulation is accessible from the test-bench performance simulation.

The total number of possible defects is of 4962. Our intention is to proove

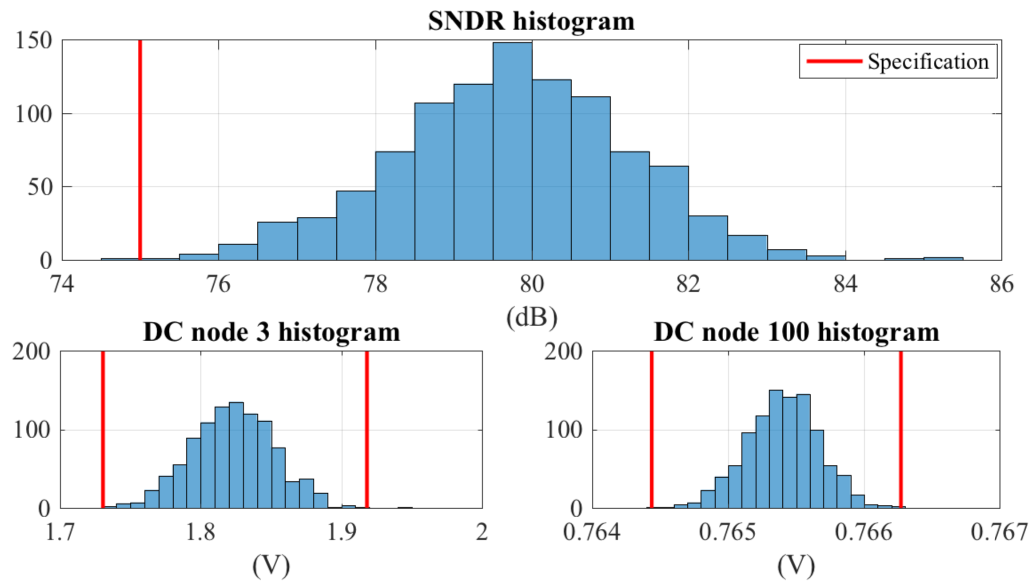


Figure 5.9: Results of the Monte-Carlo process and mismatch simulation. SNDR histogram and DC node voltage excursions of 2 nodes.

the capability of the proposed non-exhaustive defect simulation but, in order to compare results, the exhaustive simulation is necessary. For this reason, using the layout extracted netlist with parasites is not an option. Bear in mind that the exhaustive simulation at schematic level lasted two weeks with four parallel cores.

5.2.3 Results discussion

Fig. 5.9 shows the Monte-Carlo histogram results. Due to the fact that the chip was not intended to be implemented alone the SNDR specification was reduced to $75dB$ to obtain a parametric yield of 99%. The exhaustive simulation of defects led to the histogram showed in Fig. 5.10. As before, the necessity of considering defects and faults is evident. In this case, only the 20.82% of the simulated defects are a fault.

Exhaustive test reached the value of 74.26% for the Defect Coverage and a 2.18% of Fault Escape, which corresponds with a Fault Coverage of 97.82%. The contrast is again important. Severity has not been implemented in this case of study since we will not compare with any other test.

Thanks to the exhaustive defect simulation, we now know the true value of the Defect Coverage and the Fault Escape to which the estimates should converge in a non-exhaustive defect simulation framework. In order to validate

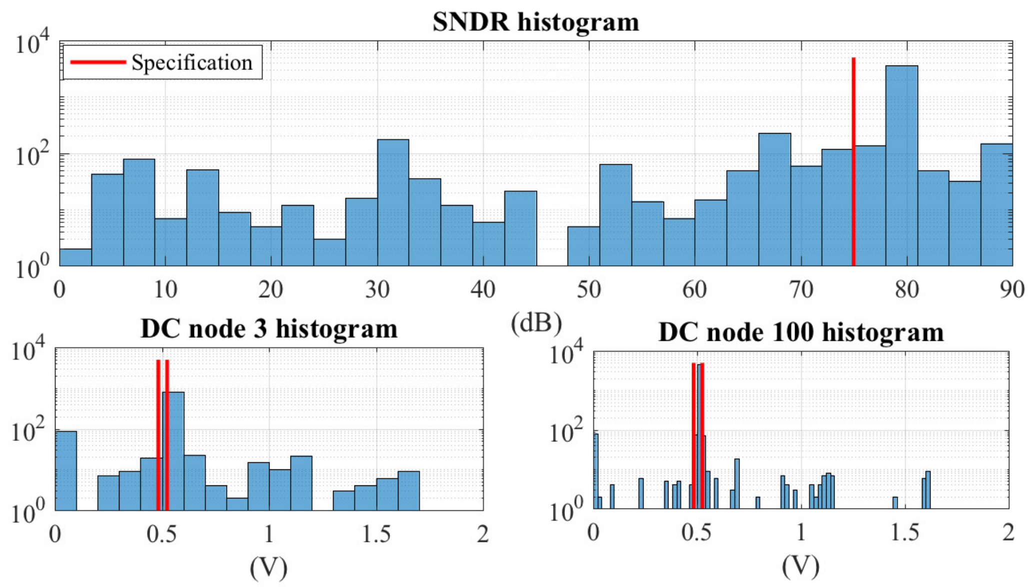


Figure 5.10: Results of the defect signature generated. SNDR histogram and DC node voltage excursions of 2 nodes.

our approximation on the Confidence Interval we will first perform the same check we did in chapter 3. That is, perform several virtual sampling of the same length for obtaining an histogram and be capable of extract a real Confidence Interval of the metrics. After that, a single defect simulation estimates of the Confidence Intervals will be compared to the true value. Results are shown in Fig. 5.11. As before, Fiellers' theorem does not give good results for low values, so it cannot be used for this work. Between Cochran and Sunter we decided to use Cochran, since the invalidity of the Confidence Interval is easy to discriminate.

First of all, the algorithm needs to set the desired Defect Coverage and Fault Escape to discard or not the test. The first loop calculates the Defect Coverage and, if it is valid, the second calculates the Fault Escape. For this reason it is needed to test the algorithm behavior with different specifications and compare with a reference. Obviously, the obtained results are compared with the exhaustive test but, in order to see the improvements in computational cost, a conventional approach would consist in computing the number of defects to be simulated in order to reach a given precision on the estimate.

That necessary sample size will be calculated with the binomial approximation of the variance. The number of samples for a determined width of the Confidence Interval is in Eq. 5.1. For a 90% Confidence Interval, $1-\alpha = 0.99$, the Z-score, $z_{\alpha/2}$, is 1.645. We will set the width at the absolute value of

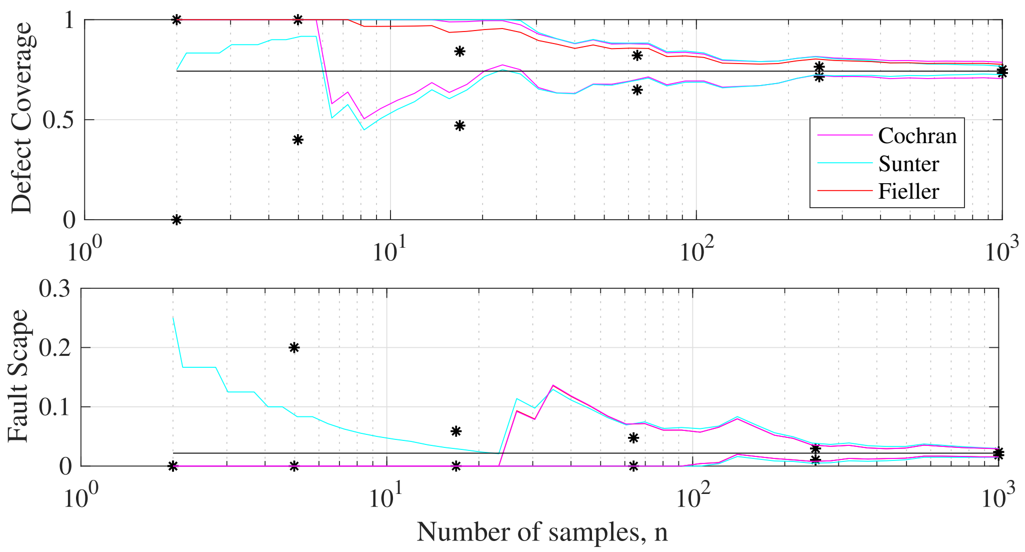


Figure 5.11: 90% Defect Coverage and Fault Scape Confidence Interval progression with the number of samples.

0.05, so $CI = 0.05/2$. Substituting the Defect Coverage and the Fault Escape expected value in r will give us the number of minimum simulation needed to reach a result with that confidence.

$$n = \frac{r(1-r) + \left(\frac{CI}{z_{\alpha/2}}\right)^2}{r(1-r)/N + \left(\frac{CI}{z_{\alpha/2}}\right)^2} \tag{5.1}$$

The sequential simulation process has been described in Chapter 2. However, for the sake of validation, we have performed an exhaustive simulation

Table 5.3: Number of samples comparison table for non-exhaustive test.

deDC	80%	74.26%	60%
deFE	1%	2.18%	5%
minSampDC	609	711	861
minSampFE	363	92	199
ds	27	279	18
fs	0	37	3
dc	$63.64\% \pm 14.46\%$	$73.45\% \pm 2.5\%$	$85\% \pm 15.95\%$
fe	Not calculated	$2.68\% \pm 2.49\%$	$0\% \pm 2.5\%$
Result	Not accepted	Accepted	Accepted

of the defect candidates, both for the proposed defect-oriented test setup and for the performance test setup. This allows us to emulate that sequential simulation as many times as desired and for several conditions in terms of defect coverage and fault escape objectives.

We thus first estimate the Defect Coverage through likelihood random sampling and we stop as soon as the test under evaluation can be considered as valid or invalid. In the case that no clear decision can be taken, we stop when the estimate of the confidence interval reaches the desired value of 0.05. With the Fault Escape, we take the list of defects that have been simulated in the defect-oriented test setup in the same order. If a defect was labeled as detected, it is not simulated for performance but it is directly assigned a fault detected label. On the contrary, if it was labeled as undetected for the test under evaluation (and was thus a escape), it is simulated for performance and its fault label is updated according to the result of that simulation. The exit criterion for the fault escape loop are the same as for the defect coverage: either the metric can be early accepted (or discarded) or the loop is stopped at an acceptable width of the confidence interval. It may occur that the list of defects simulated in the defect-oriented setup ends before a valid conclusion is reached on the fault escape estimate. In such case, additional defect simulations would have to be carried out.

Table 5.3 shows the results obtained for different situations. The rows are divided as follows:

- **deDC** & **deFE**: They are the desired Defect Coverage target and Fault Escape respectively.
- **minSampDC** & **minSampFE**: The minimal number of simulations needed to reach that confidence interval of width 0.05. This is thus the baseline to calculate the gains in computation time.
- **ds/fs**: Number of defect and fault simulations for our proposal.
- **dc/fe**: Defect Coverage and Fault Escape obtained at the end of the simulation.
- **Result**: Decision taken, if the test has been accepted or not.

We have considered three cases. The first column specified challenging values for the desired defect coverage and fault escape. In this case, the algorithm resulted to be much more efficient than the conventional approach since with barely 30 defect simulations the test was discarded. Here the gains in term of computation is in the order of a factor 20.

The second column corresponds to the most pessimistic case since we set the target requirements to the true value of the metrics. We thus know that early stopping is impossible. Yet again, the estimate of the confidence interval reaches the desired precision earlier than expected. This is due to the likelihood random sampling process that increase the convergence rate for non-uniform likelihoods (it is actually equivalent to importance sampling).

To complete the analysis, the last column shows the case when the desired value is lower than the true value. In this case we are able to identify as valid the test very fast but the number of escapes may not be sufficient to have a trustworthy Fault Escape estimation. So that, we would have to perform more defect simulations till the second Confidence Interval reaches an acceptable value.

Looking at the metrics, the early stopping worked with the first and third case, but when the desired Defect Coverage and Fault Escape are close to the real values the loop lasts longer.

In order to further validate the proposal, we repeated the second experiment a hundred times. In other words, we repeat the likelihood random sampling of the defects a hundred times but since we have performed an exhaustive simulation, this is done "virtually".

The results are showed in Fig. 5.12. Most cases required a large number of simulations, in excess of 200, to reach the desired width of confidence interval, as expected. However, in a small fraction (5%) of the cases, the test was early discarded. This rate is coherent with the fact that the early stopping criterion was computed with the 90% confidence interval. However, the decision to reject the test cannot be considered as erroneous since the desired coverage value is exactly the true value.

With this case of study it has been evidenced that statistical test is a valuable tool to reduce simulation time. In addition, the metric estimation is carried out in a sound statistical manner so that the obtained confidence intervals are realistic.

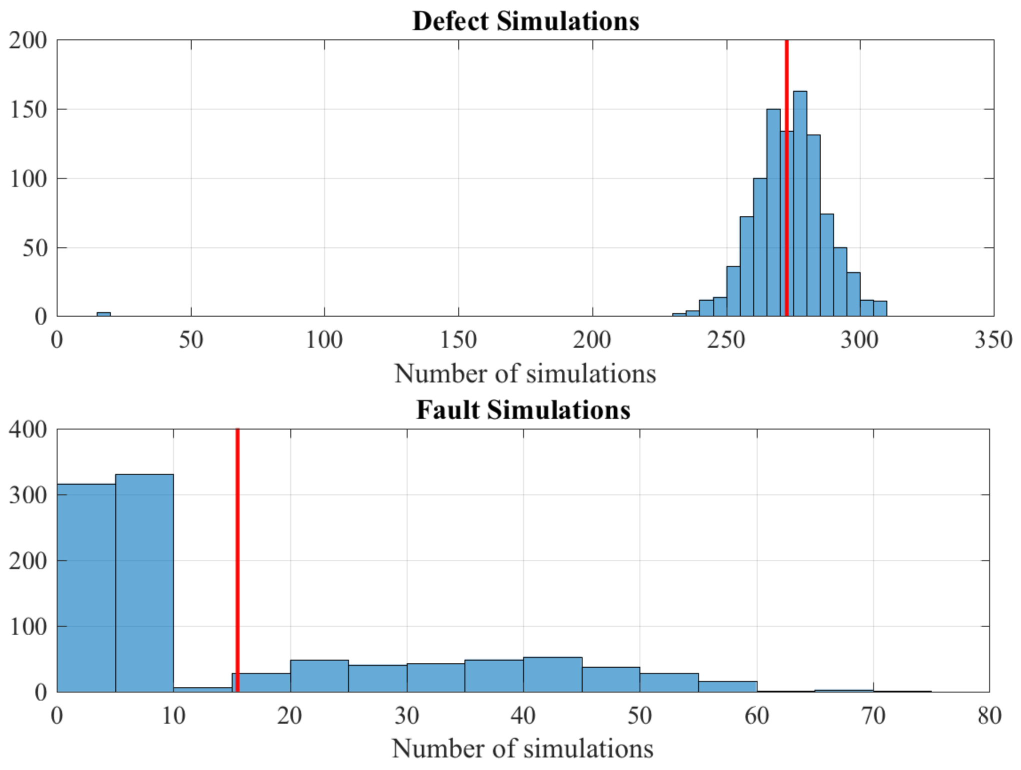


Figure 5.12: Number of simulations histogram.

Chapter 6

Conclusions

In this work, we have shown that Defect Coverage is not necessarily a sufficient metric for considering test quality. Indeed, a relatively poor defect coverage value does not mean a poor fault coverage value. As seen in the first case of study, with an average defect coverage of 63% we have reached almost the 97% of fault coverage. In addition, a close look should also be taken at the nature of the remaining fault escapes. This information is summarized in a Figure of Merit called *Severity*.

A statistically sound development has been carried out to evaluate these metrics in a non-exhaustive defect simulation. This has resulted in a study of the different Confidence Intervals used in the literature and the selection of the most useful to our case.

Finally, a sequential simulation loop has been implemented in order to perform such a non-exhaustive defect simulations. The loop contemplates an early stopping criterion for the case the test can be discarded or accepted without reaching the imposed confidence. This loop has been proved in a second case of study where the early stopping has work properly in the cases of a considerable difference between the desired Defect Coverage and Fault Escape. Thanks to that criterion, test strategies could be evaluated with significant computation time gains.

During the development of this work, three articles has been written, two for congress and one for a journal. The already published is [41] presented for the IOLST 2018. From this paper, we were invited to expand the work and publish it in "IEEE Transactions on Device and Materials Reliability (TDMR)". Finally, derived from the collateral developments, an article about Single Event detection will be presented soon, [40].

The main branch of research derived from this work for the future is to find methodologies for simulating defects more reliable and faster than the one described above. In particular, an adaptive simulation methodology to concentrate computational effort on either fault or defect is currently under development. Finally, the developed toolbox will be very useful for validating some of the research group in the field of machine-learning indirect test and robust defect filters.

Glossary

analog As mention in [49], “Analog” has a broad meaning in the IC industry. It includes mixed-signal circuits and usually custom or very-high-speed digital circuits. 1, 8

electromigration Electromigration is the transport of material caused by the gradual movement of the ions in a conductor due to the momentum transfer between conducting electrons and diffusing metal atoms. 5

golden It is used as the reference pattern, whether it’s a transistor, a measurement or a model. 4

Sensitivity Analysis Is the study of how the uncertainty in the output of a model (numerical or not) can be apportioned to different sources of uncertainty in the model input [50]. 10

Spectre A circuit simulator based on SPICE. 8, 33, 38

Acronyms

®	Registered brand. 8, 19–21, 35–40
BIST	Built-In Self-Test. 2, 53
CI	Confidence Interval. 16, 18–20, 25–29, 55, 56, 58, 61
DfT	Design For Test. 2, 6
DoT	Defect-Oriented Test. 2, 5, 13, 14, 18, 19, 53
FFT	Fast Fourier Transform. 38, 53
FGT	Floating Gate Transistor. 33, 34
FoM	Figure of Merit. 14
FPC	Finite Population Correction Ratio. 25, 26
HDL	Hardware Description Language. 8
IC	Integrated Circuit. 3, 7, 14, 31, 63
IFA	Inductive Fault Analysis. 5–7, 9, 23, 37
MSE	Mean Square Error. 25
PDF	Probability Density Function. 46, 47
RL	Relative Likelihood. 5, 6, 9, 14–16, 23–25, 27, 28, 32, 33, 37–39, 41, 53
SET	Single Event Transient. 40, 41
SNDR	Signal to Noise-Distortion Ratio. 53–55
SPICE	Simulation Program with Integrated Circuits Emphasis. 7–9, 63
THD	Total Harmonic Distortion. 53

Bibliography

[1] IEEE, *1241-2010 IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters.* 2011.

[2] G. Leger and A. Gines, "Likelihood-sampling adaptive fault simulation," in *2017 International Mixed Signals Testing Workshop (IMSTW)*, pp. 1–6, IEEE, jul 2017.

[3] M. J. Barragán, R. Fiorelli, G. Leger, A. Rueda, and J. L. Huertas, "Alternate Test of LNAs Through Ensemble Learning of On-Chip Digital Envelope Signatures," *Journal of Electronic Testing*, vol. 27, pp. 277–288, jun 2011.

[4] H.-G. Stratigopoulos and S. Sunter, "Fast Monte Carlo-Based Estimation of Analog Parametric Test Metrics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, pp. 1977–1990, dec 2014.

[5] W. R. Daasch, C. G. Shirley, and A. Nahar, "Statistics in Semiconductor Test: Going beyond Yield," *IEEE Design & Test of Computers*, vol. 26, pp. 64–73, sep 2009.

[6] S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, "Practical random sampling of potential defects for analog fault simulation," in *2014 International Test Conference*, pp. 1–10, IEEE, oct 2014.

[7] J. Shen, W. Maly, and F. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design & Test of Computers*, vol. 2, no. 6, pp. 13–26, 1985.

[8] J. Khare, W. Maly, S. Griep, and D. Schmitt-Landsiedel, "Yield-oriented computer-aided defect diagnosis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, pp. 195–206, may 1995.

[9] K. Saab, N. Hamida, and B. Kaminska, "Closing the gap between analog and digital testing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 307–314, 2001.

[10] R. Beurze, Y. Xing, R. van Kleef, R. Tangelder, and N. Engin, "Practical implementation of defect-oriented testing for a mixed-signal class-D amplifier," in *European Test Workshop 1999 (Cat. No.PR00390)*, pp. 28–33, IEEE Comput. Soc, 1999.

[11] F. M. Goncalves, I. C. Teixeira, and J. P. Teiceira, "Realistic fault extraction for high-quality design and test of VLSI systems," *Defect and Fault Tolerance in VLSI Systems, 1997. Proceedings., 1997 IEEE International Symposium on*, pp. 29–37, 1997.

[12] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI test principles and architectures : design for testability*. Elsevier Morgan Kaufmann Publishers, 2006.

[13] F. Fantini and C. Morandi, "Failure modes and mechanisms for VLSI ICs-a review," *IEE Proceedings G (Electronic Circuits and Systems)*, vol. 132, no. 3, pp. 74–81, 1985.

[14] M. Sachdev and J. Pineda de Gyvez, *Defect-oriented testing for nanometric CMOS VLSI circuits*. Springer, 2007.

[15] S. Sunter, K. Jurga, and A. Laidler, "Using Mixed-Signal Defect Simulation to Close the Loop Between Design and Test," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, pp. 2313–2322, dec 2016.

[16] W. Y. Chiew and A. K. Bin A'ain, "Characterization and model development of CMOS Floating Gate Defect (FGD)," *2007 Asia-Pacific Conference on Applied Electromagnetics Proceedings, APACE2007*, pp. 4–8, 2007.

[17] B. Esen, A. Coyette, G. Gielen, W. Dobbelaere, and R. Vanhooren, "Effective DC fault models and testing approach for open defects in analog circuits," in *2016 IEEE International Test Conference (ITC)*, pp. 1–9, IEEE, nov 2016.

[18] S. Business, "Testing Analog/Mixed-Signal Circuits Tessent DefectSim Improving AMS Test Quality and Time," tech. rep., 2017.

[19] L. Fang, Y. Zhong, and H. van de Donk, "Implementation of Defect Oriented Testing and ICCQ testing for industrial mixed-signal IC," in *16th Asian Test Symposium (ATS 2007)*, pp. 404–412, IEEE, oct 2007.

[20] R. Nivesh, B. Kruseman, B. Tasic, C. Hora, J. Dohmen, H. Hashempour, M. van Beurden, and Y. Xing, *Defect Oriented Testing for analog/mixed-signal devices.* PhD thesis, sep 2011.

[21] E. Yilmaz, G. Shofner, L. Winemberg, and S. Ozev, "Fault Analysis and Simulation of Large Scale Industrial Mixed-Signal Circuits," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*, (New Jersey), pp. 565–570, IEEE Conference Publications, 2013.

[22] A. V. Karthik, S. Ray, P. Nuzzo, A. Mishchenko, R. Brayton, and J. Roychowdhury, "ABCD-NL: Approximating Continuous Non-Linear Dynamical Systems using Purely Boolean Models for Analog/Mixed-Signal Verification," tech. rep., 2014.

[23] V. Devanathan, L. Balasubramanian, and R. Parekhji, "New Methods for Simulation Speed-up and Test Qualification with Analog Fault Simulation," in *2015 28th International Conference on VLSI Design*, pp. 363–368, IEEE, jan 2015.

[24] J. Park, S. Madhavapeddi, A. Paglieri, C. Barr, and J. A. Abraham, "Defect-based analog fault coverage analysis using mixed-mode fault simulation," *2009 IEEE 15th International Mixed-Signals, Sensors, and Systems Test Workshop, IMS3TW '09*, 2009.

[25] Z. Liu, S. K. Chaganti, and D. Chen, "Improving Time-Efficiency of Fault-Coverage Simulation for MOS Analog Circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 5, pp. 1664–1674, 2017.

[26] J. Hou and A. Chatterjee, "Concurrent transient fault simulation for analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1385–1398, 2003.

[27] C. J. Shi, M. W. Tian, and G. Shi, "Efficient DC fault simulation of nonlinear analog circuits: One-step relaxation and adaptive simulation continuation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1392–1400, 2006.

[28] H. Kerkhoff, "Fast fault simulation for nonlinear analog circuits," *IEEE Design & Test of Computers*, vol. 20, no. 2, pp. 40–47, 2003.

[29] A. Milne, D. Taylor, J. Saunders, and A. Talbot, "Generation of optimised fault lists for simulation of analogue circuits and," *IEE Proceedings - Circuits, Devices and Systems*, vol. 146, no. 6, p. 355, 1999.

[30] Jiun-Lang Huang, Chen-Yang Pan, and Kwang-Ting Cheng, "Specification back-propagation and its application to DC fault simulation for analog/mixed-signal circuits," in *Proceedings 17th IEEE VLSI Test Symposium (Cat. No.PR00146)*, pp. 220–225, IEEE Comput. Soc, 1999.

[31] W. G. W. G. Cochran, *Sampling techniques.* Wiley, 1977.

[32] W. C. Navidi, *Statistics for engineers and scientists.* McGraw-Hill, 2011.

[33] U. Von Luxburg and V. H. Franz, "A geometric approach to confidence sets for ratios: Fieller's Theorem, generalizations and bootstrap," tech. rep., 2009.

[34] J. P. Buonaccorsi, "Fieller's Theorem," in *Wiley StatsRef: Statistics Reference Online*, Chichester, UK: John Wiley & Sons, Ltd, sep 2014.

[35] E. C. Fieller, "Some Problems in Interval Estimation," Tech. Rep. 2, 1954.

[36] S. Rapp, K. McMillan, and D. Graham, "SPICE-compatible modelling technique for simulating floating-gate transistors," *Electronics Letters*, vol. 47, no. 8, p. 483, 2011.

[37] W. Liu, X. Jin, X. Xi, J. Chen, M.-C. Jeng, Z. Liu, Y. Cheng, K. Chen, M. Chan, K. Hui, J. Huang, R. Tu, P. K. Ko, and C. Hu, "BSIM3v3.3 MOSFET Model Users' Manual," tech. rep., 2005.

[38] C. D. Systems, "Virtuoso ® Simulator Measurement Description Language User Guide and Reference," 2011.

[39] C. D. Systems, "Virtuoso ® Spectre ® Circuit Simulator Reference," 2004.

[40] V. Guiterrez and G. Leger, "Single Event Transient injection in large mixed-signal circuits," in *Design of Circuits and Integrated Systems*, pp. 1–6, 2018.

[41] V. Guiterrez, A. Gines, and G. Leger, "AMS-RF test quality : Assessing defect severity .," in *International Symposium on On-Line Testing and Robust System Design*, pp. 1–6, 2018.

[42] O. Choksi and L. Carley, “Analysis of switched-capacitor common-mode feedback circuit,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, pp. 906–917, dec 2003.

[43] A. J. Gines, E. Peralias, G. Leger, A. Rueda, G. Renaud, M. J. Barragan, and S. Mir, “Design trade-offs for on-chip driving of high-speed high-performance ADCs in static BIST applications,” in *2016 IEEE 21st International Mixed-Signal Testing Workshop (IMSTW)*, pp. 1–6, IEEE, jul 2016.

[44] N. Liu, S. K. Chaganti, Z. Liu, D. Chen, and A. Majumdar, “Concurrent Sampling with Local Digitization — An Alternative to Analog Test Bus,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, may 2018.

[45] H.-G. Stratigopoulos, S. Mir, E. Acar, and S. Ozev, “Defect filter for alternate RF test,” in *2010 15th IEEE European Test Symposium*, pp. 265–270, IEEE, may 2010.

[46] A. Coyette, B. Esen, R. Vanhooren, W. Dobbelaere, and G. Gielen, “Automatic generation of autonomous built-in observability structures for analog circuits,” in *2015 20th IEEE European Test Symposium (ETS)*, pp. 1–6, IEEE, may 2015.

[47] P. Variyam, S. Cherubal, and A. Chatterjee, “Prediction of analog performance parameters using fast transient testing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 349–361, mar 2002.

[48] H.-G. Stratigopoulos and Y. Makris, “Nonlinear decision boundaries for testing analog circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 1760–1773, nov 2005.

[49] S. Sunter, “Part 1 : Analog Fault Simulation Challenges and Solutions,” 2016.

[50] A. A. Saltelli, *Sensitivity analysis in practice : a guide to assessing scientific models.* Wiley, 2004.