UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

GRADUATE SCHOOL OF NATURAL SCIENCES AND ENGINEERING

DOCTORAL PROGRAM IN COMPUTER SCIENCE

CYCLE XXXV

Moving Towards Analog Functional Safety

S.S.D. ING-INF/05

Coordinator: ______________________

Ferdinando Cicalese

Tutor: ______________________

Franco Fummi

Doctoral Student: ______________________

Dott. Sadia Azam

© 2022
Sadia Azam
ALL RIGHTS RESERVED

Abstract

Over the past century, the exponential growth of the semiconductor industry has led to the creation of tiny and complex integrated circuits, e.g., sensors, actuators, and smart power systems. Innovative techniques are needed to ensure the correct functionality of analog devices that are ubiquitous in every smart system. The standard *ISO 26262* related to functional safety in the automotive context specifies that fault injection is necessary to validate all electronic devices. For decades, standardizing fault modeling, injection and simulation mainly focused on digital circuits and disregarding analog ones. An initial attempt is being made with the *IEEE P2427 standard* draft standard that started to give this field a structured and formal organization. In this context, new fault models, injection, and abstraction methodologies for analog circuits are proposed in this thesis to enhance this application field.

The faults proposed by the *IEEE P2427 standard* draft standard are initially evaluated to understand the associated fault behaviors during the simulation. Moreover, a novel approach is presented for modeling realistic stuck-on/off defects based on oxide defects. These new defects proposed are required because digital stuck-at-fault models where a transistor is frozen in on-state or off-state may not apply well on analog circuits because even a slight variation could create deviations of several magnitudes. Then, for validating the proposed defects models, a novel predictive fault grouping based on faulty AC matrices is applied to group faults with equivalent behaviors. The proposed fault grouping method is computationally cheap because it avoids performing DC or transient simulations with faults injected and limits itself to faulty AC simulations. Using AC simulations results in two different methods that allow grouping faults with the same frequency response are presented. The first method is an AC-based grouping method that exploits the potentialities of the S-parameters ports. While the second is a Circle-based grouping based on the circle-fitting method applied to the extracted AC matrices. Finally, an open-source framework is presented for the fault injection and manipulation perspective. This framework relies on the shared semantics for reading, writing, or manipulating transistor-level designs. The ultimate goal of the framework is: reading an input design written in a specific syntax and then allowing to write the same design in another syntax. As a use case for the proposed framework, a process of analog fault injection is discussed. This activity requires adding, removing, or replacing nodes, components, or even entire sub-circuits. The framework is entirely written in C++, and its APIs are also interfaced with Python. The entire framework is open-source and available on GitHub. The last part of the thesis presents abstraction

methodologies that can abstract transistor level models into Verilog-AMS models and Verilog-AMS piecewise and nonlinear models into C++. These abstracted models can be integrated into heterogeneous systems. The purpose of integration is the simulation of heterogeneous components embedded in a Virtual Platforms (VP) needs to be fast and accurate.

Acknowledgements

In the name of ALLAH, the most gracious and the most merciful I am thankful to Almighty ALLAH for giving me the strength, Knowledge, ability, and opportunity to complete my doctorate degree. I would like to thank my respected Supervisor, Prof. Franco Fummi whose worthy guidance and professional attitude are appreciable in completing this dissertation. I am very thankful to Renaud Gillon from Sydelity B.V., Kruisem, Belgium for his collaboration and for providing me with valuable suggestions for research work. I would like to give special thanks to my research group members Enrico Fraccaroli and Nicola dall'ora. This endeavor would not have been possible without these two. They were available throughout this journey for helping and providing me guidance. I would like to thank my internal Committee members Prof. Davide Quaglia and Prof. Tiziano Villa from university of Verona to evaluate my work and providing guidance and providing valuable suggestions. Living abroad without family was not easy for me and also for my family. I am also very thankful to my family for all their sacrifices, love, support, and motivation throughout the journey. I would also like to say thanks my amazing friends Yumna Ali, Arjumandd Sadaf , Arshimna Ismail and Naila Cheema for providing me with a homely environment and for listening to me whenever I started to feel down. They all were present in my sad and happy times. I wish you all to get best in your life. I would like to say my Special thanks to my all office fellows especially Florence Dermozi for all the support and worthy guidance. I wish you all in your future endeavours. I will always remember all of you.

Contents

List of Tables . . . 1

List of Figures . . . 5

Listing . . . 7

List of Acronyms . . . 9

1 Introduction . . . 11

2 Objective . . . 13

2.1 Thesis Overview . . . 14

2.2 Thesis Organization . . . 17

3 Background and State Of The Art . . . 19

3.1 Background . . . 19

3.1.1 Some Basic Terminologies . . . 19

3.1.2 Transistor Level defect models . . . 19

3.1.3 Verilog-A language . . . 20

3.1.4 Simulation of analog circuits at different abstraction levels . . . 20

3.2 State of the Art . . . 21

3.2.1 State of the Art in Defect Modeling and Injection . . . 21

3.2.2 State of the art on transistor-level manipulation tools . . . 22

3.2.3 Fault Grouping . . . 22

4 Investigation of Realistic analog Stuck-on/off Defects . . . 25

4.1 Background . . . 26

4.1.1 Transistor-level defects . . . 26

4.1.2 AC matrices and S-, Y- or Z-parameters . . . 27

4.1.3 Defect Modeling in analog circuits . . . 27

4.2 Proposed stuck-on/off defect models . . . 28

4.3 Admittance-based analysis for fault grouping . . . 30
4.3.1 Circuit instrumentation . . . 31
4.3.2 Impact of a fault on the admittance matrices . . . 31
4.3.3 Circle-fitting method . . . 32
4.3.4 Fault grouping method . . . 33
4.4 Experimental Validation . . . 33
4.4.1 Analog circuits analyzed . . . 33
4.4.2 Transient analysis on the proposed stuck-on/off defects . . . 34
4.4.3 Fault grouping based on the circle-fitting method . . . 35

5 Predictive Fault Grouping based on Faulty AC Matrices . . . 37
5.1 Background . . . 37
5.1.1 Types of analog failure modes . . . 37
5.1.2 AC matrices . . . 38
5.1.3 Circuit instrumentation . . . 39
5.2 State of the art . . . 39
5.3 Impact of a fault on the ac matrices . . . 40
5.4 Predictive fault grouping . . . 42
5.4.1 AC-based grouping . . . 42
5.4.2 Circle-based grouping . . . 43
5.4.3 Transient-based grouping . . . 44
5.5 Methodology Validation . . . 44
5.5.1 Circle-fitting method . . . 44
5.5.2 Comparison between the different grouping techniques . . . 45
5.5.3 AC-based vs transient-based grouping . . . 45
5.5.4 Circle-based vs transient-based grouping . . . 46
5.5.5 Critical parameter value of a fault . . . 46

6 A Unified Manipulation Framework for Transistor-Level Languages . . . 49
6.1 Background . . . 49
6.1.1 Transistor-level Languages . . . 49
6.1.2 Code manipulation strategies . . . 50
6.2 State of the art on transistor-level manipulation tools . . . 51
6.3 Language independent framework . . . 51
6.3.1 Definition of a specific parser for each language . . . 52
6.3.2 Similarities between the Eldo and Spectre languages . . . 53
6.3.3 Framework internal structure . . . 54
6.4 Framework validation . . . 57

6.4.1 AST generation through grammars . . . 58
6.4.2 Semantic equivalence validation through simulation . . . 59
6.5 Framework applications . . . 59
6.5.1 Subcircuit wrapping . . . 60
6.5.2 Injection of defect models . . . 61
6.5.3 Print a netlist as XML/JSON file . . . 61
6.5.4 Visualize a netlist as a tree . . . 62

7 Verilog-A Implementation of Generic Fault Templates . . . 65
7.1 Background . . . 65
7.1.1 Transistor-level defect models . . . 65
7.1.2 Verilog-A language . . . 66
7.2 State of the art . . . 67
7.3 Verilog-A based fault templates . . . 68
7.3.1 Comparison with the state of the art fault templates . . . 69
7.4 Manipulation framework . . . 69
7.5 Experimental validation . . . 71

8 Abstraction Methodologies . . . 75
8.1 Abstraction of non-linear models (Transistor-level to Verilog-AMS) . . . 75
8.1.1 Simulation of analog circuits at different abstraction levels . . . 75
8.1.2 State of the art . . . 76
8.1.3 Automatic data driven behavioral model generation flow . . . 77
8.2 Abstraction of Piece-Wise linear models (Verilog-AMS to C++) . . . 79
8.2.1 State of the art . . . 79
8.2.2 Piecewise Linearization . . . 80
8.2.3 Abstraction methodology . . . 81
8.3 Abstraction of Non-linear models (Verilog-AMS to C++) . . . 87
8.3.1 Study of different linearization techniques for non-linear models . . . 87
8.3.2 Simulation of Non-linear models by numerical techniques . . . 88
8.3.3 Abstraction Methodology . . . 88

9 Application Of Methodology . . . 91
9.1 Integration into a Virtual Platform (VP) . . . 91
9.2 Abstraction Of Faults . . . 92

10 Conclusion and suggestions for future research . . . 95

Summary of the proposed innovative contributions . . . 97
10.1 Thesis related publications . . . 97
10.1.1 Fault Modeling and Analysis . . . 97
10.1.2 Fault Injection . . . 97
10.1.3 Model Abstraction . . . 98
10.1.4 Simulation . . . 98

References . . . 99

Appendices . . . 105

A Appendix A . . . 105

B Appendix B . . . 109

C Appendix C . . . 113

List of Tables

5.1 Critical values at different frequency points for the most significative OPAMP circuit faults, and their group. 46

8.1 State of the art on Data driven Behavioral Modeling. 77

8.2 Simulation results for the half-wave rectifier, the memristor, and the ideal relay. 87

9.1 Simulation results for the half-wave rectifier, the memristor, and the ideal relay. 92

List of Figures

2.1 Overview Of thesis . . . 15

4.1 Overview of the entire workflow. . . . 26
4.2 MOS cross-section with oxide trapped charges and interface traps. . . . 28
4.3 Proposed MOS stuck-on defect model. . . . 29
4.4 Proposed MOS stuck-off defect model. . . . 29
4.5 NMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off. . . . 30
4.6 Transistor-level description of the comparator model extracted from IEEE analog benchmarks suite. . . . 31
4.7 Fault equivalence plot relative to the P2427 defects and the proposed stuck-on/off defect models for the Operational Amplifier (OpAmp) circuit. . . . 34
4.8 Fault equivalence results between P2427 defects and the proposed stuck-on/off defect models for the Comparator (CoAmp) circuit. . . . 35
4.9 Circles generated for the faults #D125, #D152, #D20 and #D75 relatives to the OpAmp circuit. These circles are generated with the circle-fitting algorithm for two operating point (columns) and two frequency point (rows). . . . 36

5.1 Motivation behind fault grouping Work . . . 38
5.2 Overview of the proposed methodology. . . . 38
5.3 A 3-terminal circuit for S-parameter extraction. . . . 39
5.4 Implementation of a S-parameter port connected only with two terminals inside an analog circuit. . . . 40
5.5 Fault Equivalence . . . 41
5.6 Clustering trees . . . 41
5.7 AC trajectories of the fault D20 (MOS drain-source short on the transistor `MNM12`) at three frequencies and four operating points. . . . 45
5.8 Comparison between AC and transient-based grouping (with 15 cluster). . . . 45
5.9 Comparison between circle and transient-based grouping (with 15 cluster). . . . 45

6.1 Overview of the proposed manipulation framework. . . . 52
6.2 UML Class diagram of the internal structure of the framework - part 1. . . . 55
6.3 UML Class diagram of the internal structure of the framework - part 2. . . . 56
6.4 Transistor-level description of the OPAMP model extracted from IEEE analog benchmarks suite [1]. . . . 57
6.5 Abstract syntax tree of the OPAMP `xd1` component described in Eldo language. . . . 59
6.6 Abstract syntax tree of the OPAMP `xd1` component described in Spectre language. . . . 59
6.7 Comparison of the VOUT behavior for the transient simulation of the OPAMP described both in Eldo (identified by a blue line with crosses) and Spectre (identified by a purple line with circles). . . . 60
6.8 Proposed MOS stuck-on defect model. . . . 61
6.9 Proposed MOS stuck-off defect model. . . . 61
6.10 Graphical visualization of the netlist presented in Listing 6.9 produced through the proposed framework. . . . 63

7.1 Overview of the proposed framework that exploits the potentialities of the Verilog-A standard to describe fault templates injected directly into a transistor-level netlist. . . . 66
7.2 Some examples of faults in a N-Channel Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET). . . . 67
7.3 Representation of multi-level modeling. . . . 68
7.4 OPAMP output waveforms for the different faults injected on the `mpd11` Metal-Oxide-Semiconductor (MOS) component. . . . 73
7.5 OPAMP output waveforms for the different faults injected on the `mpd12` MOS component. . . . 74

8.1 Different Design and abstraction level [2]. . . . 76
8.2 Verilog-AMS Concepts. . . . 76
8.3 Proposed methodological flow divided into its macro-block parts. . . . 78
8.4 Data extraction. . . . 78
8.5 Piece-Wise Linearization . . . 80
8.6 Abstraction Methodology . . . 81
8.7 Proposed methodology flow to abstract Piecewise Linear (PWL) descriptions. . . . 82
8.8 Process flow used to abstract the model equations from Verilog-AMS to C++. . . . 82
8.9 Circuit simulation setup. . . . 85
8.10 Simulation results for the half-wave rectifier, the memristor, and the ideal relay. . . . 86
8.11 Curve Linearization Example . . . 87
8.12 Flow of abstraction of Non-linear Verilog-AMS models. . . . 88

9.1 Heterogeneous Platform. . . . 92
9.2 Abstraction Of faults. . . . 93

9.3 Abstraction Of faults using fault grouping. 93

List of Listings

6.1 Sketch of the Eldo parser written in Antlr4. 52
6.2 Sketch of the Spectre parser written in Antlr4. 53
6.3 Sketch of the Eldo grammar for the component X. 54
6.4 Sketch of the Spectre grammar for the instantiation of a component. 54
6.5 OPAMP subckt described in Eldo language. 58
6.6 OPAMP subckt described in Spectre language. 58
6.7 OPAMP subckt wrapped described in Eldo language. 61
6.8 XML fragment of the OPAMP subcircuit generated through our framework. 62
6.9 Simple netlist modeled with the ELDO language. 62
7.1 Templates for MOSFET fault models. 70
7.2 Custom Defectsim templates for short and open fault. 71
7.3 OPAMP subcircuit modeled in SPECTRE. 72
7.4 OPAMP subcircuit modeled in SPECTRE that contains the transformations in order to be able to use fault injection wrappers. 72
7.5 OPAMP1 testbench written with the SPECTRE language. 72
8.1 Non-linear model of a half-wave rectifier written in Verilog-AMS. 84
8.2 Verilog-AMS PWL model for a half-wave rectifier. 84
8.3 Verilog-AMS top level. 84
8.4 C++ analog branch structure. 84
8.5 C++ abstracted code of half-wave rectifier. Variable name that start with 'b_' are of type *analog_branch_t*. 85
8.6 C++ simulation manager. 85
8.7 Example of Verilog-AMS model simulated in C++ using numerical methods. 90
A.1 Volatge deadband amplifier. 106
A.2 memristor. 107
B.1 Templates for MOSFET fault models Part 1. 110
B.2 Templates for MOSFET fault models Part 2. 111
C.1 motor driven pendulum 114

List of Acronyms

AMS	Analog and Mixed-Signal
ANTLR	ANother Tool for Language Recognition
AST	Abstract Syntax Tree
CoAmp	Comparator
CPS	Cyber-Physical System
CUT	Circuit Under Test
DUT	Device Under Test
EDA	Electronic Design Automation
HDL	Hardware Description Language
IC	Integrated Circuit
ICPS	Industrial Cyber-Physical System
LTI	Linear Time Invariant
MOS	Metal-Oxide-Semiconductor
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
OP	Operating Point
OpAmp	Operational Amplifier
PWL	Piecewise Linear
SFG	Signal Flow Graph
SPICE	Simulation Program with Integrated Circuit Emphasis
VP	Virtual Platforms
VT	Voltage Threshold

1

Introduction

Functional safety ensures that the system will perform its associated safety measures when required. It is becoming more critical with more development in safety-critical applications including the automotive domain, e.g., Autonomous Driving and ADAS (Advanced Driver Assistance Systems)[3]. ISO 26262 [4] is a international functional safety standard for electrical and electronic systems(E/E). It was first published in 2011 and it as revised in 2018. According to ISO 26262, functional safety refers to the lack of unreasonably high risk from hazards brought on by E/E system malfunctions. It regulates the risk analysis, life cycle, safety concept development, validation activities, and safety management of automotive E/E systems.

A modern vehicle may consist of thousands of semiconductor chips. For safety-critical applications, analogue and mixed-signal (AMS) ICs play critical roles. The semiconductor industry's newest trend is applying advanced manufacturing nodes to automotive ICs. Analogue circuit defects will likely cause system failure with the shrinking device technology. Thus, it is critical to improving the functional safety of analogue circuits used in automotive. However, there are several challenges facing the functional safety of analogue circuits. Analog circuits are highly prone to parametric faults as compared to digital circuits. Lack of an industry-wide approved analog fault/defect model is another issue that analog functional safety must deal with. Although some progress has been made in the development of IEEE standards [5] for analog fault/defect models, however, there is no generally acknowledged model available in the industry.

Fault injection and simulation is also big challenge in a analog circuits and it is a basics of functional safety. It has become an indispensable tool for optimizing the diagnostic coverage of analog circuits to guarantee the functional safety [6, 4]. Fault grouping is way to find limited number of faults(by finding equivalence) for the injection to reduce the number of faults for injection in analog circuits. As opposed to an analog circuit, in a digital circuit, checking the equivalence of faults is easier to perform due to the discrete nature of signals (i.e., they can be either 0 or 1). But with analog Integrated Circuits (ICs), which are modeled using continuous values, equivalence between faults is more complex. The combination of the number of faults to consider and the length of analog simulations renders a fault-injection campaign very time-consuming and computationally expensive.

In safety critical applications circuits are analog and mixed. Analog circuits are core part of these systems. Analog circuits can be design and simulated at different abstraction level these are transistor, behavioral and functional level. The parent of transistor-level design languages is SPICE [7], the Simulation

Program with Integrated Circuit Emphasis (SPICE) [8]; after SPICE, many other proprietary design languages were developed to describe transistor-level designs. Each vendor has defined its variant of the SPICE language by extending it with the support of new functions and components, trying to satisfy designers' needs [9].. For most of them, the semantic matches those of SPICE, and only the syntax is changed. Others instead provide more default models or analysis tools. Consequently,a common commercial tool is required for simulating, analyzing, and especially manipulating all these languages. Another big challenge in analog circuits is the integration of analog and digital components into heterogeneous systems is crucial [10]. Generally, these heterogeneous systems composed of hardware descriptions and software applications are modeled as Virtual Platforms (VP) [11]. Complex embedded systems are usually modeled into the virtual platform using SystemC-AMS TLM or C++ languages. To model and integrate embedded systems into VPs, high-level descriptions of these components are required. These high-level descriptions of complex systems usually are not available for Integrated Circuit (IC) designers and generate it is an open research field. In this context, finding efficient methodologies to simulate heterogeneous VP at functional-level is fundamental. In this thesis different techniques are proposed to add some contribution to overcome the addressed challenges.

2

Objective

With the increase in the complexity of analog and mixed-signal circuits, guaranteeing the functional safety of both digital and analog circuits is fundamental to reducing every risk of failure in Cyber-Physical System (CPS) and Industrial Cyber-Physical System (ICPS) [12, 4]. In these complex systems, analog components are the core of its functionality. Research in analog defect modeling and simulation still has research gaps due to the unavailability of standards. Building efficient analog fault modeling, injection and simulation [13] techniques became strategic for manufacturers. In this context, the maturity of the techniques used in the analog field is lagging behind compared with the digital field. The *IEEE P2427*[5] work-group is currently making an attempt to standardize accounting methods for computing the defect coverage during fault injection in analog or mixed-signal circuits. SPICE-based simulators are still the core technology to simulate faults in analog circuits described at the transistor level. However, the circuits are becoming more complex as the number of transistors increases (see the reference circuit in the analog-benchmark suite [14]) thus, more evolved methodologies are required. In this context, my thesis work concentrates on functional safety of Cyber-Physical systems.

1. Main objective of this work is to support functional safety of Cyber-Physical systems (mainly analog part) see Figure 2.1 :
   a) By proposing and analyzing new fault models on different abstraction levels. These new defect models show complex behavior not covered from neither of the proposed IEEE P2427 hard defects for MOS transistors.
   b) By proposing and analyzing new analog fault templates written in Verilog-A following the IEEE P2427 defect models guidelines. These fault templates are suitable for all the simulators that support the Verilog-A language without the need to define different fault templates for each simulator.
   c) By presenting automatic fault injection manipulation framework. It is language-independent framework that allows manipulating transistor-level descriptions easily. The entire framework is open-source and available on GitHub.
   d) By investigating and implementing an automatic abstraction methodology. This abstraction algorithm supports both Piecewise Linear (PWL) and nonlinear models. These abstracted models can

be integrated into heterogeneous system for the faster simulation of system and for the abstraction of faults.

e) By implementing new fault simulation technique. This allow avoiding unnecessary waste of time and preserving safety metrics accuracy. This analysis allows to group faults with equivalent behavior to reduce the entire set of faults to simulate by selecting only one representative fault for each group.

2.1 Thesis Overview

Nowadays, it is essential to ensure the *functional safety* of Cyber-Physical Systems (CPSs). In these complex systems, analog components are the core of its functionality. Research in analog defect modeling and simulation still has research gaps due to the unavailability of standards. In ist part of thesis i have worked on proposing new fault models and templates for fault injection in analog circuits. The main commonly used defect models in analog circuits are opens and shorts, modeled through different parameter values of a resistance component. Motivation behind working this part was recent advances in the modeling of MOSFET gate defects suggest a more physical approach [15, 16, 17]. They consider an excess of fixed charges trapped in the gate-oxide layer or an excess of interface traps at its boundary as the cause of intrinsic faults in MOS transistors. The impact of these new fault models on typical analog circuits has not yet been reported. In this context, new physically-inspired MOS defect models are proposed and investigated their effects on analog circuits. The new defect models are simplified implementations of a physical model accounting for excess oxide charge and excess interface traps. The faulty circuit-level behaviors induced by the new defect models are compared with those induced by the hard defect models inspired from the P2427 draft standard, using transient simulations. In order to compare the impact of the injected defects in a generic fashion and to ensure that conclusions remain valid for the circuits under test used in other environments, albeit roughly under the same biasing conditions, *AC* matrices at multiple frequencies and various dynamic operating points during the transient simulations, and compare the obtained matrices for the various injected defects. Applying similarity measures to the collection of *AC* matrices, defects inducing similar faulty behaviors can be grouped together (grouping technique will be discussed later it is another contribution) and one representative defect can be elected for each class of similar induced faulty behaviors.

With respect to first part of thesis i also worked on proposing new Verilog-A based fault injection templates for transistor level analog circuits. The reason behind working this part was functional safety being increasingly important in the development of mixed-signal products, EDA solutions have appeared striving to help designers in the setup and execution of fault injection campaigns. Despite the on-going work to standardize the definition of defect models and coverage calculation methods in the IEEE P2427 draft standard, there is a lack of a unified and portable method to define fault templates which can be used to inject defects in an analog circuit. So in this context a Verilog-A based approach is proposed to coding fault templates, which through the compliance with the Verilog-A standard warrants portability across compatible simulators. The objective reasons for using AHDL based templates rather than the proprietary ones are : The fault

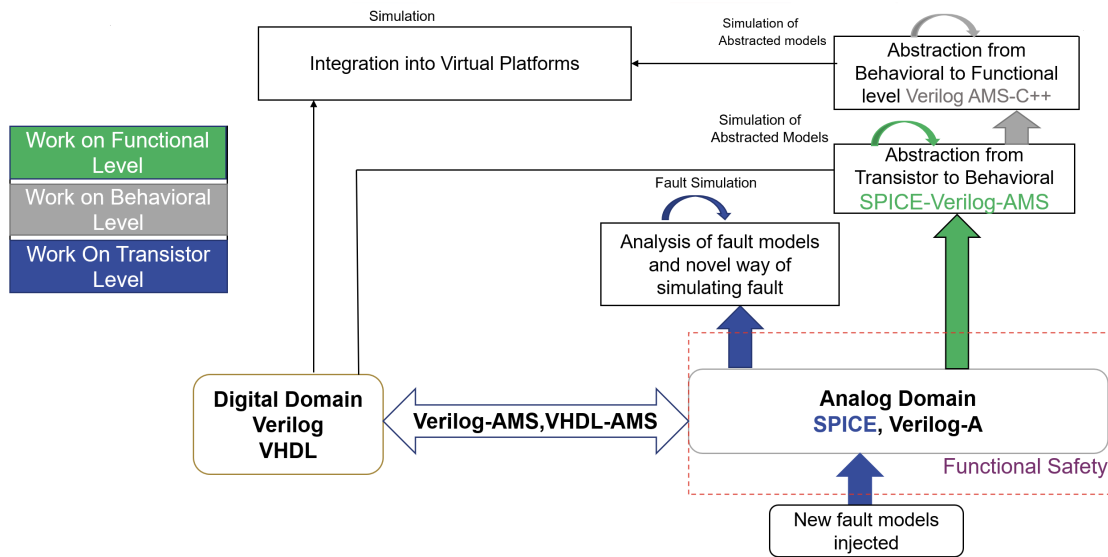


Fig. 2.1: Overview Of thesis

template can be made simulator independent. In fact, one can say the proposed template is simulator independent (depending on having the same instance parameters and model names for a component in multiple simulators). For the comparison DEFECTSIM [18]based templates are analyzed. This simulator works with Mentor Graphics Eldo and Questa ADMS simulators [7]. DEFECTSIM templates on the other hand depend on the syntax of the underlying simulator. The proposed fault templates have clean and simple syntax , one template is one module, each failure mode is a case statement. Unlike the DEFECTSIM templates which have an complex syntax mixing TCL, SPICE and other markers. Looking into simulation outputs, it is probably simpler to assess which fault was injected by looking at a dedicated instance parameter value than to inspect the master name. AHDL-based fault templates can issue messages into the simulation log file.

In second part of thesis i worked on fault simulation part for the analysis of previous proposed defects. For effective fault simulation (reducing number of faults to simulate) in analog circuits different techniques has been proposed. The promising approaches from the state-of-the-art aim at reducing the number of faults through techniques like *grouping* faults into classes and studying the *equivalence* between faults. As opposed to an analog circuit, in a digital circuit, checking the equivalence of faults is easier to perform due to the *discrete* nature of signals (i.e., they can be either 0 or 1). But with analog Integrated Circuits (ICs), which are modeled using *continuous* values, equivalence between faults is more complex. The combination of the number of faults to consider and the length of analog simulations renders a fault-injection campaign very time-consuming and computationally expensive. It has been found that several faults may induce similar faulty behavior at the circuit-block level. Identifying which group of faults induce the same failure mode allows defining fault-equivalence classes and reduces the simulation effort, as it is only necessary to check the coverage once for every failure mode at the chip or system-level. A complete characterization of faults, even at the circuit-block level, requires a significant computational effort [19]. It is common practice today to assess the equivalence of faults by comparing the results of test procedures applied in simulation to a Circuit Under Test (CUT). This analysis typically requires running a combination of several DC and tran-

sient simulations, a high time-consuming activity. To avoid this time consumption a novel predictive fault grouping based on faulty AC matrices is applied to group faults with equivalent behaviors. The proposed method is computationally cheap because it avoids performing DC or transient simulations with faults injected and limits itself only to faulty AC simulations. In this work, two grouping methods operating on AC data are compared: (1) grouping based on the comparison of feature vectors consisting of S-parameters, (2) grouping based on feature vectors constructed from the centers of the fault circles. The proposed fault grouping methods are validated with respect to a transient-based grouping. It also possible to group the faults based on the result of the circle-fitting method, but computationally more expensive. The proposed work's effectiveness is tested on circuits taken from the IEEE analog benchmarks [1].

As I have discussed previously research in fault injection also has research gaps. In this context, in next part of thesis i worked in fault injection and manipulation in analog circuits. For this a common manipulation framework is proposed . The motivation behind this work was manipulation of transistor-level descriptions has always been complex in the absence of proprietary tools and complicated in their presence. This led us to create a new framework for manipulating transistor-level languages, especially Eldo and Spectre the parent of transistor-level design languages is SPICE [7], the Simulation Program with Integrated Circuit Emphasis (SPICE) [8]; after SPICE, many other proprietary design languages were developed to describe transistor-level designs. Each vendor has defined its variant of the SPICE language by extending it with the support of new functions and components, trying to satisfy designers' needs [9]. Nonetheless, starting from any one of those transistor-level descriptions, it is possible to synthesize an electrical circuit to be printed on silicon. The extension of the SPICE language has been carried out by defining more electrical components (e.g., a MOSFET) that are aimed at modeling complex physical behaviors. Many of these design languages have been created by vendors who have created and marketed tools to manipulate them. These tools provide a programmatic way to manipulate these languages, but they are complex and difficult to use outside vendor-designed development environments. Furthermore, their internal functions are hidden and immutable, designers' needs must bend to what is available, or they need to open a ticket to the technical support.

The fourth part of thesis is related to work on abstraction methodologies at different abstraction level for PWL and non linear analog circuits. In electronic design and testing, the simulation speed of analog components is crucial. Moreover, the simulation of heterogeneous components embedded in a Virtual Platforms (VP) needs to be fast and accurate. Often, the analog components are non-linear, and simulating them is not easy to ensure the model's convergence. In this context, techniques for simulating linear circuits are stable and efficient, but there are still many research gaps for non-linear circuits. There are no systematic methods available to solve non-linear equations efficiently. One standard method is to solve these non-linear equations by describing them as a PWL models. PWL techniques approximate non-linear functions with a set of linear functions. This is common to most solver methods: they linearize to compute an inverse. Initially, i have worked on abstraction of PWL models form verilog-AMS to C++. To check the accuracy and the speed-up of these abstracted models, i have simulated them and compared them with SPICE-based simu-

lators. Furthermore, the final C++ models are embedded in a VP to demonstrate how to create a functional simulation of an intelligent system. Further, for the non-linear models abstraction different techniques are analyzed and discussed .

2.2 Thesis Organization

This thesis consist of ten chapters. These chapters are organized as follows:

- **Chapter 1** explain the introduction of thesis;
- The present **Chapter 2** describe objective of this work, Overview of thesis and Organization of thesis;
- **Chapter 3** presents the state of the art and background to better understand the concepts of functional safety;
- **Chapter 4** will presents the work on novel proposed fault that is Investigation on Realistic Stuck-on/off Defects to Complement IEEE P2427 Draft Standard ;
- **Chapter 5** will explain the novel fault grouping based on AC analysis technique that is computationally cheap ;
- Next, **Chapter 6** will be related to analog fault injection tool that is available on github for the manipulation of transistor level netlists ;
- **Chapter 7** will explains Verilog-AMS templates for fault injection ;
- **Chapter 8** will discuss about abstraction methodologies for PWL and non linear analog circuits. This work is still on going work flow of methodologies are discussed here ;
- **Chapter 9** explain applications of proposed methodologies ;
- Finally, in **Chapter 10**, the conclusion and recommendations for future work are discussed ;

3

Background and State Of The Art

This chapter is subdivided into two parts: The first part briefly explains the background of different concepts used throughout in thesis to better understand the work . The second part introduces the state-of-the-art related to analog fault modeling , injection and grouping.

3.1 Background

This section presents the definitions related to analog faults and essential knowledge of the different classes of possible faults.

3.1.1 Some Basic Terminologies

- **Defect**: unwanted modification of a physical structure likely to impact the function of a subsystem;
- **Fault**: a model of the impact of a defect at component level in a certain discipline, e.g., a drain/source short, a transistor stuck on/off;
- **Failure mode**: deviant behavior of a subsystem that may cause the system to fail to execute its intended function, e.g., an operational amplifier output that oscillates, or that has a voltage or current offset concerning the fault-free behavior;
- **Fault-site**: each component instance that can be affected;
- **Fault coverage**: percentage of fault detected during the fault injection campaign.

These terms are used in the context of transistor-level circuit realization.

3.1.2 Transistor Level defect models

Adopting the terminology being standardized in the IEEE P2427 draft, a *fault* is considered here to be an unexpected change in the performance of a circuit, whilst a *defect* is an unexpected modification of a physical structure likely to impact the function of a subsystem and cause a *faulty behavior*, a *failure mode* or, equivalently, a *fault*.

The IEEE P2427 draft standard, proposes to distinguish between *hard* and *parametric* defects, the latter being associated with a progressive parametric deviation and considered to be more difficult to detect. *Hard*

are defects considered to cause a permanent change in the circuit's topology, and are often associated with the application of basic *short* or *open* defects. For MOS transistors, *hard* defects may be further characterized by their effect on the main current path between the transistor's anode (drain) and cathode (source), and split accordingly into:

- Defects creating a permanent conductive path between anode and cathode, either directly, such as a *drain-source short*, or indirectly, such as a *drain-gate short*.
- Defects preventing any current to flow between anode and cathode, either directly, such as a *source open* or a *drain open*, or indirectly, such as *a gate-body short* or a *gate-source short*.
- Defects causing a loss of control on the transistor state, such as the *open gate*.

3.1.3 Verilog-A language

The design of integrated circuits is complex, and it is challenging for engineers to characterize complex behaviors. Hardware Description Language (HDL), such as Verilog and VHDL, provide various levels of abstraction to design Integrated Circuit (IC). The system behavior can be described at a high level, then gate-level descriptions can be generated using simulation synthesis programs. The interaction between digital and analog signals can be managed with mixed-signal languages. Verilog-A is a subset of Verilog-AMS [20], and it describes the behavior of analog signals with the additional functionality of interfacing some of the behavior of digital signals. Analog behaviors of the conservative systems can be described at a high level with Verilog-A, which supports conditional statements, usually part only of the programming languages. Verilog-A is supported by many Electronic Design Automation (EDA) vendors, so different descriptions written with this language can be simulated with different commercial tools. By exploiting the potentialities of Verilog-A, new defect models can be developed rapidly at the behavioral level and can be simulated into SPICE designs, as proposed in this article.

3.1.4 Simulation of analog circuits at different abstraction levels

Transistor Level Simulation

Analog circuits can be simulated at different abstraction levels. For transistor-level simulation, there are some tools [21, 9, 7] available. A netlist that represents the model is required for the simulation of analog circuits. SPICE [8] has been a widely used simulator in the past for detailed transistor-level simulation. However, it is efficient only for designs with few transistors. The modern circuits are complex and contain many million transistors; therefore, new methodologies are required to speed up and robust the simulations.

Behavioral Level Simulation

Analog circuits can be simulated at behavioral level [22] to speed up the simulation. Behavioral models can be a set of equations that describe the behavior of a circuit. The most used behavioral modeling languages

are Verilog-A, Verilog-AMS, VHDL-A, VHDL-AMS, and System-C AMS. The acronym AMS stands for Analog/Mixed-Signal descriptions that combine digital and analog components. In this article, the proposed methodology is based on the Verilog-AMS language. Verilog-AMS is a Hardware Description Language (HDL) specifically designed to describe mixed-signal systems. It can be considered as an extension and combination of Verilog-A and Verilog-HDL [20].

Functional Level Simulation

Functional level simulations can be performed using C++ and SystemC-AMS TLM languages [23]. Functional level simulations are closer to actual hardware as compared to behavioral modeling. High abstraction level models are required to improve the simulation speedup, primarily when many simulations are used to test a large set of faults.

3.2 State of the Art

3.2.1 State of the Art in Defect Modeling and Injection

In the literature, many fault modeling and injection techniques have been proposed for effective fault simulation campaigns. A mixed signal fault injection technique has been proposed in [24], in which the Verilog-A language is used to develop fault models directly onto the circuit devices. The authors present an approach based on pre-processor commands that involve mixed-signal co-simulation of Verilog/VHDL with SPICE. These pre-processor commands can only be set as arguments to the call to the simulator and not by using the simulator sweep command or instance parameters as proposed in our work. In [25], behavioral fault modeling of CMOS circuits has been proposed for fault simulation. In the above-mentioned work, some parts of the circuits and faults are modeled using the VHDL-AMS language. In [26], authors selected a wide range of short and open defects with different resistance values. Open and short defects have been investigated by injecting the range of $1k$, $2K$, $5K$, $10K$, $100K$, ∞ Ω, and 1, 100, $1K$, $2k$, $5K$ Ω. A new open-gate model for DC simulations has been proposed in [27] after analyzing the open defects in analog circuits. This proposed model was based on voltage-dependent voltage sources, which keep the transistor in the sub-threshold region, as concluded from the in-depth analysis. In [28], authors presented an overview of different fault model techniques present in the literature. According to this article, there is no industry-accepted technique for efficient fault modeling and injection in analog circuits. The impact of various defects including open, short, extreme variation, or user-defined defects can be analysed in a defect simulator called Tessent DefectSim [18]. This simulator works with Mentor Graphics Eldo and Questa ADMS simulators [7]. Tessent DefectSim first uses a random selection technique to inject the defects into circuits, then efficiently simulates and analyzes the results across multiple testbenches. For finding the value of the fault parameter of fault Tessent Defectsim uses a relative likelihood algorithm [29]. The main limitations of these approaches are related to the fact that they are not portable between different simulators.

3.2.2 State of the art on transistor-level manipulation tools

Nowadays, there are many tools used to simulate the transistor level circuits [21, 30, 7]. Every tool supports there own syntax/format of language for the simulation. Usually, these languages have standard semantics but different syntax. Researchers in industry and researchers are working on manipulating and converting these languages from one to another. After the definition of many transistor-level languages created from the original SPICE language, few tools were developed that allow the manipulation and conversion of these languages from one to the others. These tools can convert libraries and circuits between these transistor-level languages, e.g., Spectre to Eldo converter(spect2el), Netlist Translator for SPICE, and Spectre [31]. Spectre to Eldo converter is available in the Eldo simulator that allows converting a Spectre circuit to the equivalent circuit described in Eldo. However, there are some limitations of using spect2el. Firstly, it does not support all the statements of the Spectre language, e.g.; if some sections are written in spice spect2el, the tool can not correctly translate the circuit. Secondly, conversion is uni-directional, only from Spectre to Eldo [30], and the other problem is that some features require manual effort for the modifications. Both Eldo and Spectre languages have some similarities but differ in many aspects. Currently, there is no available framework that provides unified semantics. Above mentioned Netlist translator [31] was developed to simulate the Spectre and Simulation Program with Integrated Circuit Emphasis (SPICE) (Spice2, Spice3, PSpice, and HSpice) in Advance Design System (ADS). In [32] a python interface for SPICE-based simulations was presented. The purpose of that interface is to help the designer in the industry solve circuit sizing problems of integrated analog CMOS circuits for SPICE-based simulations

3.2.3 Fault Grouping

Several analog fault simulation and grouping techniques are presented and compared with our methodology in this section. In [33] the authors discussed fault grouping based on hierarchical clustering on the faulty transient waveform. They showed that the grouping they realized allows predicting the final classification of faults in functional safety categories reliably. This result constitutes the *golden reference* of the present article, as our method strives to reproduce the same grouping as obtained from transient waveforms but by alternative much less computationally intensive means. In [34], the authors show the limitations of the random sampling [35] for choosing the faults to be simulated in an analog circuit and highlight the necessity to consider the uncertainty of the fault parameter before simulating a fault. For that reason, in this work, we try to prioritize faults by applying an approximated grouping on the faults and taking the *central* component as representative fault for each group. With *central* component we mean the one that minimizes the distances to all the other group members. We also analyze the uncertainty of the fault parameter value that the representative faults assume by calculating a *critical value* for it. The *critical value* for a fault leads to an AC parameter that has the potential to deviate the most from its fault-free value. In [36] authors presented a dynamic fault grouping algorithm for non-linear circuits. This methodology dynamically divides the list of faults into groups based on the transient behavior. Different time steps are used for different groups in simulation; however, the time step for faults within the same group stays the same. Another fault

list compression technique is presented in [37], which uses stratified fault grouping for the identification of representative faults. Most of the techniques mentioned above are based on transient simulations, which is renowned for being time-consuming. Other techniques have explored different solutions that tend to reduce the number of transient simulations. Frequency-based analysis was presented in papers like [38] and [39]. In [19] authors presented a fault clustering technique combining faulty DC Operating Point (OP) and frequency domain analysis. That technique requires N transient simulations (i.e., one for each fault), and then, at each OP they perform an AC analysis.

Fault injection and simulations tools

In literature, some articles describe tools for automatic fault injection and simulation [40]. Legato Reliability Solution has been introduced by Cadence to support analog defect simulation. It has been built on top of P2427 which offers various options to designers to accelerate the defect simulation and allows for exploring systems testability. When P2427 is further developed, new upgrades will be added to the tool, and other fundamental methods to increase simulation efficiency will be investigated [41]. In [42], an open-source tool that can be used for transistor-level fault injection has been proposed. The impact of user-defined, open, short, and extreme variation defects modeled within schematic netlist or layout-extracted can be measured in Tessent DefectSim [18], which works with Questa ADMS and Mentor Graphics Eldo circuit simulators. Tessent DefectSim first uses a random selection technique to inject the defects into circuits, then efficiently simulates and analyzes the results across multiple testbenches. For finding the value of the fault parameter of fault Tessent Defectsim used a relative likelihood algorithm [29]. In [43], an interface for analog fault injection and simulation based on saber has been discussed. Saber is one of the most powerful and popular EDA software for simulating digital and complex analog integrated circuits. It contains a comprehensive model base for various types of circuit components.

In [13] authors proposed another way flow for the efficient simulations of defects according to rules suggested by the IEEE P2427 draft standard. MATLAB-Simulink-based framework for fault simulation for linear analog circuits has been proposed in [44]. The simulation is initiated by identifying the type of fault and then constructing a Signal Flow Graph (SFG) of the corresponding faulty circuit. In [45], another framework SLIDER has been proposed for the injection and simulation of layout level defect injection. SLIDER offers fully automated defect generation and injection, which makes it simple to apply to various designs.

4

Investigation of Realistic analog Stuck-on/off Defects

I have started from the modeling of analog fault models. Recent progress in understanding the physical mechanisms responsible for the temporal degradation, including aging, of semiconductor devices has been driven by the rising concern about the reliability of advanced CMOS technologies. Negative-bias temperature instability (NBTI) and channel hot carrier (CHC) damage are two common mechanisms that cause the degradation and aging of p-channel and n-channel semiconductor devices, respectively. These effects are primarily attributed to the accumulation of trapped charges and/or trap buildup near the interface between the semiconductor and the oxide layer [46]. NBTI affects p-channel MOSFETs by accumulating positive charges in the oxide layer that trap mobile holes in the p-channel and lead to a shift in the device's threshold voltage. CHC damage affects n-channel MOSFETs by injecting hot carriers into the channel region of the device, creating traps at the semiconductor/oxide interface and impacting the electrical characteristics of the device over time. To mitigate these aging effects, device design and operating conditions can be optimized, and advanced materials and processing techniques can be used to minimize charge trapping and trap buildup. The reason behind this discussion is we also needs to consider such behaviors while fault modeling. Historically, the concept of stuck-on/off defect did not originate from physical observations but rather to model the behavior of faults at the gate level.

Digital stuck-at fault models where a transistor is considered frozen in on-state or off-state may not apply well on analog circuits because even a slight variation could create deviations of several magnitudes. This implies that standard stuck-at faults would not be general enough for analog behavior, i.e., the transistor will not be stuck to have $dI/dVg = 0.0$. Recent works have suggested to consider physical phenomena, like modeling oxide defects into the transistor, less abstract with respect to digital stuck-at fault and analog stuck-on/off defect.

This work focuses on evaluating faults proposed by the *IEEE P2427 draft standard* [5], which is still a work-in-progress standard. Figure 4.1 shows the proposed workflow. Moreover, a novel approach is presented for modeling realistic stuck-on/off defects based on oxide defects. To investigate the impact of these faults on the circuit on two designs are taken from the IEEE analog-benchmarks circuit collection [1]: an Operational Amplifier (OpAmp) and a Comparator (CoAmp). Furthermore, a novel method is applied that relies on AC matrices extracted at several operating points and combines it with a circle-fitting technique to compare faults with uncertain parameters.

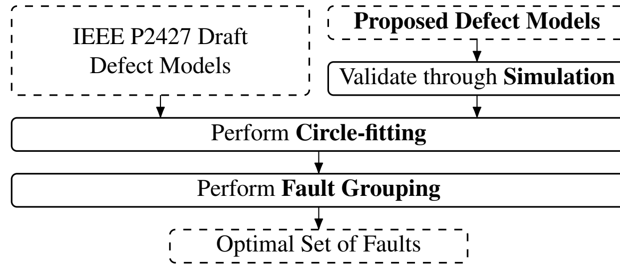


Fig. 4.1: Overview of the entire workflow.

4.1 Background

4.1.1 Transistor-level defects

Adopting the terminology being standardized in the IEEE P2427 draft, a *fault* is considered here to be an unexpected change in the performance of a circuit, whilst a *defect* is an unexpected modification of a physical structure likely to impact the function of a subsystem and cause a *faulty behavior*, a *failure mode* or, equivalently, a *fault*.

The IEEE P2427 draft standard, proposes to distinguish between *hard* and *parametric* defects, the latter being associated with a progressive parametric deviation and considered to be more difficult to detect. *Hard* are defects considered to cause a permanent change in the circuit's topology, and are often associated with the application of basic *short* or *open* defects. For MOS transistors, *hard* defects may be further characterized by their effect on the main current path between the transistor's anode (drain) and cathode (source), and split accordingly into:

- Defects creating a permanent conductive path between anode and cathode, either directly, such as a *drain-source short*, or indirectly, such as a *drain-gate short*.
- Defects preventing any current to flow between anode and cathode, either directly, such as a *source open* or a *drain open*, or indirectly, such as *a gate-body short* or a *gate-source short*.
- Defects causing a loss of control on the transistor state, such as the *open gate*.

Interestingly, there is still an on-going debate whether transistor *stuck-on* or *stuck-off* defects as generated by excess oxide charges or interface traps should be considered *hard* or *parametric* defects. In particular, one may wonder whether transistor fault models generated by the application of basic *short* or *open* defects may "cover" the behavior induced by excess oxide charges or interface traps in MOS transistors as physically described in [15].

Considering that *stuck-on* or *stuck-off* transistor defects due to excess oxide charges or interface traps are defects that, unlike the *hard* defects listed previously, change the intrinsic characteristics of the transistor, rather than modifying its connectivity to the circuit, one may already suspect that their induced faulty behavior will be "specific" or "distinguishable" from defects induced by the application of generic *opens* and *shorts*.

4.1.2 AC matrices and S-, Y- or Z-parameters

Linearization of analogue circuits at multiple operating points is a known method to characterize their behavior, capable of capturing even non-linear aspects [47, 48]. The technique will be applied here to compare the faulty behavior of circuits induced by injected defects.

Most modern simulators support an *AC* analysis mode, where the circuit is linearized at an operating point and responses are computed in the frequency domain (the imaginary axis of the Laplace domain). This analysis mode has the advantage of being computationally very efficient, and allows to extract complex matrices, often called "*AC matrices*", representing the small-signal input-output transfer coefficients of a circuit under test at a series of specified frequencies.

Obtaining these *AC matrices* is conditioned by the definition of *ports*, often modeled as independent sources, which uniquely identify how the quantities to be used as input or output signals are to be measured in the circuit. Taking *AC port* voltages as input signals and *AC port* currents as output signals yields admittance or Y-parameter matrices. Reversing roles yields impedance or Z-parameter matrices. Scattering or S-parameter matrices result from taking incoming traveling waves as inputs and reflected traveling waves as outputs [49]. The vector of traveling waves can be obtained as a linear transformation of the vector of *AC port* voltages and currents, assuming some reference impedance Z_{ref} for each port.

4.1.3 Defect Modeling in analog circuits

The modeling of defects in analog circuits for injecting them into analog circuits is still the topic of ongoing discussions in the design and testing communities, involving both researchers and industrial contributors. Generally, faults are modeled by adding a resistor component with a fixed value into an electrical network. The IEEE P2427 work-group is currently making an attempt to standardize accounting methods for computing the defect coverage during fault injection in analog or mixed-signal circuits [5]. The IEEE P2427 draft standard considers two main classes of defects: *hard* and *parametric* defects. In [25], behavioral fault modeling of CMOS circuits has been proposed for fault simulation. In the above-mentioned work, some parts of the circuits and faults are modeled using the VHDL-AMS language. In [26], authors selected a wide range of short and open defects with different resistance values. Open and short defects have been investigated by injecting the range of $1k$, $2K$, $5K$, $10K$, $100K$, ∞ Ω, and 1, 100, $1K$, $2k$, $5K$ Ω. A new open-gate model for DC simulations has been proposed in [27] after analyzing the open defects in analog circuits. One common problem in analog fault-injection is that the fault parameter's value is unknown or not accurately known. It is unknown what is the correct value to assign to an open or a short resistance. The P2427 IEEE standard [5] provides hints, but the ranges are typically wide (short: $0 - 200k\Omega$, open: $100M\Omega - \infty$). A new open-gate model for DC simulations has been proposed in [27] after analyzing the open defects in analog circuits. A physical defect-oriented compact model for MOS transistors was presented in the work of Esqueda *et al.* [15]. These authors analyzed the effect of an excess of trapped charge in the oxide and an excess of interface traps into a MOSFET. Our proposed stuck-on/off defect models implemented at the transistor level are based on the concept of excess of trapped charge in the oxide and

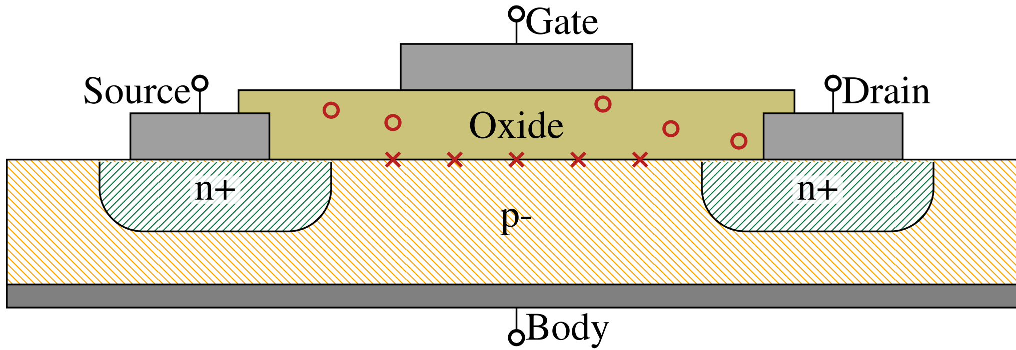


Fig. 4.2: MOS cross-section with oxide trapped charges and interface traps.

excess of interface traps. With the number of defects to account for being related to the number of possible topological changes that are possible (or probable) in the circuit considering opens and shorts and eventually some parametric faults for some transistor types on top, the size of the defect universe to consider for an analogue circuits very quickly grows to large numbers in function of the number of components in the circuit. In the real world, the distribution that a fault parameter value may assume is related to uncontrolled features of the fabrication process. As a result, significant variations of fault parameters should be expected across all occurrences of defects, and one needs to account for a broad distribution of fault parameter values when attempting to characterize defects or their impact [50].

4.2 Proposed stuck-on/off defect models

In proposed work the investigation is driven by the intuition that none of the *hard* defects as envisioned by the IEEE P2427 draft standard seem to naturally "cover" the transistor *stuck-on* or *stuck-off* behaviors which are known to occur in MOSFET's and are caused by excess trapped charge in the gate oxide or excess interface states [15] [51]. With analog circuits in mind, this work focused on physically-inspired defect models, which render the effect on transistor characteristics of excess gate-oxide charges or excess interface traps in a realistic fashion.

As illustrated in figure 4.2, *trapped charges*, marked in red "○", are fixed positive charges which are trapped in the gate oxide and have as primary effect to shift the threshold-voltage. *Interface traps*, represented with red "x", are parasitic states which can capture and release minority carriers normally intended to populate the channel but which get blocked in the trap and can not participate to the current conduction in the channel anymore. Traps reduce the efficiency of the MOS gate to attract carriers into the channel for conduction.

They manifest themselves as a reduction of the sub-threshold slope and a shift of the threshold voltage. Esquada and Barnaby [15] chose to model the effects of excess fixed oxide charges and excess interface states in the form of a bias-dependent voltage source placed in series with the gate of the original transistor model. However the resulting model references many parameters typically used in MOS transistor models and is not particularly appropriate for use in automatic fault injection templates. Proposed work adopted an alternative but otherwise equivalent approach based on the equivalent circuits shown in figures 4.3 and 4.4.

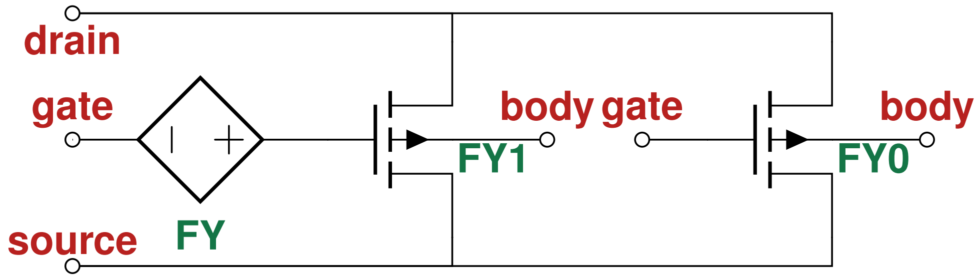


Fig. 4.3: Proposed MOS stuck-on defect model.

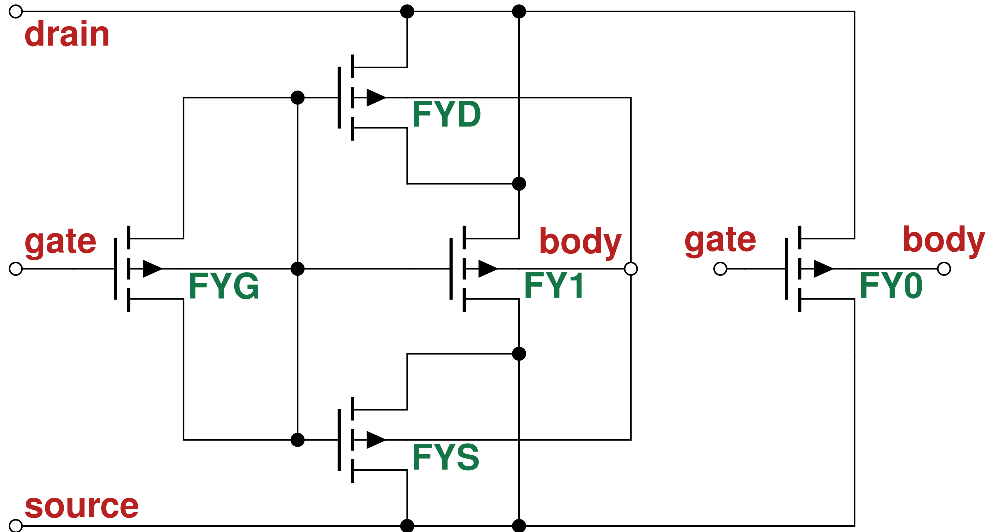


Fig. 4.4: Proposed MOS stuck-off defect model.

The equivalent circuit shown in figure 4.3 models the effect of excess oxide-trapped charges. These charges being generally of positive type, have the following effects on MOS transistors :

- For N-type MOS, more oxide charges reduce the Voltage Threshold (VT) and cause a current increase;
- For P-type MOS, more oxide charges increase the magnitude of the VT (more negative) and cause a current decrease.

The equivalent circuit shown in figure 4.4 models the effect of excess interface traps. A capacitive divider with `FYG` is used to reduce the voltage swing by half on the gate electrode of `FY1`, mimicking the efficiency reduction caused by the interface states. In order to maintain the overall capacitance values at the original level, dummy MOS capacitors and `FYS` and `FYD` are added. The net effect of these additions on the MOS transistor model is as follows:

- On N-type MOS, more traps increase the VT and cause a decrease of the current;
- On P-type MOS, more interface traps increase the magnitude of the VT (more negative) and cause a current decrease.

The equivalent circuits of figures 4.3 and 4.4 further also contain a fault-free instance labeled `FY0`. This instance is used in conjunction with the faulty `FY1` instance to allow a modulation of the defect model by a fault parameter `FP` which ranges between 1 and 0. The defect modulation is achieved by specifying a multiplier $M_{Y1} = FP$ on the faulty `FY1` instance and a multiplier $M_{Y0} = (1 - FP)$ on the fault-free

instance, realizing in effect a convex interpolation between the two cases. The parameter `FP` represents the fraction of the MOSFET affected by the defect, and will be used here to generate fault-circles.

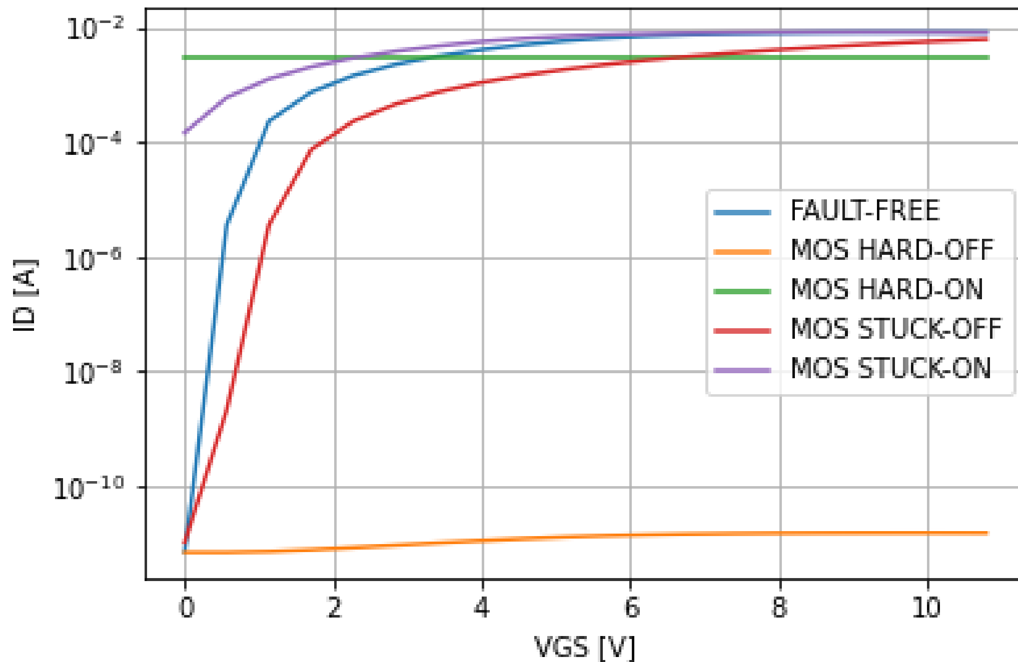


Fig. 4.5: NMOS ID(VG) characteristics for fault-free, hard-on/off, and stuck-on/off.

Figure 4.5 illustrates the behavior of various defect models in the case of the a N-type MOSFET. The defect for excess fixed oxide charges is labeled "*MOS stuck-on*", as this is the behavior which most closely approximates the effect of those charges. The defect corresponding to excess interface charges is labeled as "*MOS stuck-off*" as the current in this case is substantially (10x) lower than that of the fault-free case.

The figure 4.5 further also references two alternative "naive behavioral" implementations of MOSFETs defects:

- "*MOS hard-off*", where the gate is forced in the off-state by disconnecting the gate terminal and forcing it to follow the potential of the body;
- "*MOS hard-on*", where the gate is forced in the on-state by disconnecting the gate terminal and forcing it at a suitable potential difference above the body terminal.

4.3 Admittance-based analysis for fault grouping

To analyze the proposed faults faults needs to be simulated. This section describes the steps necessary to simulate and group faults with the same frequency behavior. Fault grouping is a complementary technique, allowing to partition the defects injected at block level into classes of defects inducing similar behavior. A novel fault grouping technique is also a part of thesis this will be explain in next chapter with detail. The explanation is subdivided into four sections. The *first* part describes how to instrument a circuit with the

S-parameter ports. The *second* section explains the impact of a fault on the admittance matrices. The *third* section describes the circle fitting method. Finally, the *last* part explains how to verify the exact equivalence of two faults by exploiting the circle fitting method.

4.3.1 Circuit instrumentation

All the analyses presented in this section are carried out on the simulation traces retrieved from the simulation. A circuit instrumentation phase is required to inject the S-parameters and the faults correctly to retrieve these simulation traces. A transient simulation of the Circuit Under Test (`CUT`) is performed for each fault; then, a frequency analysis is performed to retrieve the admittance matrices at specific operating points utilized to linearize the circuit.

The frequency analysis at specific operating points allows to study the state space around the operating point, where the linear model is valid [47]; For each port of the `CUT`, a standard S-parameter probe is injected by connecting it to a switch. In this switch, the `CUT` port is connected with the circuit under analysis, while the `ENV` port is connected to the rest of the circuit (the environment). During transient simulation (S-parameter disconnected), the switch closes towards the environment, while it closes towards the S-parameter port during the frequency analysis (S-parameter connected) [52]. This behavior of the switch allows activating the probe only during the frequency analysis. While injecting the faults in the CUT and retrieving the necessary data to use the circle-fitting method requires a double opposite switch. The switches can switch between the faulty case and the fault-free case. The two switches allow injecting a fault in an existing connection, select the fault-free case during the transient simulation and the faulty case during the frequency analysis. This condition is necessary to avoid the considered operating point (at which the frequency analysis is performed) change to another operating point. This condition is mandatory to analyze the admittance data with the circle-fitting method to avoid misleading results.

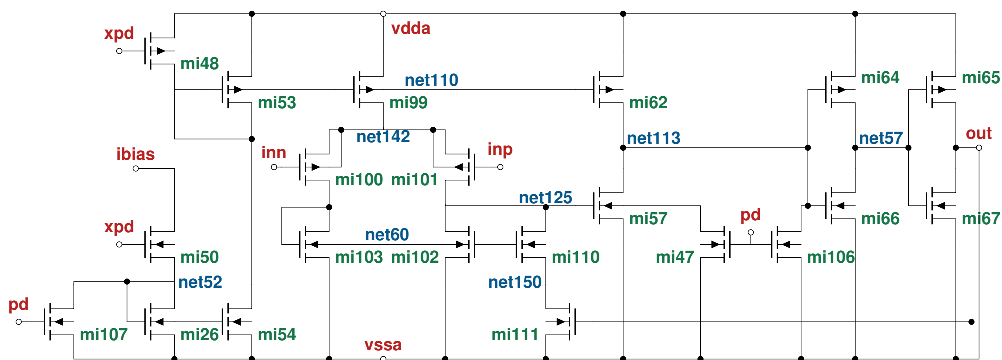


Fig. 4.6: Transistor-level description of the comparator model extracted from IEEE analog benchmarks suite.

4.3.2 Impact of a fault on the admittance matrices

The admittance matrices can be easily calculated from the AC matrices (composed of complex values) retrieved through the S-parameter ports [53]. Assume a circuit with `N` pins represented by its admittance

matrix: $I_N = Y_N \cdot V_N$ where I_N and V_N are vectors of pin currents and pin voltages response. Imagine one branch inside with a faulty conductance g_f between nodes p and n. It is possible to formulate a nodal admittance matrix $Y_N + 2$ the circuit with two additional pins such that the fault is externalized as described in the Equation (4.1).

$$\begin{bmatrix} I_1 \\ \vdots \\ I_N \\ I_p \\ I_q \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Y_{PP} \end{bmatrix} & \begin{bmatrix} Y_{PF} \end{bmatrix} \\ & \\ & \\ \begin{bmatrix} Y_{FP} \end{bmatrix} & \begin{bmatrix} Y_{FF} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ \vdots \\ V_N \\ V_p \\ V_q \end{bmatrix} \tag{4.1}$$

The previous system can be simplified taking the Equation (4.2).

$$V_f \triangleq V_p - V_q \text{ and } I_f \triangleq 1/2(I_q - I_p) \tag{4.2}$$

Then substituting the Equation (4.2) in the Equation (4.1) are derived the Equation (4.3) that represent the circuit behavior.

$$\begin{bmatrix} I_1 \\ \vdots \\ I_N \\ -If \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Y_{PP} \end{bmatrix} & \begin{bmatrix} Y'_{PF} \end{bmatrix} \\ & \\ \begin{bmatrix} Y'_{FP} \end{bmatrix} & \begin{bmatrix} Y'_{FF} \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} V_1 \\ \vdots \\ V_N \\ V_f \end{bmatrix} \tag{4.3}$$

The fault model is defined in the Equation (4.4).

$$I_f = g_f \cdot V_f \tag{4.4}$$

The impact of the fault on the matrix Y_N can be obtained by solving for V_f as depicted in the Equation (4.5).

$$Y_N(gf) = Y_{PP} + Y'_{PF} \cdot (gf - Y'_{FF})^{-1} \cdot Y'_{FP} \tag{4.5}$$

The previous equation means that: at an frequency, each element of the admittance matrix can be formulated with the Equation (4.6).

$$Y_{i,j}(gf) = Y_{PPi,j} + \frac{Y_{Xi,j}}{g_f - Y'_{FF}} \tag{4.6}$$

Where $Y_{PPi,j}$, $Y_{Xi,j}$ and Y'_{FF} are complex constants, independent of the value of g_f.

4.3.3 Circle-fitting method

By considering a fault site and varying a fault parameter, it is possible to identify a circle in the frequency output results [54]. The circle equation can model any continuous parameter shifting (soft) or hard fault to linear and piece-wise linear circuits from the admittance matrices. S-parameters are defined as a bi-linear transformation of admittance parameters. These transformations are known in the RF domain to

create circles or lines in the Cx plane. The circle law had been mainly proven till now for impedance or admittance. It is possible to automatically discover how many circles are needed to fit the data points nicely with circle fitting algorithms. With only two injections of the uncertain fault with two distinct values of the parameter plus the fault-free case, it is possible to determine the entire fault distribution.

After sweeping the fault parameter, it is possible to identify one circle (described with a minimum of three complex values in the plane) by analyzing the admittance matrices by interpolating with all the points generated with the parameter's sweeping. Suppose the circuit is simulated with different frequencies. In that case, it is possible to identify a circle that precisely represents the fault parameter distribution for each frequency in the complex plane. When the fault parameter's sweeping changes the current operating point, other complex points are required to identify new circles that correctly fit the fault parameter's new distribution. The fault circles for the admittance matrix elements represent the signature of the uncertain fault [52]. They may be used to compare potentially equivalent faults despite variations of their respective fault parameters. When two faults are identified by the same circle in the *Cx* plane, then we need to simulate only one of them because both faults are inducing the same behavior on the output of a circuit. The fault grouping described in Section 4.3.4 exploits the circle-fitting to reduce the set of faults to be simulated.

4.3.4 Fault grouping method

Comparing fault circles retrieved through the circle-fitting method allows group faults in equivalent classes. Suppose two faults have the same distribution of the fault parameter values. In that case, they are equal because all the possible behavior of the faults is identified with a circle in the complex plane. This comparison between faults must be made at fixed operating points.

Because if a different value of the fault parameter induces a new operating point, then a new circle must be considered. Some additional simulations are needed to characterize that new circle. Faults with the same circles described in the complex plane are equivalent. Based on the circles produced from the admittance matrices, it is possible to cluster the faults based on the feature vectors. These vectors are constructed from a distance calculated between the centers of the fault circles. Consequently, the number of all possible faults to simulate is reduced.

4.4 Experimental Validation

4.4.1 Analog circuits analyzed

The proposed workflow is validated by using transistor-level descriptions extracted from the analog-benchmark circuits collection [1]. In particular, the Circuit Under Tests (CUT) is an operational amplifier (OpAmp, depicted in [52]) and a comparator (CoAmp, depicted in Figure 4.6). Principal circuit elements of the CoAmp and the OpAmp are NMOS and PMOS transistors, diodes, JFET transistors, and capacitors.

In these transistor-level descriptions, many defect-site locations are possible. All the defects proposed in the P2427 draft standard, plus two new stuck-on/off defect models, are used to validate the workflow. In

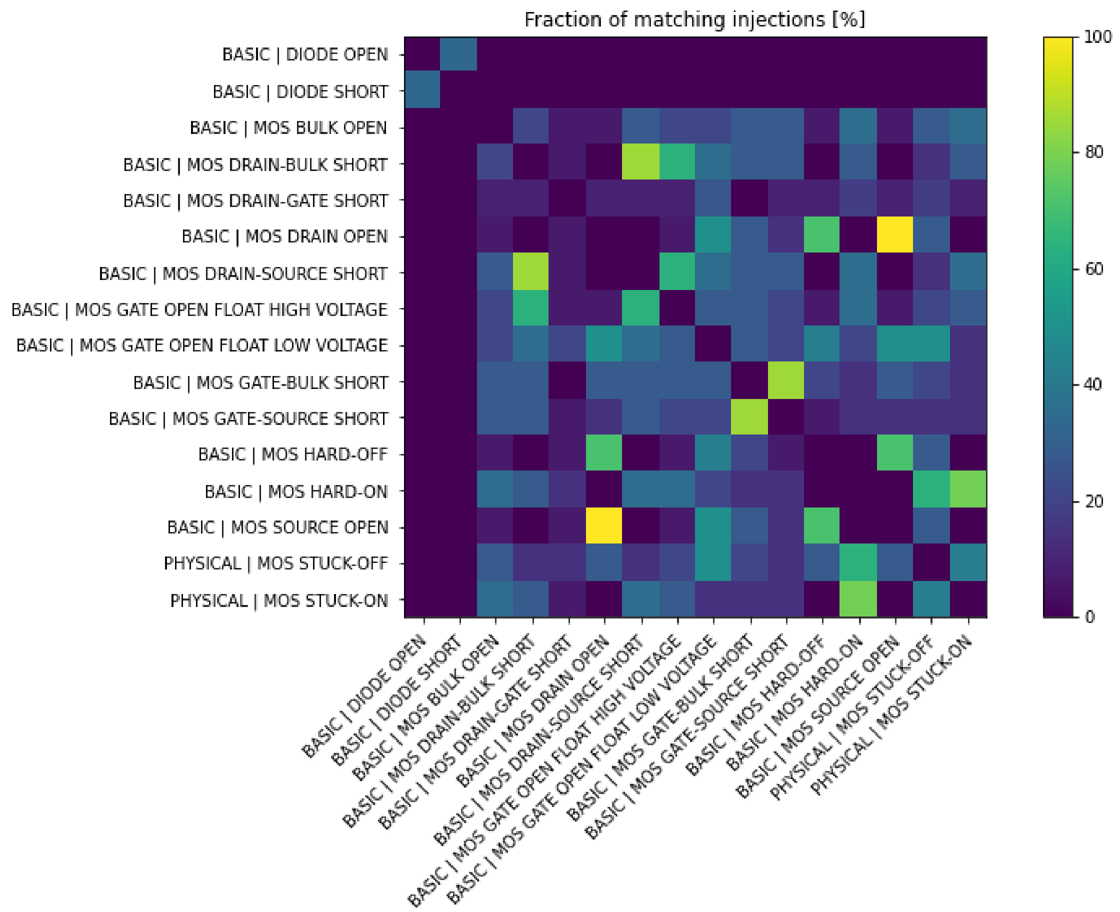


Fig. 4.7: Fault equivalence plot relative to the P2427 defects and the proposed stuck-on/off defect models for the OpAmp circuit.

particular, the defects considered are hard defect models as specified in IEEE P2427 draft standard, e.g., one open for a resistor (R), capacitor (C), and the inductor (L); two opens at the gate and drain for a MOS; a short between terminal pairs unless the nodes are already tied together in the design.

4.4.2 Transient analysis on the proposed stuck-on/off defects

The proposed stuck-on/off defects described in Section 4.2 are compared with the defects proposed in the IEEE P2427 draft standard. To compare the faults, a transient analysis is performed for each fault.

The results are plotted in Figure 4.8 for the CoAmp and Figure 4.7 for the OpAmp circuit. In the plots, matching close to 100% (yellow) means that the fault-type on the row was in the same group as the fault-type on the column nearly all the time, for all instances in which it was injected in the reference circuit. In Figure 4.8 it looks like for some faults there are more than one faults candidate that is equivalent respect to Figure 4.7. From these graphs, it is easy to see how standard defects (defined in the IEEE P2427 draft standard) cannot produce the same transient behaviors generated by the proposed stuck-on/off defects.

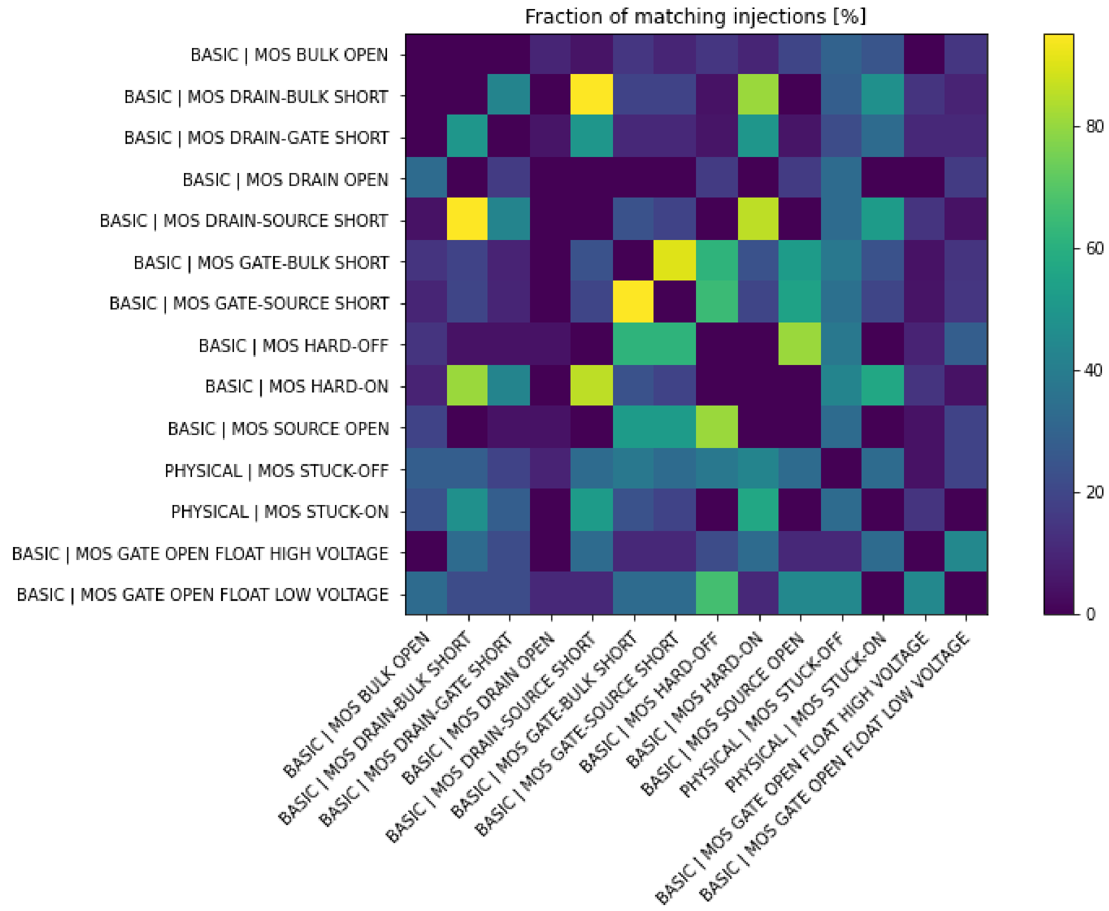


Fig. 4.8: Fault equivalence results between P2427 defects and the proposed stuck-on/off defect models for the CoAmp circuit.

4.4.3 Fault grouping based on the circle-fitting method

By performing the simulation of the circuit instrumented as described in Section 4.3.1 the admittance matrices are extracted. From the admittance matrices, it is possible to identify the circles that determine the distribution of the fault parameter value for each selected fault. Figure 4.9 shows the distribution of the fault parameter in the complex plane for four different faults, two different operating points (columns), and different frequency values, 10.0 MHz and 10.0 GHz (rows) for the OpAmp circuit. These four faults plotted are open and short faults injected in four MOS injected one fault at a time. Only three complex values are required to identify a circle for a single fault with a fixed frequency [54]. Starting from these complex values, the circle-fitting algorithm based on the Hyper Circle Algorithm allows finding the center of the circle and the radius. Figure 4.9 is generated through the data computed from the circle-fitting algorithm [55].

These results are observed for the S-parameter `S(3,1)` placed between the ports `out` and `inn`. After completing the circle fitting phase that extracts circles from the S-parameters, it is possible to group equivalents faults by analyzing the distance between the center of these circles. By grouping them, the overall number of faults to simulate is reduced.

In the end the summary of this work is that these faults are also need to be considered when dealing with analog circuits. These new defect models show complex behavior not covered from neither of the proposed

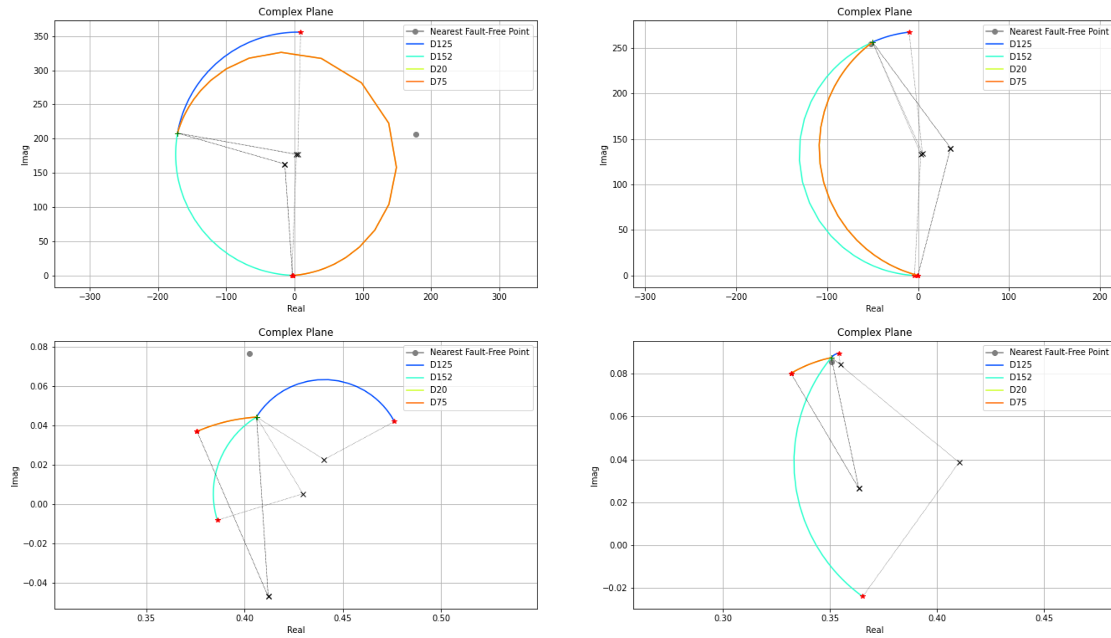


Fig. 4.9: Circles generated for the faults #D125, #D152, #D20 and #D75 relatives to the OpAmp circuit. These circles are generated with the circle-fitting algorithm for two operating point (columns) and two frequency point (rows).

IEEE P2427 *hard* defects for MOS transistors. This analysis also allows to group faults with equivalent behavior to reduce the entire set of faults to simulate by selecting only one representative fault for each group.

5

Predictive Fault Grouping based on Faulty AC Matrices

In a previous chapter we saw how to model analog defects. To analyse the effect of that fault fault grouping techniques is discussed in previous chapter. In this chapter that novel predictive fault grouping based on the collection of faulty AC matrices at fault-free operating points is explained in detail, as a means to approximate the final distribution of faults in equivalence classes using a minimal computational effort. The method is computationally cheap because it avoids performing DC or transient simulations with faults injected and limits itself only to AC simulations with faults activated. The technique provides an approximation, since it does not characterize faults at the corresponding faulty operating point but instead looks at how they would modify the fault-free operating point once injected. The approximate grouping achieves an excellent correlation to the final classification based on the comparison of faulty transient wave-forms. It is not meant as a substitute for the traditional fault injection simulations but as a support to decision making. It allows prioritizing faults to characterize the possible failure modes with a minimum number of fault injections, pushing out fault injections which are estimated to marginally increase the learning. Figure 5.2 depicts the methodological flow. First, we instrument the circuit with probing ports required to extract the AC matrices. Then, we perform a single transient simulation interleaved at specific Operating Points (OPs) by several AC simulations. We perform one AC simulation for the fault free behavior, and at most three for each fault. The number of AC simulation for each fault varies from one to three based on the applied method (i.e., AC- or circle-based grouping).

5.1 Background

This section explains background of concepts used in this chapter to better understand the methodology.

5.1.1 Types of analog failure modes

There are usually two types of faults in analog descriptions: *soft* and *hard* faults [56]. A subtle deviation of components parametric values to their nominal value, due to either systematic or random process variations, is called *soft* fault. A change of the circuit's topology due to the presence of *primarily* short or open circuits between nets is called *hard* fault.

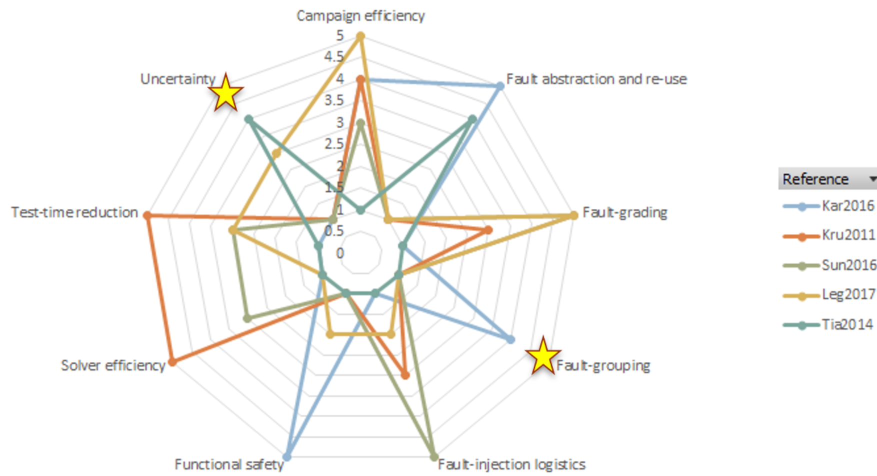


Fig. 5.1: Motivation behind fault grouping Work

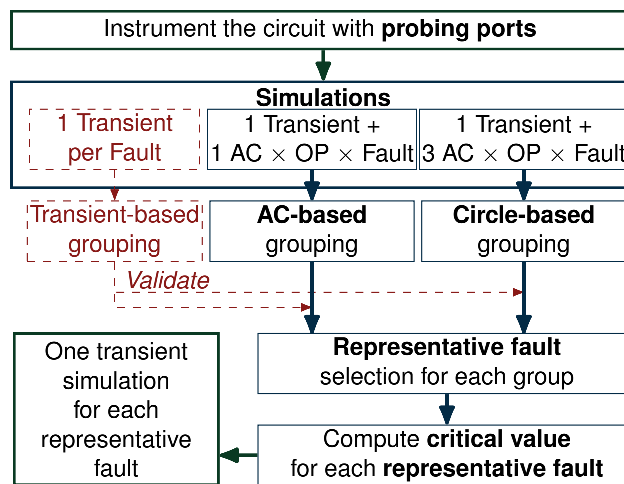


Fig. 5.2: Overview of the proposed methodology.

For this work different faults are considered, in details these faults are grouped for each electrical component following the IEEE P2427 draft standard [5]:

- Capacitor (C): open circuit, short circuit;
- Diode (D): open circuit, short circuit;
- Transistor (MOS):
  - open circuit: bulk, drain, gate, source;
  - short circuit: drain-bulk, drain-gate, drain-source, gate-bulk, gate-source.

5.1.2 AC matrices

The AC matrices are retrieved as the scattering parameters, also called S-parameter, during an AC simulation. These AC matrices describe the input-output relations between the ports in an electrical circuit. These relations are defined as a complex matrix (AC matrix) that shows the reflection/transmission characteristics (amplitude/phase) in the frequency domain [57]. Thus, it is possible to analyze the circuit's frequency behavior through the S-parameters [58]. The numbering convention for S-parameters contains two numbers

enclosed in round brackets and preceded by an `S`, e.g., `S(N1, N2)`. Here, `N1` identifies the port where the signal appears (the output), and `N2` identifies the port number where the signal is applied (the input). Starting from the AC matrices retrieved with the S-parameters it is possible to extract the Y-parameters, Z-parameters [59], H-parameters, T-parameters or ABCD-parameters [60].

5.1.3 Circuit instrumentation

A transient simulation is performed for each fault, and depending on the behavior of design, the circuit is linearized at one or more operating points. At each of these operating points, pause the simulation and perform an AC simulation. During the AC simulation, standard S-parameter probes are used to retrieve the circuit's behavior at different frequencies. The S-parameter probes are connected to our design through the 3-terminal circuit detailed in Figure 5.3. When the internal switch is in the *first* positioned between `cut` and `env` the S-parameter probe is not active, while when it is in the *second* position between `cut` and `gnd` the probe is active. When the *second* position is activated, the `cut` is isolated from the rest of the circuit and through the stimuli generated by the probe it is possible to extract the frequency behavior of the `cut`.

The 3-terminal circuit is useful for analyzing circuits behavior, however, main objective is to analyze their faulty behavior. For this purpose, the component shown in Figure 5.4 is developed, consisting of two switches connected through two pins inside the circuit. This component allows reproducing both faulty and fault-free behaviour, as well as probing S-parameters. Different configurations of the two switches are created, to perform the different simulations on the Circuit Under Test (CUT) required for the grouping phases. When the two switches are *both*: in the *upper* position, the circuit does not alter the CUT behavior during transient simulation; in the *middle*, it injects a parametrized faulty resistor in parallel to a S-parameter probe, which is required during the AC simulation for extracting the faulty AC matrices; in the *lower* position, it extracts the S-parameters with the probe, saving the AC matrices at the point where the fault is injected. The S-parameters are then used to calculate the critical value for a fault parameter.

5.2 State of the art

Several analog fault simulation and grouping techniques shown are presented and compared with proposed methodology in this section. During the analysis of the state of the art for analog fault simulation, it has been noticed that the most promising techniques that make fault simulation do not give due weight to the concept of the value assigned to the fault. Furthermore, few perform grouping to reduce the number of faults to be simulated, and generally use other techniques as shown in figure 5.1. The promising approaches from the state-of-the-art aim at reducing the number of faults through techniques like grouping faults into classes and studying the equivalence between faults as shown in Figure 5.5 . As opposed to an analog circuit, in a digital

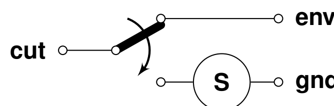


Fig. 5.3: A 3-terminal circuit for S-parameter extraction.

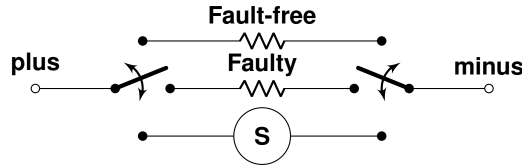


Fig. 5.4: Implementation of a S-parameter port connected only with two terminals inside an analog circuit.

circuit, checking the equivalence of faults is easier to perform due to the discrete nature of signals (i.e., they can be either 0 or 1). But with analog Integrated Circuits (ICs), which are modeled using continuous values, equivalence between faults is more complex. In [33] the authors discussed fault grouping based on hierarchical clustering on the faulty transient waveform. They showed that the grouping they realized allows predicting the final classification of faults in functional safety categories reliably. This result constitutes the *golden reference* of the present article, as our method strives to reproduce the same grouping as obtained from transient waveforms but by alternative much less computationally intensive means.

In [34], the authors show the limitations of the random sampling [35] for choosing the faults to be simulated in an analog circuit and highlight the necessity to consider the uncertainty of the fault parameter before simulating a fault. For that reason, in this work, we try to prioritize faults by applying an approximated grouping on the faults and taking the *central* component as representative fault for each group. With *central* component we mean the one that minimizes the distances to all the other group members. We also analyze the uncertainty of the fault parameter value that the representative faults assume by calculating a *critical value* for it. The *critical value* for a fault leads to an AC parameter that has the potential to deviate the most from its fault-free value.

In [36] authors presented a dynamic fault grouping algorithm for non-linear circuits. This methodology dynamically divides the list of faults into groups based on the transient behavior. Different time steps are used for different groups in simulation; however, the time step for faults within the same group stays the same. Another fault list compression technique is presented in [37], which uses stratified fault grouping for the identification of representative faults. Most of the techniques mentioned above are based on transient simulations, which is renowned for being time-consuming. Other techniques have explored different solutions that tend to reduce the number of transient simulations. Frequency-based analysis was presented in papers like [38] and [39]. In [19] authors presented a fault clustering technique combining faulty DC OP and frequency domain analysis. That technique requires N transient simulations (i.e., one for each fault), and then, at each OP they perform an AC analysis. In proposed methodology, faulty AC matrices are extracted at the fault-free OP, not at the faulty one. The difference is that for N faults, only one transient simulation is required to generate the OPs, then, at each OP AC analysis is performed with the faulty matrix.

5.3 Impact of a fault on the ac matrices

A fault in an analog circuit can lead the circuit to assume behaviors different from the nominal one, or behave like the circuit without the presence of the fault. To identify these abnormal behaviors of the CUT we strive to characterize faults in a generic fashion, independent of a given test method or performance

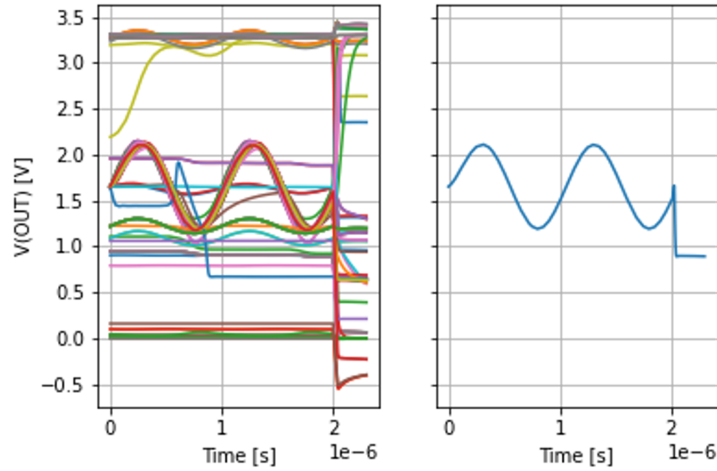
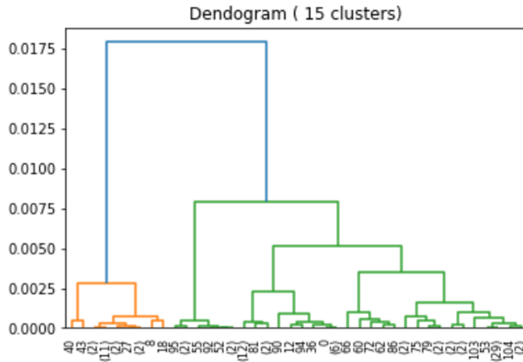


Fig. 5.5: Fault Equivalence

Dendogram of the **waveform-based** grouping



Dendogram of the **AC-based** grouping

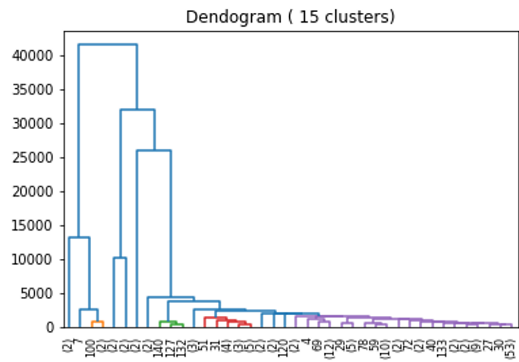


Fig. 5.6: Clustering trees

metric. We want to characterize *raw behavior* such that the grouping result would remain valid whatever the situation the circuit is put in later. It is known that collection of AC matrices at various operating points are sufficient to characterize the non-linear behavior of an analogue circuit (indeed, this is the fundamental principle of SPICE solvers). Like in [19], we use AC matrices at several operating points to characterize the circuit behavior in a generic fashion.

Let us take a circuit with N ports represented by its scattering matrix: $b_N = S_N \cdot a_N$, where a_N and b_N are vectors of incoming and out-going (reflected) traveling waveform response and S_N the scattering matrix. We can formulate a scattering matrix S_{N+1} for the circuit with an additional port such that the fault is externalized as detailed in the Equation 5.1:

$$\begin{bmatrix} b_1 \\ \vdots \\ b_N \\ b_X \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} S_{PP} \\ \\ \end{bmatrix} & \begin{bmatrix} S_{PF} \\ \\ \end{bmatrix} \\ \begin{bmatrix} S_{FP} \\ \\ \end{bmatrix} & \begin{bmatrix} S_{FF} \\ \\ \end{bmatrix} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_N \\ a_X \end{bmatrix} \tag{5.1}$$

where a_X is defined as $a_X = \mathit{\Gamma} \cdot b_X$. The fault model is represented by the Equation 5.2:

$$\mathit{\Gamma}_f = \frac{1 - g_f \cdot Z_c}{1 + g_f \cdot Z_c} \tag{5.2}$$

where g_f is the fault parameter, and Z_C is the reference impedance. By substituting a_X and solving for b_X we obtain an equation for the external S-parameters of the circuit in function of the fault parameter $\mathit{\Gamma}_f \in]-1, +1[$ as described in the Equation 5.3:

$$S_N(\mathit{\Gamma}_f) = S_{PP} + S_{PF} \cdot (1 - \mathit{\Gamma}_f \cdot S_{FF})^{-1} \cdot \mathit{\Gamma}_f \cdot S_{FP} \tag{5.3}$$

The Equation 5.3 is also valid in-case the fault network is a multi-port. Focusing on a single element of the matrix S_N we obtain the Equation 5.4:

$$S_{i,j}(\mathit{\Gamma}_f) = S_{PPi,j} + \frac{\mathit{\Gamma}_f \cdot S_{Xi,j}}{1 - \mathit{\Gamma}_f \cdot S_{FF}} \tag{5.4}$$

where $S_{PPi,j}$, $S_{Xi,j}$ and S_{FF} are complex constants, independent of the value of $\mathit{\Gamma}_f$. For physical realizations, $\mathit{\Gamma}_f$ never reaches $+1$ or -1, which correspond to the ideal open or the ideal short fault. The Equation 5.4 shows that some values of $\mathit{\Gamma}_f$ will result in a maximal deviation of the $S_{i,j}$, representing the so-called *critical value* that will be thoroughly discussed in Section 5.5.5.

5.4 Predictive fault grouping

In this Section, the fundamental steps of the proposed methodology are explained. It starts from the AC-based grouping based on the S-parameter and moves to the Circle-based-grouping based on the circle-fitting method applied on the AC matrices. The above-mentioned grouping techniques are compared with the transient-based grouping to experimentally validate the methodology.

5.4.1 AC-based grouping

The paper of [19] introduced the idea of fault grouping based on the AC matrices response. It combines the analysis of faulty OP in DC simulations and faulty AC simulations to group the faults and compare it with the transient results. In this approach, to perform the clustering `N` transient simulations (each time the fault is injected) are required and then at each OP an AC simulation is performed. Conversely, proposed approach

is based on the extraction of the AC matrices at fault-free OP. The hierarchical clustering is used because it reveals the distance structure between the faults. Let us consider `N` faults, we need to do one transient simulation to generate the OPs, then at each OP do the AC simulation with the fault and the probe injected. The AC-based grouping relies on feature vectors constructed from the elements of the complex-valued scattering matrices, extracted at various frequencies and time-points.

After collecting all the S-parameter the data are processed using a clustering technique. The metric used to subdivide the cluster is depicted in the Equation 5.5 and represents the Euclidean distance between two S-parameter matrices:

$$d(\mathbf{S_A}, \mathbf{S_B}) = \sqrt{\sum_{i,j,k,l} \left(S_{A(i,j)}(t_k, f_l) - S_{B(i,j)}(t_k, f_l)\right)^2} \tag{5.5}$$

where S_A and S_B are two different S-parameter matrices relative to two distinct fault. Yes the clustering algorithm is based on a measure of distance. In fact, I have produced some dendrography which are a visual display of the number of groups you get based on how you set the threshold for group separation. In the case of groupings I have done, the distance was not computed on the wave forms but on collections of small-signal matrix elements over frequency and bias (operating points).

5.4.2 Circle-based grouping

In the world of Integrated Circuit (IC) testing, it is known that the AC matrix elements at a given frequency and operating point describe circular trajectories in the complex plane when the fault parameter is varied over a wide range of values [54]. These circles represent the locus of AC matrix element values for a distribution of fault parameter values and can be used to uniquely represent a fault characterized by an uncertain fault parameter. Hence the idea to use the distance between fault circles to compare uncertain faults.

The circle equation identified from the AC matrices can model any continuous parameter shifting (*soft*) or topological change (*hard*) fault to linear and piece-wise linear circuits. Circle-fitting is commonly used with S-parameters because any resonance causes the S-parameter to describe an arc in the *Cx* plane. S-parameters are a bi-linear transformation of admittance parameters, and these transformations are known to create circles or lines.

The proposed circle-fitting method allows proving that two faults are equivalent and enables fault clustering by looking at the feature vectors constructed from the centers of the fault circles. Implementation of circle-fitting method is based on the Hyper Circle algorithm [55]. At least two faulty AC simulations must be performed with different fault parameter values plus the AC fault-free simulation, to retrieve the necessary data to apply the Circle-based grouping. These three AC simulations allow to identify at least three points in the *Cx* plane on which the circle-fitting algorithm is applied. After identifying all the circles in the AC faulty matrices, the clustering is performed based on the metric of Equation 5.6:

$$d(\mathbf{C_{SA}}, \mathbf{C_{SB}}) = \sum_{i,j,k,l} \Big| cc(S_{A(i,j)}(t_k, f_l, \mathit{\Gamma}_A)) - cc(S_{B(i,j)}(t_k, f_l, \mathit{\Gamma}_B)) \Big| \quad (5.6)$$

This equation computes the distance between the center of two circles in the *Cx* plane. The *cc* function computes the circle center with respect to a given S-parameter. The distance measure is computed for all time points and all frequencies.

5.4.3 Transient-based grouping

This section shows how to perform a clustering of waveforms (`VOUT` waveform) based on their trajectory. It is well known that it is possible to group faults by running transient fault simulations and then group the waveforms [33]. However, this technique is time-consuming and requires a sensible amount of computational resources, so it is presented in this work for the sole purpose of validating the proposed methodology (see Figure 4.1). Basically, waveforms are compared for various faults using a *modified Euclidean distance* measure over the space of functions applied to the simulation time interval. This measure is used as matrix to perform the clustering.

Computing the distance measure between two waveforms W_A and W_B is performed with the Equation 5.7:

$$d(W_A, W_B) = \frac{t_{e,ff} - t_0}{t_{e,fy} - t_0} \cdot \int_{t_0}^{t_{e,fy}} (W_A(t) - W_B(t))^2 \cdot dt \quad (5.7)$$

with t_0 the start-time of the simulations, $t_{e,ff}$ and $t_{e,fy}$ the end times of the faulty and fault-free simulations with the same stimuli (sometimes the simulator aborts the faulty simulation prematurely – this is compensated by the fraction in front of the integral). The innovation proposed with Equation 5.7 is the calculus of the distance between two waveform, by considering a *modified Euclidean distance* that is integrated over a weighted time correction factor. This correction factor used in the formula allows to penalize simulations which did not run till the end.

5.5 Methodology Validation

In this section, the proposed methodology flow is applied to the Operational Amplifier (OpAmp) circuit shown in Figure 6.4, retrieved from the analog-benchmark circuits collection [1]. The different grouping techniques are compared to show the potentialities of the AC-based and the Circle-based ones.

5.5.1 Circle-fitting method

The Circle-based grouping is relying on the circle-fitting applied to the faulty AC matrices. In Figure 5.7 some circles are plotted for the fault `D20`, that is, a MOS drain-source short on the transistor `MNM12` located in the OpAmp circuit, in the *Cx* plane. The circles are plotted for three different frequency values: $1Hz$, $10kHz$ and $10MHz$. In these plots, blue lines are the fault trajectories in function of the fault parameter value. One blue curve is related to one time point.

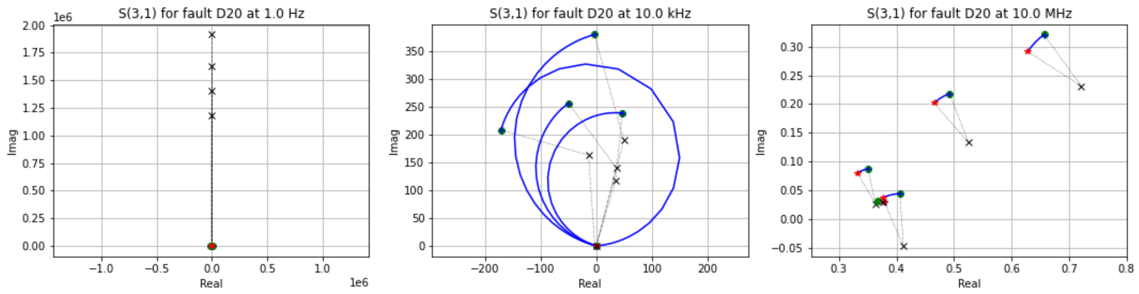


Fig. 5.7: AC trajectories of the fault D20 (MOS drain-source short on the transistor `MNM12`) at three frequencies and four operating points.

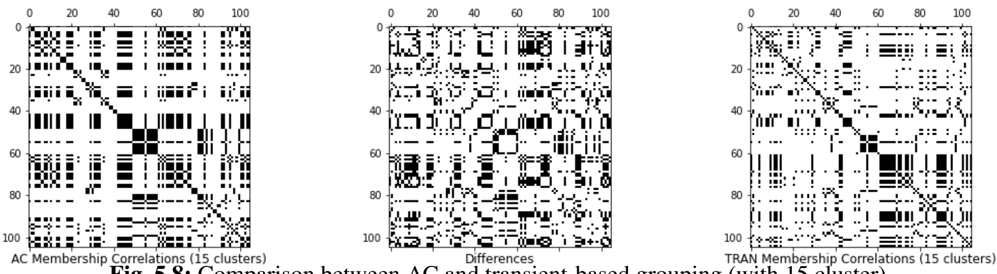


Fig. 5.8: Comparison between AC and transient-based grouping (with 15 cluster).

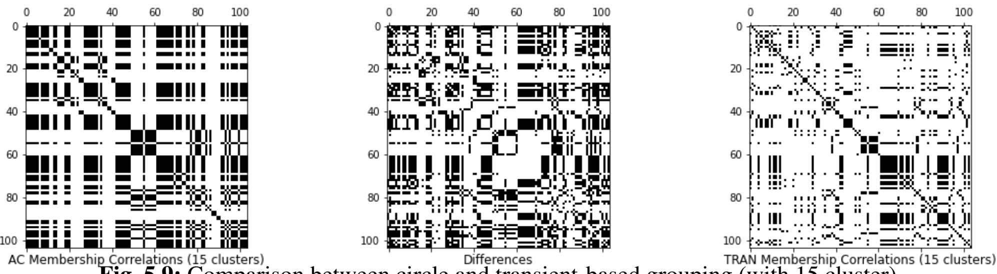


Fig. 5.9: Comparison between circle and transient-based grouping (with 15 cluster).

The red star identifies the faulty case. The green ring is the fault-free case retrieved from the fault-free simulation with no fault probe injected. From these plots is simple to identify well defined circles in the middle plot. This experimental validation allows to prove that the S-parameter circle equation depicted in Section 5.3 is correct.

5.5.2 Comparison between the different grouping techniques

Figure 5.8 and Figure 5.9 present the group-membership correlation matrices of the faults. An element at position `(i,j)` is set to 1 if the faults corresponding to indices `i` and `j` belong to the same group and to 0 otherwise. We generated middle plots of both Figure 5.8 and Figure 5.9 called “Differences”, by an element-wise application of the xor logic operator to the two outer matrices. The xor operation will produce a 1 (non-zero value) if only if the two input values are different. As the differences increase the middle plot will have more black dots, while if the differences are fewer the plot will have fewer black dots.

5.5.3 AC-based vs transient-based grouping

In Figure 5.8 on the left is plotted the AC membership correlation based on the S-parameters (with 15 clusters), while, on the right the transient membership correlation is plotted (15 clusters). As shown by the plot

in the middle, the error in the group membership correlations are around the 23%, meaning that the overall group structure has many similarities. This correlation error allows us to state that the proposed technique is robust, and efficient, as it allows us to perform only AC simulations, saving time and computational resources.

5.5.4 Circle-based vs transient-based grouping

In Figure 5.9 on the left is plotted the AC membership correlation based on the Circle-fitting method (with 15 clusters), while, on the right the transient membership correlation is plotted (15 clusters). As shown by the plot in the middle, the error in the group membership correlations are around the 34%, meaning that the overall group structure has less similarities respect to the grouping based on the S-parameters. This higher error, is mostly due to numerical errors that arise in the computation of circles, especially for low frequencies, because the imaginary part of the AC matrices tends to zero.

5.5.5 Critical parameter value of a fault

Table 5.1: Critical values at different frequency points for the most significative OPAMP circuit faults, and their group.

Fault Number	Component	Fault Type	Fault Group	Nominal Value (Ω)	Cvalue_10kHz (Ω)	Cvalue_10MHz (Ω)	Cvalue_10GHz (Ω)
D12 (net13, vssa)	mnm12	short	3	10e9	426749.5336	506.9381274	233.697556
D15 (net27, net21)	mnb02	open	2	0.1	838344.1240	188682.4351	930.80153
D17 (net21, vssa)	mnpd1	open	2	0.1	838121.6656	187899.1321	811.7642162
D22 (net13, vssa)	mnpd2	short	3	10e9	426749.5336	506.9381274	233.697556
D25 (net27, net158)	mpb02	open	2	0.1	5484.617763	5022.308952	73.51478842
D29 (net31, vdda)	mps11	open	3	0.1	762.2196089	875.5493889	552.1862102
D31 (net56, net31)	mpd11	open	1	0.1	3628749.934	-347289.8597	-163026.9545
D32 (net56, net31)	mpd11	short	3	10e9	1583.158658	1581.238934	1384.352676
D36 (net13, net31)	mpd12	short	1	10e9	221250.6446	574.4758309	457.4399353

After clustering on the AC matrices (faulty and non-faulty) and selecting a representative fault for each cluster, chosen in the middle of the cluster it is necessary to simulate these faults in order to understand the behavior of design in the presence of faults. These clusters are shown in 5.6. To simulate these faults and produce alterations that deviate significantly from the fault-free behavior it is necessary to inject each fault with an appropriate value. This value is the resistance value that is associated with the resistance used to model the fault. The critical values and the fault grouping are reported in Table 5.1 for a subset of the OpAmp' faults. The table reports: the fault group, the nominal value that allows the fault to behave as fault-free, and the critical values calculated for three different frequencies: $10kHz$, $10MHz$ and $10GHz$. These critical values are computed by studying the admittance at the fault location with the fault disable (switch placed in the third position, see Figure 5.4). This critical value means that the sensitivity of the AC parameter to the fault parameter is maximal. This means that for any other values, the sensitivity will be lower. It is crucial to identify this critical value to avoid injecting an undetectable fault that behaves like a faulty-free. The critical value for a given frequency is extracted from the admittance matrix, by looking for those values that have the modulus of the imaginary part closest to zero. The critical value (i.e., g_f) will be

the real part of those values. If the fault-free value of a fault parameter is far from the critical value, then the circuit has a high tolerance against faulty values. Instead, if the faulty-free value is close to the critical value, then the circuit is susceptible to any fault.

6

A Unified Manipulation Framework for Transistor-Level Languages

This chapter explain a unified manipulation tool for the transistor level languages. There are several ways to build complex transistor-level netlists and testbenches; this is closely related to the evolution of the languages with which these models are described. Initially, only one language allowed to model transistor-level netlist: the Simulation Program with Integrated Circuit Emphasis (SPICE) born in the 1973. Many other "dialects" were born starting from the original SPICE language and have added many analysis and modeling capabilities over the decades. For most of them, the semantic match those of SPICE, and only the syntax is changed. Many models with complex behaviors have been defined within the new SPICE-based languages and always reflect the original semantics defined by the SPICE language. Instead, there are differences when more advanced circuit analysis methods have been defined, *i.e.*, Radio-Frequency (RF) analysis on circuits. Consequently, a commercial tool is usually required for simulating, analyzing, and especially manipulating these languages. This work proposes a novel open-source *framework* that relies on the shared semantic for reading, writing, or manipulating transistor-level designs. The entire framework is open-source and available on GitHub: *https://github.com/sydelity-net/EDACurry*.

6.1 Background

This section describe some background and state of the art related to proposed methodology to understand the work.

6.1.1 Transistor-level Languages

Electronic circuit simulation is necessary for designers and manufacturers. If we look back into history, many tools were developed to simulate these circuits. Developing these tools was necessary to validate the models before the fabrication due to the increase in the complexity of the circuits. Simulation Program with Integrated Circuit Emphasis (SPICE), Eldo, and Spectre are the simulators used to simulate transistor-level descriptions of analog circuits. From the original SPICE language developed by the UC Berkeley University, many dialects were born: Spice2, Spice3, PSpice, and HSpice. UC Berkeley invented SPICE and all the dialects from 1975 to 1993. At the same time, Cadence designed PSpice in 2001. Avati/Meta Software manufactured HSpice in 2001.

Another tool with an associated language was Spectre [21], created by Cadence. This tool is a SPICE-class circuit simulator. It supports the fundamental SPICE analyses: DC, AC, TRAN, and HB. Moreover, it supports the Verilog-A and Verilog-AMS languages. An enhanced version of Spectre provides support of RF and mixed-signal simulation. Currently, Spectre is one of the leading circuit simulators and competing with HSPICE and many others. At the same time, Eldo supports the simulation of extensive analog circuits in both transient and frequency domains. Furthermore, it provides a simulation speed greater than SPICE-based simulators available for commercial use and maintains the same accuracy.

6.1.2 Code manipulation strategies

Parsers are programs that allow analyzing strings of symbols. Once the input string is parsed and mapped to a data structure, it can also be manipulated. The proposed framework is built with the support of the ANother Tool for Language Recognition (ANTLR) tool. In particular, starting from our grammar, the ANTLR is used to generate the skeleton of the internal parsers. ANTLR is a parser and translator generator tool that allows defining grammars in both ANTLR syntax (similar to EBNF and YACC) and in a special abstract syntax for Abstract Syntax Tree (AST). ANTLR can create parsers (in Java or C++) and ASTs. To generate the parser skeleton through the ANTLR library, it is necessary to define for each language a *lexer* and a *parser*. The task of the lexer is to collect the stream of characters that the syntactic analyzer reads as input into groups of characters. These groups of characters, processed by the parser, acquire meaning. Each character or group of characters collected through the lexer is called a token. Tokens are components of the programming language read in input, such as keywords, identifiers, and symbols.

The tokens recognize through the lexer acquire an individual semantic value. It does not consider their semantic value in the context of the whole program because this is the duty of the parser. The parser organizes the tokens into allowed sequences of the language's grammar. If the language is used as defined in the grammar, the parser will recognize patterns that constitute specific structures and group them appropriately. If the parser encounters a sequence of tokens that matches none of the allowed sequences, it will raise an error and perhaps try to recover it by making some assumptions about the nature of that error.

The parser checks that the token conforms to the syntax of the language defined by the grammar. Usually, the parser converts the sequences of tokens for which it was built by matching them to another form, such as an AST. An AST is easier to translate into an objective language because it implicitly contains additional information due to its structure and is the essential part of the language-translation process. When for a language the lexer and parser are defined, ANTLR allows the definition of rules that the lexer should follow to tokenize a stream of characters and rules that the parser should use to interpret the stream of tokens. ANTLR can generate a lexer and a parser that can be used to interpret programs written in one language and translate them into other languages and ASTs. The ANTLR project allows many extensions and has many applications.

6.2 State of the art on transistor-level manipulation tools

There are many tools used to simulate the transistor level circuits [21, 30, 7]. Every tool supports there own syntax/format of language for the simulation. Usually, these languages have standard semantics but different syntax. Researchers in industry and researchers are working on manipulating and converting these languages from one to another. After the definition of many transistor-level languages created from the original SPICE language, few tools were developed that allow the manipulation and conversion of these languages from one to the others. These tools can convert libraries and circuits between these transistor-level languages, e.g., Spectre to Eldo converter(spect2el), Netlist Translator for SPICE, and Spectre [31]. Spectre to Eldo converter is available in the Eldo simulator that allows converting a Spectre circuit to the equivalent circuit described in Eldo. However, there are some limitations of using spect2el. Firstly, it does not support all the statements of the Spectre language, e.g.; if some sections are written in spice spect2el, the tool can not correctly translate the circuit. Secondly, conversion is uni-directional, only from Spectre to Eldo [30], and the other problem is that some features require manual effort for the modifications. Both Eldo and Spectre languages have some similarities but differ in many aspects. Currently, there is no available framework that provides unified semantics. Above mentioned Netlist translator [31] was developed to simulate the Spectre and SPICE (Spice2, Spice3, PSpice, and HSpice) in Advance Design System (ADS). In [32] a python interface for SPICE-based simulations was presented. The purpose of that interface is to help the designer in the industry solve circuit sizing problems of integrated analog CMOS circuits for SPICE-based simulations.

6.3 Language independent framework

This section presents the manipulation framework that allows manipulating transistor-level descriptions in a structured way. The manipulation framework structure is presented in Figure 6.1. This common framework is based on the fact that all the transistor-level languages that exist on the market have a different *syntax*, but usually a common *semantics*. The entire framework is publicly available on GitHub [61]. Several front-ends bring heterogeneous descriptions into an in-memory description based on the AST generated by the parsing phase within the framework. These language descriptions that the framework reads as input can be written in Eldo, Spectre, or XML languages as output. After the parsing phase, the structure of the considered AST will be fully represented within the internal structure of the framework. This means that no information will be lost during this transcription from the input description to the intermediate description within the framework. If required, it is possible to specify through the framework whether to skip the parsing of the comments or not. In the following sections, the entire internal structure of the framework will be described in detail. The tool allows performing several manipulation operations on the internal structure, for example, changing the master of a component or the pins of a subcircuit. Finally, several back-ends allow generating the manipulated design as an output file written in Eldo, Spectre, or XML descriptions. These back-ends allow simulating these new descriptions produced through the framework using commercial and

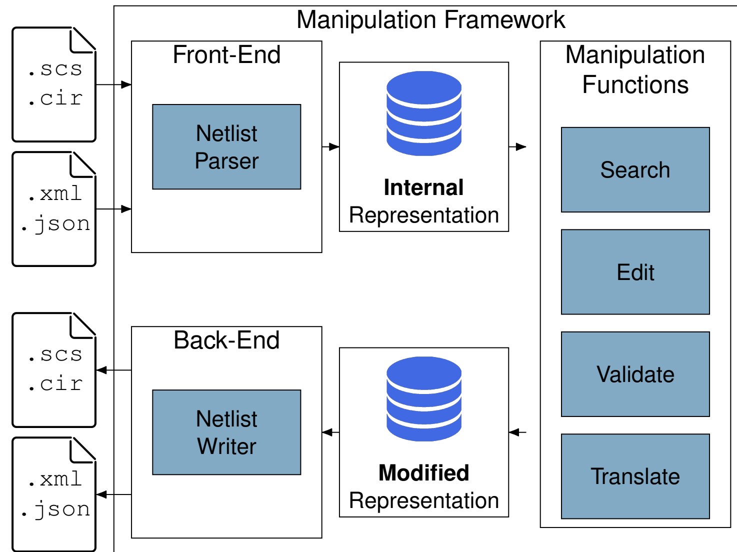


Fig. 6.1: Overview of the proposed manipulation framework.

```
parser grammar ELDOParser;
options { tokenVocab = ELDOLexer; }
netlist
    : netlist_title? (NL+|EOF) netlist_entity* (NL+|EOF);
netlist_title
    : ID+;
netlist_entity
    : include
    | library
    | subckt
    | analysis
    | global
    | model
    | global_declarations
    | control
    | component
    | end;
```

Listing 6.1: Sketch of the Eldo parser written in Antlr4.

non-commercial analog circuit simulators. Furthermore, this framework can convert an Eldo to Spectre and vice-versa. This technology is based on the equivalence of the semantics of these languages that are used to describe transistor-level components. Therefore, the development of the Eldo and Spectre grammars is the core of this framework.

The framework's support to read and write XML descriptions allows the designer to build pipelines of tools built upon the proposed framework's APIs. These intermediate tools could be shared among designers and enable the re-utilization.

6.3.1 Definition of a specific parser for each language

The input netlist should be read in input from the proposed manipulation framework to store all the information inside the internal structure. To read a netlist in input, it is necessary to have a unique front-end for each language that will parse the information read in input and map it to the internal structure of the framework. For this reason, grammars are developed by following Antlr4 syntax to read input netlists. In particular, these grammars are for the Eldo language and one for Spectre. The Eldo grammar can also completely

```
parser grammar SPECTREParser;
options { tokenVocab = SPECTRELexer; }
netlist
    : netlist_title? (NL+|EOF) netlist_entity+ (NL+|EOF);
netlist_title
    : ID+;
netlist_entity
    : include
    | library
    | subckt
    | analysis
    | global
    | model
    | global_declarations
    | control
    | component
    | lang
    | section
    | analogmodel
    | statistics;
```

Listing 6.2: Sketch of the Spectre parser written in Antlr4.

cover the original SPICE language from which it derives. To define these two grammars, the commonalities relatives to the semantic of these languages are exploited. For Eldo and Spectre, a lexer and a parser are defined. Into the lexer all the possible string tokens are defined. While, into the parser it is defined the significance of series of contiguous tokens. Then, through the Antlr library, the empty skeleton of the C++ parsers is generated. The skeleton of the parser generated through Antlr can be a Visitor or a Listener, both top-down recursive-descent. The *Visitor* and *Listener* pattern allows to visit the Abstract Syntax Tree (AST) generated from the grammar, but there are important differences between them. A *Visitor* pattern allows to visit each element of an AST with specific visit functions. While, the *Listener* pattern allows to visit an AST whit an enter and exit function for each element of the tree. In the proposed framework the *Visitor* pattern is used to create the front-ends.

Let us focus on the two languages, Eldo and Spectre. The former has a well-defined syntax without ambiguity, while the latter has a syntax that is sometimes ambiguous [30, 21]; for that reason, the complexity of the two proposed grammars is different. Nevertheless, both languages are equivalently used to represent transistor-level netlists. Firstly the commonalities between the Eldo and Spectre grammar are studied and then defined their parsers accordingly. Consequently, it makes it easier to map the parsed descriptions (*e.g.*, Eldo, Spectre, SPICE) to the internal representation of the framework. In Listing 6.1 a fragment of the Eldo grammar we defined is proposed. While in Listing 6.2 a fragment of the Spectre grammar is proposed. Analyzing these two grammar fragments makes it easy to identify similar categories of elements in a netlist written in Eldo and in Spectre. For example, in common, it is present: subcircuits, control commands, analysis statements, and components in both languages.

6.3.2 Similarities between the Eldo and Spectre languages

Eldo and Spectre syntactically are very different from each other, but the points of contact of semantics are many, as evidenced in the structure of the two grammars shown in Listing 6.1 and Listing 6.2. For example,

a subcircuit in both languages may contain definitions and instantiations of components. The parts that can be described equivalently in either language are:

- Analyses statements: AC, DC, Transient, *etc*;
- Control statements: Option, Save, Alter, *etc*;
- Sources: Isource, Vsource, *etc*;
- Components: Current probe, *etc*.

Now let us consider the component that is used in both languages to instantiate subcircuits. The Listing 6.3 shows how you can instantiate a subcircuit with the Eldo language; this is possible with the `X` component. This `X` component is only used to instantiate custom models and not standard models of the language. A standard component in Eldo is defined through components that start with alphabetic letters.

```
// Subcircuit Instance
subckt_instance : SUBCKT_INSTANCE node_list ID
                | SUBCKT_INSTANCE (MODEL COLON)? ID;
```

Listing 6.3: Sketch of the Eldo grammar for the component X.

Let us consider the same syntax used for instantiating subcircuits in the Spectre language. The Listing 6.4 shows the generic definition of a component for the SPECTRE language. Different from what happens in the Eldo language, where we have a specific component `X` to instantiate a generic subcircuit, in Spectre, we do not have a specific component. A component, either library or custom (thus, defined via a new subcircuit) in Spectre, is defined in the same way: the component identifier, the list of nodes, the master, and its attributes. Therefore, the master can be a generic subcircuit or a standard library component.

```
// Generic Component for Subcircuit Instantiation
component
    : component_id node_list? component_master
      component_attribute* (NL* | EOF);
component_id
  : ID;
component_master
    : ID | component_type;
component_attribute
    : component_value
    | component_value_list
    | component_analysis
    | parameter_assign;
```

Listing 6.4: Sketch of the Spectre grammar for the instantiation of a component.

6.3.3 Framework internal structure

The internal classes representation of our C++ framework is depicted in Figure 6.2 and 6.3 as Unified Modeling Language (UML) diagram. Each class into the UML diagram represents a fundamental block of a transistor-level netlist. The grammar written for Eldo and Spectre is structured to reflect the internal

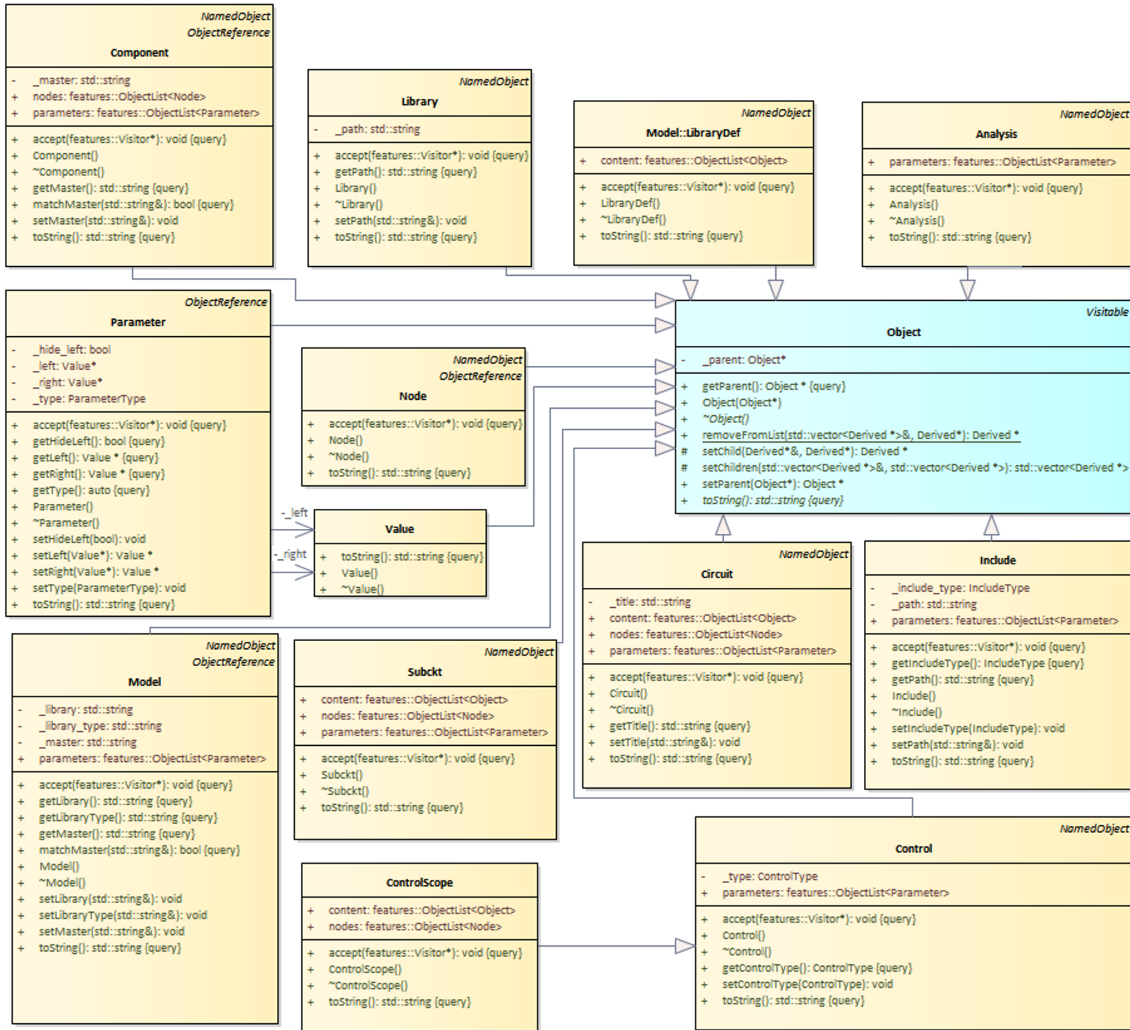


Fig. 6.2: UML Class diagram of the internal structure of the framework - part 1.

class structure of proposed framework. With structured grammar, it is easier to build front-ends to populate the internal representation of the framework. The framework is developed with a unique internal structure that can collect all the netlist information. The concept behind the idea of a unique internal structure for manipulating languages is the basis of the HIFSuite [7] framework. HIFSuite allows manipulating another class of languages: Hardware Description Languages (HDL), *e.g.*, VHDL, Verilog, Verilog-AMS. Usually a transistor-level netlist contains a circuit (`Circuit` class) that could contain many subcircuits (`Subckt` class) and simulation control commands (`Command` class). Then, inside a subcircuit are stored the components (`Component` class), and for each component, many parameters (`Parameter` class) and assignments to a parameter (`ParameterAssign` class). Moreover, a subcircuit had defined its pins (`Node` class) to interconnect it with other designs. In details, the UML diagram in Figure 6.2 presents the set of classes necessary to save all the netlist elements, *e.g.*, `Circuit`, `Subckt`, `Model`, `Control`, `Include`, `Node`, `Parameter`, `Component`, `Library`, `Analysis`.

All these classes that represent netlist elements implement the functions of the abstract class `Object` (non-dashed arrow with a full point towards the extended class). It is easy to build an internal tree-structured

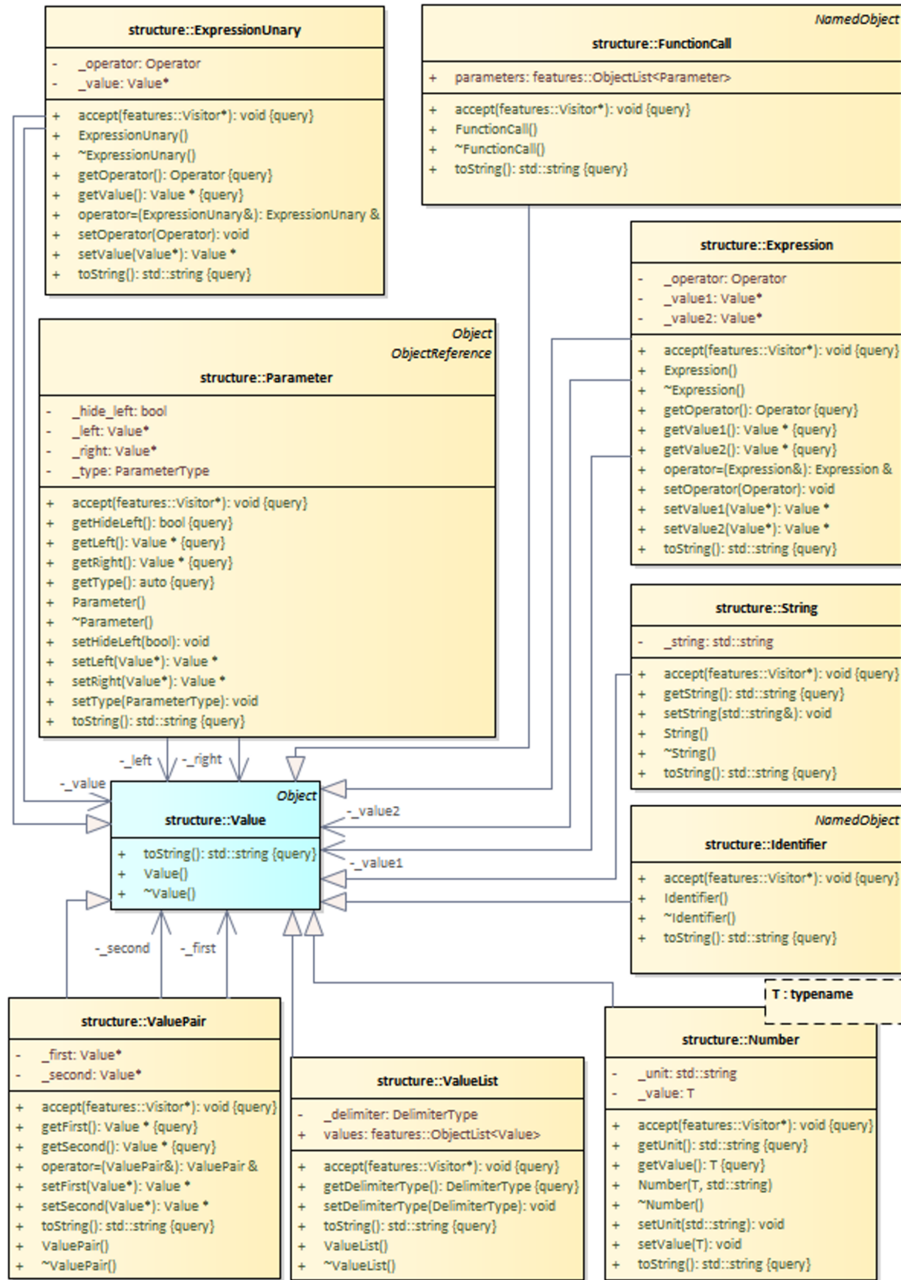


Fig. 6.3: UML Class diagram of the internal structure of the framework - part 2.

representation of all the netlist information with this class structure. This proposed structure of the framework implements the *Composite* design pattern, necessary when it is required to compose objects into tree structures. This is because the class `Circuit` for example has a set of ports (represented as a list of `Node`), a list of parameters (represented as a list of `Parameter`) and a content (represented by a list of `Object`). In this way, a `Circuit` class can structurally contain other elements of a netlist that implement the `Object` class. In addition, the *Visitor* behavioral design pattern is implemented by adding the `accept()` function to each class. In this way, it is simple to visit each element of the internal structure through a visitor or listener class to create a dedicated back-end to print in output the structure information.

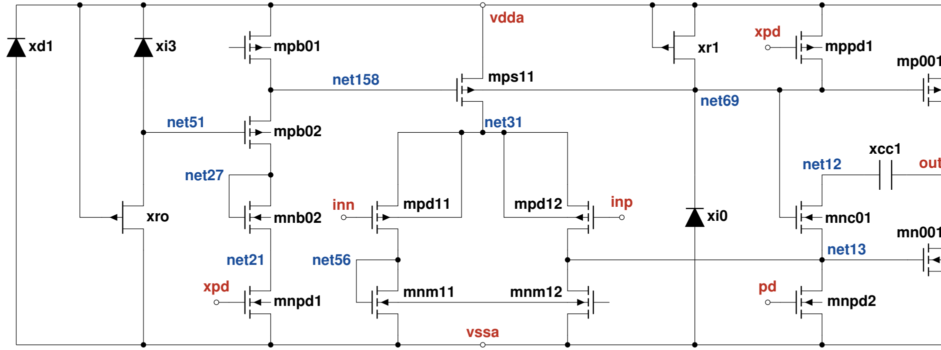


Fig. 6.4: Transistor-level description of the OPAMP model extracted from IEEE analog benchmarks suite [1].

The second UML diagram in Figure 6.3 presents the set of classes necessary to save the netlist values. A value (Value class) could be a number (template's class Number), a string (String class), an expression (Expression class), *etc*. The Value class is defined as abstract. Consequently, the derived classes should implement their interfaces. Moreover, all these classes that represent netlist values have an accept() function necessary to visit it with a visitor. Other design patterns besides the *Visitor* design pattern are used within the proposed framework, for example, the *factory method* and *composite* design patterns. Through the design pattern *factory method*, part of the creational patterns, it is easy to instantiate objects that are part of our internal structure. Similarly, the front-end created from a specific grammar will use the functions provided by the Factory class to instantiate objects by avoiding instantiation issues. Another design pattern used for the debug print is named *Template method*, and it is a behavioral design pattern. Following this design pattern, the skeleton of an algorithm is defined into the abstract (or superclass), and into the subclasses the same method is overidde by adding additional steps to the algorithm without changing its structure.

All the classes of the framework are wrapped with the Pybind11 header library [62]. Pybind11 is a header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code libraries. This wrapping allows to call all the functions, classes, and class elements defined in the C++ code from Python.

6.4 Framework validation

The framework presented in Section 6.3 has been validated with different designs with open access. In particular, to show how it is possible to define the same design with different languages by keeping the same semantics,it is considered the model of an Operational Amplifier (OPAMP), presented in Figure 6.4. This model is retrieved from the analog benchmarks suite available online [1]. The Listing 6.5 shows the subcircuit for the model of the OPAMP described using the Eldo language. This electrical network is composed of several components, especially MOSFET, which are interconnected to define the behavior of the OPAMP. The OPAMP model is an integrated circuit that can amplify weak electric signals. The component parameters are exemplified with the token <parameters>. The Listing 6.6 shows the subcircuit for the model of

```
1 .subckt OPAMP1 inn inp out pd xpd vdda vssa
2     xd1 vssa vdda prim_nwd AREA=6.13907e-9 PJ=476.6e-6 M=1
3     xi0 vssa net69 prim_nd AREA=1e-12 PJ=4e-6 M=1
4     xi3 net51 vdda prim_pd AREA=790e-15 PJ=3.3e-6 M=1
5     xr0 vssa vdda net51 prim_rdiffp L=2.1e-6 W=700e-9 M=1
6     xr1 vdda vdda net69 prim_rdiffp L=2.1e-6 W=700e-9 M=1
7     xcc01 net12 out prim_cpoly AREA=4.7e-9 PERI=453.7e-6 M=1
8     mnm12 net13 net56 vssa vssa nmos1 <parameters>
9     mnm11 net56 net56 vssa vssa nmos1 <parameters>
10    mnb02 net27 net27 net21 vssa nmos1 <parameters>
11    mnpd1 net21 xpd vssa vssa nmos1 <parameters>
12    mn001 out net13 vssa vssa nmos1 <parameters>
13    mnpd2 net13 pd vssa vssa nmos1 <parameters>
14    mnc01 net13 net69 net12 vssa nmos1 <parameters>
15    mpb02 net27 net51 net158 vdda pmos1 <parameters>
16    mppd1 net158 xpd vdda vdda pmos1 <parameters>
17    mps11 net31 net158 vdda vdda pmos1 <parameters>
18    mpd11 net56 inn net31 net31 pmos1 <parameters>
19    mp001 out net158 vdda vdda pmos1 <parameters>
20    mpd12 net13 inp net31 net31 pmos1 <parameters>
21    mpb01 net158 net158 vdda vdda pmos1 <parameters>
22 .ends
```

Listing 6.5: OPAMP subckt described in Eldo language.

```
1 subckt OPAMP1 inn inp out pd xpd vdda vssa
2     xd1 (vssa vdda) prim_nwd AREA=6.13907e-9 PJ=476.6e-6 m=1
3     xi0 (vssa net69) prim_nd AREA=1e-12 PJ=4e-6 m=1
4     xi3 (net51 vdda) prim_pd AREA=790e-15 PJ=3.3e-6 m=1
5     xr0 (vssa vdda net51) prim_rdiffp l=2.1e-6 w=700e-9 m=1
6     xr1 (vdda vdda net69) prim_rdiffp l=2.1e-6 w=700e-9 m=1
7     xcc01 (net12 out) prim_cpoly <parameters>
8     mnm12 (net13 net56 vssa vssa) nmos1 <parameters>
9     mnm11 (net56 net56 vssa vssa) nmos1 <parameters>
10    mnb02 (net27 net27 net21 vssa)nmos1 <parameters>
11    mnpd1 (net21 xpd vssa vssa) nmos1 <parameters>
12    mn001 (out net13 vssa vssa) nmos1 <parameters>
13    mnpd2 (net13 pd vssa vssa) nmos1 <parameters>
14    mnc01 (net13 net69 net12 vssa) nmos1 <parameters>
15    mpb02 (net27 net51 net158 vdda) pmos1 <parameters>
16    mppd1 (net158 xpd vdda vdda) pmos1 <parameters>
17    mps11 (net31 net158 vdda vdda) pmos1 <parameters>
18    mpd11 (net56 inn net31 net31) pmos1 <parameters>
19    mp001 (out net158 vdda vdda) pmos1 <parameters>
20    mpd12 (net13 inp net31 net31) pmos1 <parameters>
21    mpb01 (net158 net158 vdda vdda) pmos1 <parameters>
22 ends OPAMP1
```

Listing 6.6: OPAMP subckt described in Spectre language.

the OPAMP described using the Spectre language. This electrical network is precisely the same specified in the Listing 6.5 but written in Spectre. Comparing these two listings described in Eldo and Spectre, we can see that the syntax is different. For example, to specify the list of pins of a component in Spectre, we use the brackets, while in Eldo, these are missing. However, it is possible to model the same physical behaviors for a given design starting from a different syntax and to know the two languages.

6.4.1 AST generation through grammars

Starting from the grammars defined for Eldo and Spectre (lexer and parser) developed per the ANTLR library, we show how the AST generated by ANTLR from our grammars are different for the same component described using the Eldo and Spectre languages. The proposed framework exploits the capabilities of ANTLR and our grammars to be able to read as input any description written in either of these languages. Through a Visitor that reads every portion of the AST that is generated by ANTLR during the parsing phase, we can populate our internal representation of the framework, which was developed in order to store all the information of both Eldo and Spectre designs, without losing any information. In particular in Figure 6.5 is reported the AST of the first component of the `OPAMP` subcircuit (see Listing 6.5), while in Figure 6.6 is

reported the AST of the first component of the OPAMP subcircuit (see Listing 6.6). From these two ASTs is clearly visible that the differences between the two trees are many. Nevertheless, these syntax differences do not impact our transistor-level model's semantic (behavior) when we describe it with a different language.

6.4.2 Semantic equivalence validation through simulation

To show an equivalent design in both Eldo and Spectre languages by exploiting the semantics commonalities, we defined the same design, the OPAMP taken from the analog benchmarks suite [1] originally defined in SPICE and adapted in Eldo and Spectre. The transient simulation is used to validate the behavior of both designs and verify the equivalence. Figure 6.7 shows the results of transient simulations of the same design, the OPAMP 6.4 described in both Eldo and Spectre. The blue line (marker x) is used to simulate the design described with the Eldo language and simulated with Eldo. The purple line (marker o) is for the simulation of the same design described with the Spectre language and simulated with Spectre. By analyzing this graph, it is possible to understand how the behaviors of the two simulations of the OPAMP are precisely equivalent. The plotted waveform has a difference of less than 10% error.

6.5 Framework applications

he proposed manipulation framework allows helping designers to design and manipulate transistor-level netlists. The framework is very flexible and extensible and allows the creation of pipelines of complex manipulation tasks. For example, it is possible to read in input Eldo, Spectre, or XML descriptions, manipulate them, and then generate other new descriptions that store all the information of the original netlist.

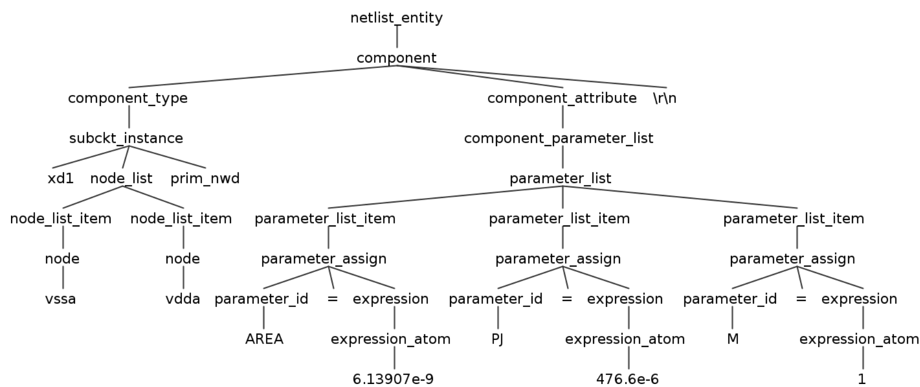


Fig. 6.5: Abstract syntax tree of the OPAMP `xd1` component described in Eldo language.

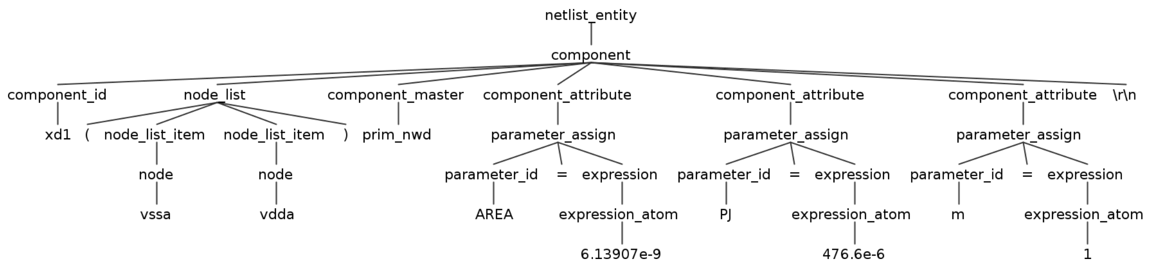


Fig. 6.6: Abstract syntax tree of the OPAMP `xd1` component described in Spectre language.

Fig. 6.7: Comparison of the VOUT behavior for the transient simulation of the OPAMP described both in Eldo (identified by a blue line with crosses) and Spectre (identified by a purple line with circles).

The proposed framework can efficiently perform several tasks:

1. Checking correctness of subcircuits internal components, e.g., check if each component has a master defined;
2. Renaming all the elements that compose a netlist, *e.g.*, components, nodes;
3. Wrapping a subcircuit inside another one, e.g., when extending a component or its interface;
4. Injection of defect models;
5. Print a netlist as an XML or JSON file;
6. Visualize a netlist as a tree.

Moreover, by exploiting the JSON front-end and back-end, it is possible to build a tools pipeline. If the appropriate front-end reads a JSON description previously generated by the framework, the internal structure will automatically be populated, allowing further code manipulation. This feature, for example, can be useful when at each call of our framework, it is required to perform manipulations of the code in sequence, that is, manipulations that strictly depend on the previous manipulation. In the following explanation, the principal features of the framework will be explained in detail.

6.5.1 Subcircuit wrapping

It is possible to inject a wrapper for a specific subcircuit. This means that the functionality of a subcircuit is embedded into another subcircuit, and the pins connected between the internal and the external subcircuit. The wrapping of a subcircuit could be done in two different ways:

- External: replace the ports of the subcircuit with new ports (port's name set by default: env_<port-name>). The new ports are connected through other netlist component or probes with the internal nodes;

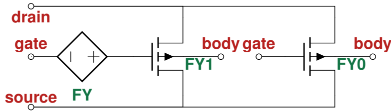


Fig. 6.8: Proposed MOS stuck-on defect model.

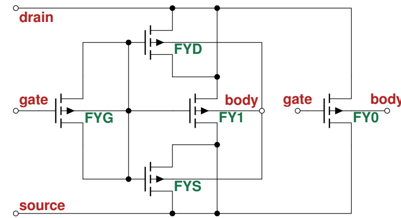


Fig. 6.9: Proposed MOS stuck-off defect model.

- Internal: replace the internal nodes of the subcircuit with new nodes (node's name set by default: `dut_<name>`) to connect them through the other netlist component or probes with the ports of the subcircuit.

For example, it is possible to wrap with the internal mode the subcircuit proposed in Listing 6.5. A possible wrapping for the `OPAMP` subcircuit is showed in Listing 6.7. All the internal nodes of the subcircuit became named dut_<node-name>. While, the pins of the wrapped subcircuit will be connected to the external subcircuit with additional component, in the example, with another subcircuit istantiated with the component of Eldo `X` necessary to istantiate subcircuits.

```
1 .SUBCKT OPAMP1_WRAPPED inn inp out pd xpd vdda vssa
2     * List of probes.
3     XPORT_1 dut_inn inn   0 Probe
4     XPORT_2 dut_inp inp   0 Probe
5     XPORT_3 dut_out out   0 Probe
6     XPORT_4 dut_pd pd     0 Probe
7     XPORT_5 dut_xpd xpd   0 Probe
8     XPORT_6 dut_vdda vdda 0 Probe
9     XPORT_7 dut_vssa vssa 0 Probe
10
11    * Original components with pin names changed.
12    xd1 dut_vssa dut_vdda primitive_nwd AREA=6.13907e-9 PJ=476.6e-6 M=1
13    xi0 dut_vssa net69 primitive_nd AREA=1e-12 PJ=4e-6 M=1
14    xi3 net51 dut_vdda primitive_pd AREA=790e-15 PJ=3.3e-6 M=1
15    ...
```

Listing 6.7: OPAMP subckt wrapped described in Eldo language.

6.5.2 Injection of defect models

Through the framework it is possible to inject defect models in a systematic way. Currently no techniques are implemented to decide which defect models to inject and where based on the failure probabilities (see the work of Sunter [63] for detailed information regarding defect random sampling), these can be added by potential users of the framework. The issue of defect models and how to inject them within transistor-level descriptions is still a hot topic of research. The *IEEE P2427 Standard* [5] is working to standardize the set of possible defects within transistor-level netlists. If we consider the stuck-on and stuck-off faults proposed in Figure 6.8 and Figure 6.9 respectively, it is easy to see these defect patterns as additional subcircuits that need to be added at precise points in our netlist. Usually these are injected into Eldo via `.ALTER` statament in order to be individually activated during simulations. It is critical to be able to activate one fault at a time and proceed with the simulation to calculate the correct coverage metrics.

6.5.3 Print a netlist as XML/JSON file

Another feature of the framework is to output the netlist read as input (and modified internally if necessary) as an XML or JSON description. This is possible by exploiting the two purpose-built back-ends. These

two formats allow to maintain the tree structure in which the information is saved within the internal representation of the framework. An example of XML description for the OPAMP subcircuit represented as schematic in Figure 6.4 is depicted in the Listing 6.8. This structure of the XML description is agnostic to the language and represents exactly the internal representation of our framework. This XML could contain all the information of the SPICE, Eldo and Spectre languages.

```
1 <CIRCUIT>
2     <SUBCKT>
3         <NAME>OPAMP1</NAME>
4         <PORTS>
5             <PORT>inn</PORT>
6             <PORT>inp</PORT>
7             <PORT>out</PORT>
8             <PORT>pd</PORT>
9             <PORT>xpd</PORT>
10            <PORT>vdda</PORT>
11            <PORT>vssa</PORT>
12        </PORTS>
13        <CONTENT>
14            <COMPONENT>
15                <NAME>xd1</NAME>
16                <PORTS>
17                    <PORT>vssa</PORT>
18                    <PORT>vdda</PORT>
19                </PORTS>
20                <MASTER>primitive_nwd</MASTER>
21                <PARAMETRIZATION>
22                    <ASSIGNMENT>
23                        <NAME>AREA</NAME>
24                        <VALUE>6.13907e-9</VALUE>
25                    </ASSIGNMENT>
26                    <ASSIGNMENT>
27                        <NAME>PJ</NAME>
28                        <VALUE>476.6e-6</VALUE>
29                    </ASSIGNMENT>
30                </PARAMETRIZATION>
31            </COMPONENT>
32        </CONTENT>
33    </SUBCKT>
34 </CIRCUIT>
```

Listing 6.8: XML fragment of the OPAMP subcircuit generated through our framework.

6.5.4 Visualize a netlist as a tree

Visualizing graphically the structure and content of a netlist, especially if it is very complex, is a feature very requested by transistor-level netlist designers. Exploiting the JSON back-end provided inside the framework and combining it with the Graphviz [64] library (based on the DOT language) it is possible to visualize graphically the netlist. Let us consider the netlist proposed in Listing 6.9. This simple example of a netlist described with the Eldo language, models the main components that we can find in a netlist, *e.g.*, subcircuit, parameter, analysis staments.

```
1 .param rvalue=3.3k
2 .param length=0.023u width=0.027u
3 c5 2 0 100p
4 r5 n3 n4 value=rvalue
5 .subckt nw n1 n2 l=length w=width
6     .param rsh=1100
7     .subckt rngps n3 n4 l2=l*2 w2=w*2
8         .param rsh2=5.0
9         r2 n3 n4 'rsh2*(l2/w2)'
10    .ends rngps
```

```
11      r1  n1  1  'rsh*(1/w)'
12      x1  1  n2  FOO  A=1  B=1  M=2
13 .ends  nw
14 .alter  a1
15      r5  1  2  10k
16      c5  2  0  100p
17      .ac  dec  15  200  1500meg
18 .end
19
20 .tran  2e-6  2e-3
```

Listing 6.9: Simple netlist modeled with the ELDO language.

By reading in input this Eldo netlist with the framework and using the back-end JSON combined with a Python script to convert it in DOT language it is possible to visualize the netlist graphically. Figure 6.10 shows a possible graphical representation of the netlist presented in Listing 6.9. With a graphical representation it is easy to understand the structure of the netlist, in this specific example we have only one circuit (root of our netlist modeled with a circle with white background), global parameters and not modeled with circles with gray background, two subcircuits modeled with circles with blue background, components modeled with circles with green background, ports modeled with circles with purple background and infinite commands to manage the simulations modeled with circles with yellow background.

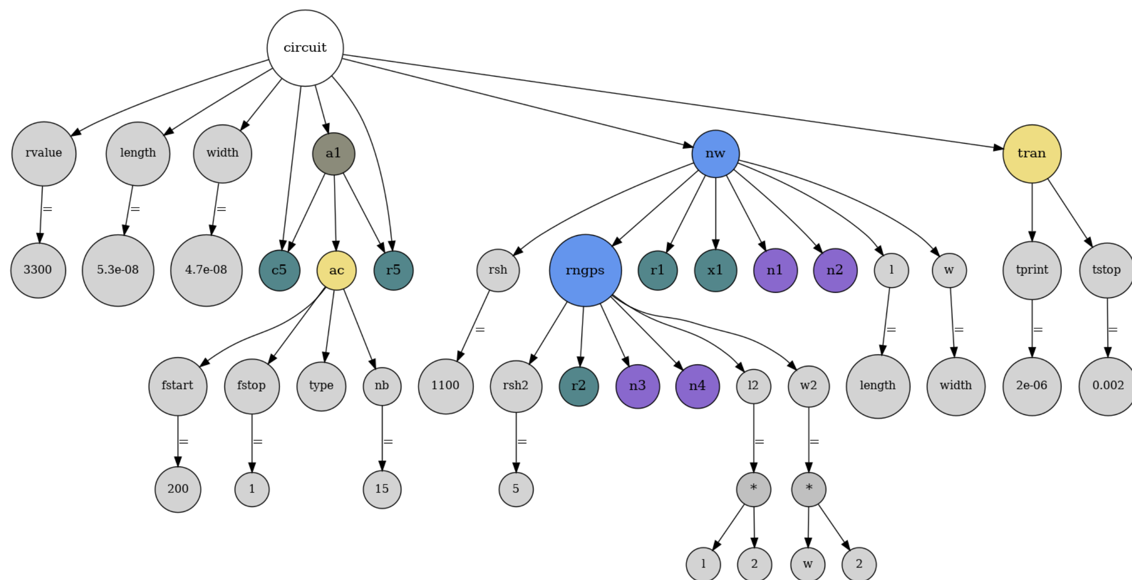


Fig. 6.10: Graphical visualization of the netlist presented in Listing 6.9 produced through the proposed framework.

7

Verilog-A Implementation of Generic Fault Templates

With functional safety being increasingly important in the development of mixed-signal products for automotive applications, EDA solutions have appeared striving to help designers in the setup and execution of fault injection campaigns. Despite the on-going work to standardize the definition of defect models and coverage calculation methods in the IEEE P2427 draft standard, there is a lack of a unified and portable method to define fault templates which can be used to inject defects in an analogue circuit. Each of the existing EDA tool-sets for fault injection proposes its own proprietary method to specify how defects should be injected. This chapter presents a Verilog-A based approach to coding fault templates, which through the compliance with the Verilog-A standard warrants portability across compatible simulators.

7.1 Background

7.1.1 Transistor-level defect models

For transistor-level models, there is no industry-accepted definition of analog defects. According to the definition provided by the IEEE P2427 draft standard [5], a defect is considered an unexpected change in the physical structure of the circuit. The difference inside the physical structure affects the function of a subsystem that can produce faulty behaviors. For example, changes in the physical structure can be considered excess of gate-oxide charges or excess of interface trap in a Metal-Oxide-Semiconductor (MOS) as described in [15]. Usually, soft and hard defects are the two types of defects considered in analog circuits. The IEEE P2427 draft standard presents differences between hard and soft defects. The random and systemic process variations in analog circuits cause changes in parametric values such as capacitance and resistance, and these parametric defects are usually considered soft defects. The hard defects generate open and short circuits between the connections of components that change the topology of a circuit. The most used components that compose an analog circuit affected by defects are the MOS that can be subdivided into MOS type -n and type -p. Figure 7.2 presents four different defects that could happen in an N-channel MOS: (a) drain open, (b) source open, (c) gate-drain short, and (d) gate-source short. Open defects are represented with a red `X`, while short with a red connection between the two MOS pins affected by the fault. The behavior of these defects depends on their impact on the primary current path between the cathode (drain) and anode (source) in MOS transistor by defining three different groups:

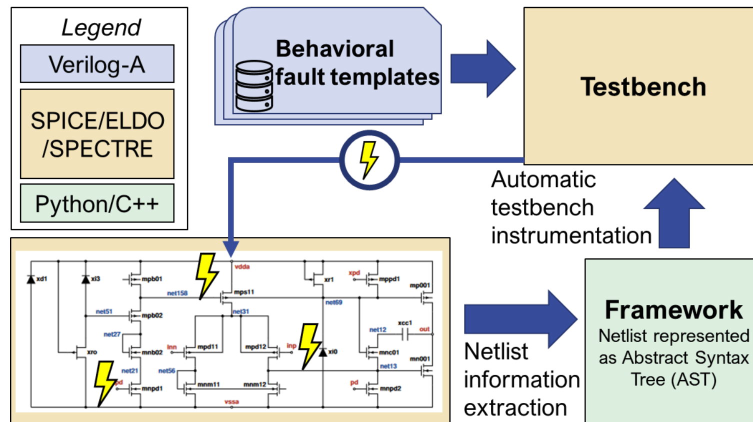
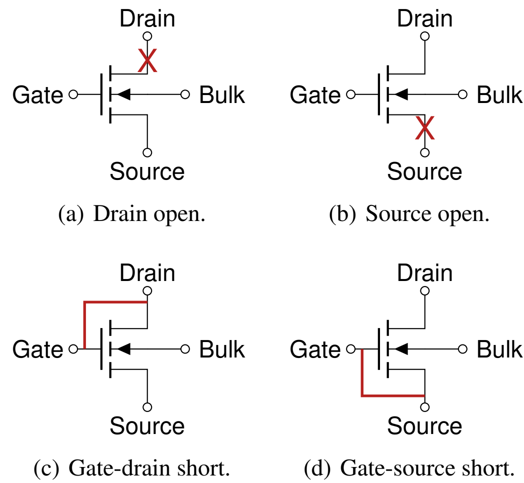


Fig. 7.1: Overview of the proposed framework that exploits the potentialities of the Verilog-A standard to describe fault templates injected directly into a transistor-level netlist.

- Hard defects can create a permanent direct or indirect conductive path between the cathode and anode. The direct path defect is referred as drain-source short, and the indirect path defect is referred as drain-gate short;
- Hard defect can also prevent the direct or indirect current flow between the cathode and anode. The defect that causes direct prevention of current is referred as drain open or source open, and the defect that causes indirect prevention of current is referred as gate-source short or gate-body short;
- Hard defect can also result in loss of control of the state of the transistor, which is referred as an open gate;

7.1.2 Verilog-A language

The design of integrated circuits is complex, and it is challenging for engineers to characterize complex behaviors. Hardware Description Language (HDL), such as Verilog and VHDL, provide various levels of abstraction to design Integrated Circuit (IC). The system behavior can be described at a high level, then gate-level descriptions can be generated using simulation synthesis programs. The interaction between digital and analog signals can be managed with mixed-signal languages. Verilog-A is a subset of Verilog-AMS [20], and it describes the behavior of analog signals with the additional functionality of interfacing some of the behavior of digital signals. Analog behaviors of the conservative systems can be described at a high level with Verilog-A, which supports conditional statements, usually part only of the programming languages. Verilog-A is supported by many Electronic Design Automation (EDA) vendors, so different descriptions written with this language can be simulated with different commercial tools. By exploiting the potentialities of Verilog-A, new defect models can be developed rapidly at the behavioral level and can be simulated into SPICE designs, as discussed in this chapter.



(a) Drain open.
(b) Source open.
(c) Gate-drain short.
(d) Gate-source short.

Fig. 7.2: Some examples of faults in a N-Channel Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET).

7.2 State of the art

In the literature, many fault modeling and injection techniques have been proposed for effective fault simulation campaigns. Fault simulation and injection can be performed at a high level as described in [65]. Applying behavioral models is one of the methods proposed in the literature to increase fault simulation speed. These behavioral models can be a set of equations relating inputs and outputs or can be several lines of micro-code. Verilog-A, Verilog-AMS, VHDL-A, VHDL-AMS, and System-C are behavioral modeling languages. VHDL-AMS simulators are powerful at a higher level and can make the modeling of analog or mixed signals easier, speeding up the simulation time. Behavioral modeling [66] and simulation of failure modes in analog blocks are possible. However, it also involves a comprehensive analysis of possible faults. While such behavioral modeling is possible using SPICE macro-models, VHDL-AMS, and other analog hardware description languages. These techniques are possibly best used in multi-level fault simulations. The idea of modeling faults at the behavioral level has been discussed in [67, 24, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77] for speed-up the fault simulation process. In multi-level fault simulations, faults can be modeled at the transistor level, whereas fault-free parts of the circuit can be modeled at the behavioral level as shown in Figure 7.3. In some articles, the multi-level fault simulation is referred such as mixed-mode simulation. Multi-level hierarchical analog fault simulation refers to the use of behavioral models for components composed of the components defined at the transistor level, fault injection at various abstraction levels, and a hierarchically handling of all different definitions of circuit components and faults during the process of fault simulation. Multi-level hierarchical analog fault simulation is a valuable method for dealing with the complexities of analog circuits and producing test signals with high fault and defect coverage [72]. The simulation time can be decreased by using behavioral models for components of the circuit. Because behavioral models have a small number of variables and system of equations for circuit equations also have less computational complexity [78]. In multi-level fault simulation, the faults can be modeled at the transistor level to improve the speed of fault simulation, while fault-free parts of the circuit can be modeled at the behavioral level [72]. The primary drawback of this technique

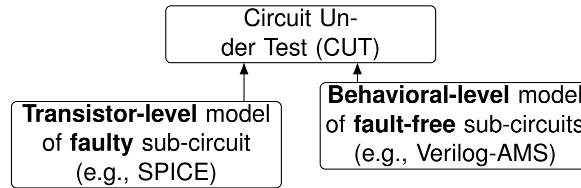


Fig. 7.3: Representation of multi-level modeling.

is that faults in a circuit can take other parts of a circuit out of their normal operating regions. A similar technique is explained in [24], with the difference that faults are modeled at the behavioral level through the Verilog-AMS language while the circuit remains described at the transistor level.

7.3 Verilog-A based fault templates

The proposed fault templates are based on defect models described by the IEEE P2427 draft standard [5], especially defects that can affect p-type and n-type MOSs will be considered. These faults are described as hard defects and can be related to manufacturing errors of an IC or to a degradation of the physical properties of a circuit due to external causes, e.g., temperature and electromagnetic radiation. This template-based fault approach is not limited to faults that may impact on a MOS but can be used to define fault templates for several SPICE components, including defects at the block level. Listing 7.1 describes some fault templates for a MOS coded in Verilog-A. In appendices B Listing B.1 and Listing B.2 show templates in details. The templates are defined inside a module named `TARGET_MOSFET_MODEL` that contains the fault-free case and ten fault models as envisioned by the IEEE P2427 draft standard. The Verilog-A module consists of a `generate-case` block driven by a `failMode` parameter, which activates the selected one fault/failure mode. The complete list of faults modeled within this Verilog-A module is: (0) fault-free, (1) drain open, (2) gate open, (3) source open, (4) bulk open, (5) gate-drain short, (6) gate-source short, (7) gate-body short, (8) body-drain short, (9) body-source short, and (10) drain-source short. Inside the `generate-case` blocks, the template can instantiate the fault-free model, which allows calling a native subcircuit model or a primitive from within the Verilog-A code. The fault template is free to play around with instance parameters of the fault-free instance, e.g., use the multiplier parameter `M` to mimic 1 out N capacitor open in an array. Normally, the fault template should reveal the same instance parameters as the original instance of the fault-free model, meaning that a given template can only be applied to models that provide a compatible set of instance parameters. In order to specify the fault-free model to use in the template, a pre-processor macro is used and is named `ORIGINAL_MOSFET_MODEL`. The construct with the `for()` loop is used to pass parameter values to the `genvar mode` in order to instantiate the different `case` blocks at compilation time. In addition, the fault template can instantiate other components to model the defect or by defining constraints on branch voltages and currents. For the injection, a `sweep` statement is used in the test bench to substitute the original instance of the fault-free model by an instance of the fault template. If the fault has a fault-free model associated with it, this substitution can be made at once at multiple places, and the fault activation can be done by setting the `failMode` parameters to the wanted value (one by one).

The main features of the proposed implementation can be summarized in four points. First, the fault templates can be made simulator-independent (depending on having the same instance parameters and model names for a component in multiple simulators). Second, the syntax is clean and simple, one template is one module, and each failure mode is a case statement. There exist highlighting modes in multiple editors for it. Third, by looking into the simulation outputs, it is simpler to assess which fault was injected by looking at a dedicated instance parameter value than by inspecting the master name. Moreover, with our approach, designers need to validate the behavior of the templates only once time and not for each different simulator used. Last, AHDL-based fault templates can issue messages into the simulator log file, enabling custom optimizations.

7.3.1 Comparison with the state of the art fault templates

The state-of-the-art methodology to inject faults into transistor-level descriptions is to use the proprietary Defectsim Suite provided by Siemens EDA [18]. Defectsim templates depend on the syntax of the underlying simulator and have a complex syntax mixing TCL and SPICE. The faults are injected one at a time by creating a faulty subcircuit for each fault injected [79]. While the activations of these faults can be obtained only by using `alter` statements and not by using parametric sweep as with the proposed fault templates. Listing 7.2 describe with pseudocode the definition of two Defectsim templates used to inject two faults. The first code describes how to inject a short fault between two ports, and it uses many internal conventions. The square brackets represent TCL function calls to calculate the Relative Likelihood (RL) and the RL activity. While the second code describes how to inject an open fault between two ports by exploiting the Defectsim tool.

Other EDA simulators have tried to implement their own fault templates. For example, into SPECTRE by Cadence, a `faults` command is added to inject faults into a netlist. This command knows about shorts (or bridges), opens, and custom faults. Custom faults allow to inject a fault block, or replace a block with another, or remove an entire block. The injected block can contain a Verilog-A module instance, but this instance cannot be used as a replacement block but must be wrapped in a subcircuit.

7.4 Manipulation framework

This section presents the language-independent framework that allows manipulating transistor-level descriptions easily. The entire framework is open-source and available on GitHub. The entire framework is written in C++, and all the functions are wrapped in Python to allow better usage. Several front-ends bring heterogeneous descriptions into an in-memory description based on the Abstract Syntax Tree (AST) generated by the parsing phase within the framework. These language descriptions that the framework reads as input can be written in Eldo, Spectre, or XML languages as output. A grammar for Eldo (the Eldo grammar also covers SPICE) and Spectre written in a format compatible with ANTLR4 are provided within the framework. After the parsing phase, the structure of the considered AST will be fully represented within

```
1  // Macro for the underlying model.
2  `ifndef ORIGINAL_MOSFET_MODEL
3      `define ORIGINAL_MOSFET_MODEL `ORIGINAL_MOSFET_MODEL
4  `endif
5  // Macro for the wrapper name.
6  `ifndef TARGET_MOSFET_MODEL
7      `define TARGET_MOSFET_MODEL anabasis_`ORIGINAL_MOSFET_MODEL
8  `endif
9  // Fault injection wrappers for MOSFET components.
10 module `TARGET_MOSFET_MODEL (D, G, S, B);
11   // Interface nodes.
12   inout D, G, S, B;
13   electrical D, G, S, B;
14   // Geometrical instance parameters.
15   parameter real l   = 0.35E-6;
16   parameter real w   = 10.0E-6;
17   parameter real ad  = 0.0;
18   parameter real as  = 0.0;
19   parameter real pd  = 0.0;
20   parameter real ps  = 0.0;
21   parameter real nrd = 0.0;
22   parameter real nrs = 0.0;
23   // Failure mode parameters.
24   parameter integer failmode = 0;
25   parameter real ropen = 1.0E9;
26   parameter real rshort = 0.1;
27   // Use genvar to instantiate the right model.
28   genvar mode;
29   for (mode = failmode; mode <= failmode; mode = mode + 1) begin
30     case (mode)
31       (0): begin
32         // FAULT-FREE
33         // ----------
34         // Instanciating the fault-free instance
35         `ORIGINAL_MOSFET_MODEL #(.l(l),.w(w),.ad(ad),.as(as),.pd(pd),
36         .ps(ps),.nrd(nrd),.nrs(nrs))  core(D, G, S, B);
37       end
38       (1): begin
39         // DRAIN OPEN
40         // ----------
41         // Internal nodes
42         electrical DD;
43         // Instanciating the fault-free instance
44         `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),
45         .pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(DD, G, S, B);
46         // Drain open
47         resistor #(.r(ropen)) defect(D, DD);
48       end
49       ...
50       (5): begin
51         // GATE-DRAIN SHORT
52         // ----------------
53         // Instanciating the fault-free instance
54         `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),
55         .pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs)) core(D, G, S, B);
56         // Gate-Drain short
57         resistor #(.r(rshort)) defect(G, D);
58       end
59       ...
60       (10): begin
61         // DRAIN-SOURCE SHORT
62         // ----------------
63         // Instanciating the fault-free instance
64         `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),
65         .pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))  core(D, G, S, B) ;
66         // Body-Drain short
67         resistor #(.r(rshort)) defect (D, S) ;
68       end
69     endcase
70   end
71 endmodule
```

Listing 7.1: Templates for MOSFET fault models.

the internal structure of the framework. This means that no information will be lost during this transcription, from the input description to the internal structure of the framework. The framework allows several manipulations of the internal structure, such as changing the master of a component or the subcircuit pins. After applying manipulation techniques, the internal structure of the framework is printed through several back-ends. It is possible to print the internal structure as Eldo or Spectre netlist or to XML or JSON files.

In the proposed workflow (see Figure 7.1 for the schematic flow), the framework is used to manipulate the netlist and the testbench (written in SPICE, Eldo, or SPECTRE) to add the lines of code required to inject

```
1 <component anabasis_IEEE2427hard_m, defect_model pre_ds_short__>
2 * IEEEP2427hard type UDAFM for M, pre_ds_short__
3 * Uses the following parameters:
4 * - FAULT_VALUE: is the value of the faulty resistor in Ohms (may vary between 0 and infinity).
5 *   Defaults to the DEFECTSIM variable 'short_defect_resistance' ( = $short_defect_resistance ).
6 *   The value corresponding to the fault-free case is hard-coded as 1.0 TOhm.
7     $defect_description = IEEEP2427 MOS drain-source short rl_transistor_v
8     $defect_type = short
9     $subckt_header
10    X_ORIGINAL $subckt_ports $subckt_name $pass_instance_parameters
11    X_FAULT $subckt_port_1 $subckt_port_3 RESFAULT FAULT_VALUE=FAULT_VALUE NOMINAL_VALUE=1.0E12
12    $subckt_footer
13    $activity_nodes = AC ($port_1 $port_3)
14    $RL = [rl_calc_proc ...]
15    $RL_activity= [rl_activity_call ...]
16 <endcomponent anabasis_IEEE2427hard_m
17
18 <component anabasis_IEEE2427hard_m, defect_model pre_sour_open_>
19 * IEEEP2427hard type UDAFM for M, pre_sour_open_
20 * Uses the following parameters:
21 * - FAULT_VALUE: is the value of the faulty resistor in Ohms (may vary between 0 and infinity).
22 *   Defaults to the DEFECTSIM variable 'open_defect_resistance' ( = $open_defect_resistance ).
23 *   The value corresponding to the fault-free case is hard-coded as 1.0 mOhm.
24    $defect_description = IEEEP2427 MOS source open rl_opens_default
25    $defect_type = open
26    $subckt_header
27    X_ORIGINAL $subckt_port_1 $subckt_port_2 DEFECTSIM_OPEN $subckt_port_4 $subckt_name $pass_instance_parameters
28    X_FAULT $subckt_port_3 DEFECTSIM_OPEN RESFAULT FAULT_VALUE=FAULT_VALUE NOMINAL_VALUE=1.0E-3
29    $subckt_footer
30    $activity_nodes = DC ($port_3)
31    $RL = [rl_calc_proc ...]
32    $RL_activity= [rl_activity_call ...]
33 <endcomponent anabasis_IEEE2427hard_m>
```

Listing 7.2: Custom Defectsim templates for short and open fault.

the Verilog-A fault templates (proposed in Section 7.3). In detail, two manipulations are performed by the framework; first, by adding the fault parameters to the subcircuit under analysis as shown in Listing 7.4 for the OPAMP circuit. Second, by adding to the original testbench the Verilog-A fault templates by using the include command, the correct model instantiation, and the sweeping on the fault parameters.

7.5 Experimental validation

The fault templates proposed in Section 7.3 and the netlist manipulation framework are validated using the circuits in the analog testbench suite provided by the IEEE P2427 standard [1]. The purpose of using this framework (already discussed in previous chapter) is to inject proposed fault templates automatically inside two MOSs present inside the Operational Amplifier (OPAMP) circuit. The schematic of the OPAMP amplifier used for the experiments is shown in Figure 6.4, and it has six input pins and one output pin (both highlighted with a red label). The netlist description of the same circuit written with the SPECTRE language is presented in Listing 7.3. For each `pmos1` and `nmos1` inside the netlist, the relative parameters are simplified with a `<params>` keyword. The netlist is composed by fourteen MOS components, respectively seven MOS type-n and seven type-p.

The proposed MOS fault templates can be applied for each MOS that composes the netlist. To show the effectiveness of the proposed approach, the fault templates are applied on two MOSs, the `mpd11` and `mpd12` components. By using the framework infrastructure presented in Section 6.3, the original OPAMP netlist (see Listing 7.3) is instrumented automatically to include the parameters `failmode_mpd11=0` and `failmode_mpd12=0` and to change the model name for the MOS under test from `pmos1` to `anabasis_pmos1`.

```
1 subckt OPAMP1 (inn inp out pd xpd vdda vssa)
2     xd1   (vssa vdda)        primitive_nwd    <params ...>
3     xi0   (vssa net69)       primitive_nd     <params ...>
4     xi3   (net51 vdda)       primitive_pd     <params ...>
5     xr0   (vssa vdda net51) primitive_rdiffp <params ...>
6     xr1   (vdda vdda net69) primitive_rdiffp <params ...>
7     xcc01 (net12 out)        primitive_cpoly  <params ...>
8     mpb01 (net158 net158 vdda vdda) pmos1     <params ...>
9     mnm12 (net13 net56 vssa vssa)   nmos1     <params ...>
10    mnm11 (net56 net56 vssa vssa)   nmos1     <params ...>
11    mnb02 (net27 net27 net21 vssa)  nmos1     <params ...>
12    mnpd1 (net21 xpd vssa vssa)     nmos1     <params ...>
13    mn001 (out net13 vssa vssa)     nmos1     <params ...>
14    mnpd2 (net13 pd vssa vssa)      nmos1     <params ...>
15    mnc01 (net13 net69 net12 vssa)  nmos1     <params ...>
16    mpb02 (net27 net51 net158 vdda) pmos1     <params ...>
17    mppd1 (net158 xpd vdda vdda)    pmos1     <params ...>
18    mps11 (net31 net158 vdda vdda)  pmos1     <params ...>
19    mpd11 (net56 inn net31 net31)   pmos1     <params ...>
20    mp001 (out net158 vdda vdda)    pmos1     <params ...>
21    mpd12 (net13 inp net31 net31)   pmos1     <params ...>
22 ends OPAMP1
```

Listing 7.3: OPAMP subcircuit modeled in SPECTRE.

```
1 subckt OPAMP1_WRAPPED (inn inp out pd xpd vdda vssa)
2   parameters failmode_mpd11=0 failmode_mpd12=0
3   ...
4   mpd11 (net56 inn net31 net31) anabasis_pmos1 <params ...> failmode = failmode_mpd11
5   mp001 (out net158 vdda vdda) pmos1 <params ...>
6   mpd12 (net13 inp net31 net31) anabasis_pmos1 <params ...> failmode = failmode_mpd12
7   mpb01 (net158 net158 vdda vdda) pmos1 <params ...>
8 ends OPAMP1_WRAPPED
```

Listing 7.4: OPAMP subcircuit modeled in SPECTRE that contains the transformations in order to be able to use fault injection wrappers.

```
1 simulator lang=spectre
2 // Other include.
3 ...
4 // Include the fault templates.
5 ahdl_include "./fault_models.va"
6 // Global node for ground.
7 global 0
8 // Testbench parameters.
9 parameters VDD=3.3 VSS=0.0 NPD=10 FUP=10G FLO=1
10 parameters failmode_mpd11=0 failmode_mpd12=0
11 // Testbench
12 SSA  (ssa 0)  vsource dc=VSS type=dc
13 DDA  (dda 0)  vsource dc=VDD type=dc
14 XPD  (xpd 0)  vsource dc=VDD type=pwl wave=[0 VDD 2u VDD 2.1u VSS]
15 PD   (pd 0)   vsource dc=VSS type=pwl wave=[0 VSS 2u VSS 2.1u VDD]
16 INP  (inp 0)  vsource dc=(VDD/2) type=sine sinedc=(VDD/2) ampl=500m freq=1M
17 CLoad (out 0) capacitor c=50f
18 // Instantiate the Device Under Test.
19 DUT  (out inp out pd xpd dda ssa ) OPAMP1_WRAPPED \
20     failmode_mpd11=failmode_mpd11 failmode_mpd12=failmode_mpd12
21 // Define transient sweep for first failure mode.
22 sweep_dpi_power sweep param=failmode_mpd11 \
23     values=[0 1 2 3 4 5 6 7 8 9 10] {
24   tran tran stop=2.3u errpreset=conservative write="spectre.ic" \
25       writefinal="spectre.fc" annotate=status maxiters=5
26 }
27 // Define transient sweep for second failure mode.
28 sweep_dpi_power sweep param=failmode_mpd12 \
29     values=[0 1 2 3 4 5 6 7 8 9 10] {
30   tran tran stop=2.3u errpreset=conservative write="spectre.ic" \
31       writefinal="spectre.fc" annotate=status maxiters=5
32 }
```

Listing 7.5: OPAMP1 testbench written with the SPECTRE language.

A sketch of the resulting netlist is shown in Listing 7.4, where these changes are visible for the components `mpd11` and `mpd12`.

Then, the next step before simulating the fault templates applied to different MOS components is to instrument the testbench. Starting from an existing testbench written in SPECTRE through the manipulation framework is easy to instrument the testbench with the additional lines required to perform the fault injection campaign. The final testbench used to perform the fault injection campaign on the MOS `mpd11` and `mpd12`

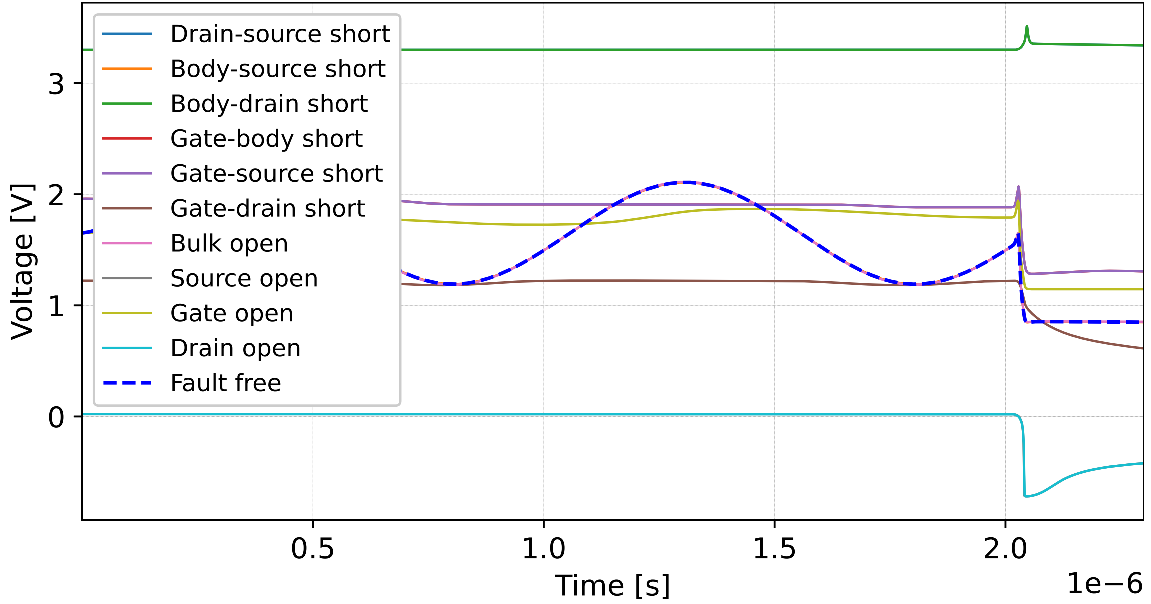


Fig. 7.4: OPAMP output waveforms for the different faults injected on the `mpd11` MOS component.

components is shown in Listing 7.5. The transformations that the manipulation framework has applied in order to build this instrumented version are on line 6, where the fault templates written in Verilog-A are included with the `ahdl_include` command that allows to include HDL descriptions into SPECTRE or SPICE compatible code. Then, the `OPAMP1_WRAPPED` Device Under Test (DUT) is instantiated in line 23, with the associated parameters `failmode_mpd12` and `failmode_mpd12` necessary to inject the fault templates on the two MOS components. The last change produced by the manipulator framework is the most important to enable the fault injection campaign. In lines 31 and 35, the sweeping on the fault parameters `failmode_mpd12` and `failmode_mpd12` are added going from 0 to 10 in order to simulate all the fault parameter templates for both the MOS components. The complete list of faults injected for each component guided by the fault template is: (0) fault-free, (1) drain open, (2) gate open, (3) source open, (4) bulk open, (5) gate-drain short, (6) gate-source short, (7) gate-body short, (8) body-drain short, (9) body-source short, and (10) drain-source short.

The transient simulation of the presented test case is performed by using the Siemens EDA Symphony simulator on a CentOS 7 machine. The resulting waveforms for the `out` pin of the OPAMP circuit are plotted in order to compare the different fault templates injected. Figure 7.4 shows the fault-free waveforms and the faulty waveforms for each fault injected on the `mpd11` MOS component. While Figure 7.5 shows the resulting waveforms for the `mpd12` MOS component. The simplicity of the presented approach allows to easily compare the results and to apply further analysis on the faulty behaviors, like fault grouping techniques.

The market of EDA fault injection solutions is fragmented across several proprietary solutions, forcing the multiplication of dedicated set-ups, fault templates, and analysis scripts. This paper proposes a generic fault template definition based on the Verilog-A standard to inject defects into transistor-level descriptions according to the guidelines of the IEEE P2427 draft standard. These fault templates are easy to code and

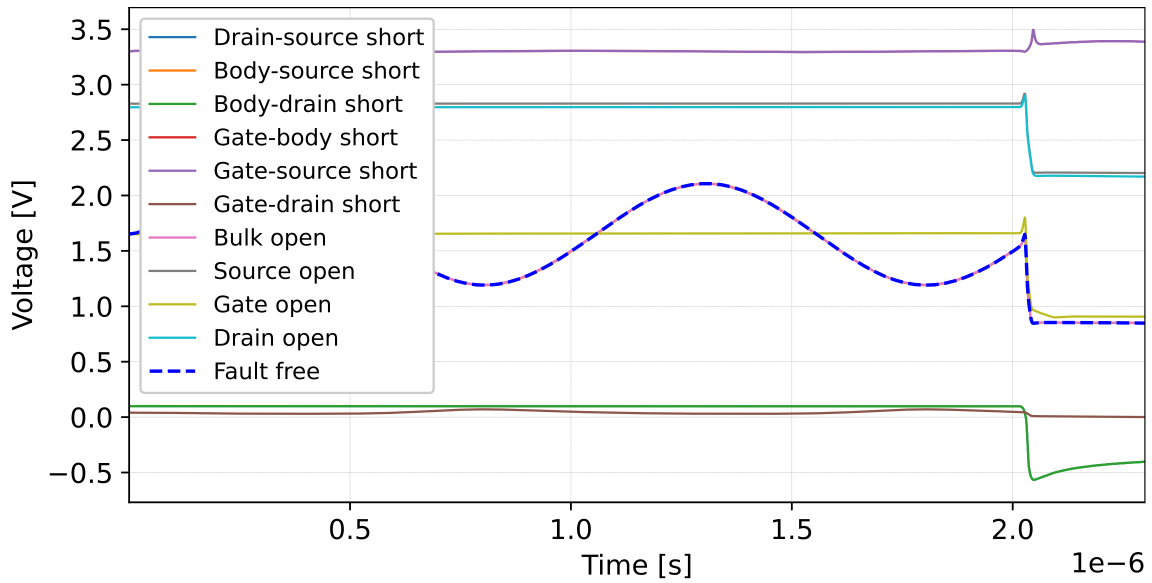


Fig. 7.5: OPAMP output waveforms for the different faults injected on the `mpd12` MOS component.

intrinsically portable across multiple simulators that support the Verilog-A standard. Furthermore, an infrastructure to automate the fault-injection process with the fault templates is presented and validated in different test cases. In future work, the template will be extended with a mechanism to compute fault-injection "weights" and pass them to the simulation scheduler.

8

Abstraction Methodologies

This chapter explain abstraction methodologies of piece-wise linear and non-linear analog models at different abstraction level. Techniques for simulating linear circuits are stable and efficient, but there are still many research gaps for non-linear circuits.

8.1 Abstraction of non-linear models (Transistor-level to Verilog-AMS)

8.1.1 Simulation of analog circuits at different abstraction levels

This section explain the different level of simulation available for analog circuits as shown in figure 8.1 .

Transistor Level Simulation

Analog circuits can be simulated at different abstraction levels. For transistor-level simulation, there are some tools [21, 9, 7] available. A netlist that represents the model is required for the simulation of analog circuits. SPICE [8] has been a widely used simulator in the past for detailed transistor-level simulation. However, it is efficient only for designs with few transistors. The modern circuits are complex and contain many million transistors; therefore, new methodologies are required to speed up and robust the simulations.

Behavioral Level Simulation

Analog circuits can be simulated at behavioral level [22] to speed up the simulation. Behavioral models can be a set of equations that describe the behavior of a circuit. The most used behavioral modeling languages are Verilog-A, Verilog-AMS, VHDL-A, VHDL-AMS, and System-C AMS. The acronym AMS stands for Analog/Mixed-Signal descriptions that combine digital and analog components as shown in figure 8.2 . In this work, the proposed methodology is based on the Verilog-AMS language. Verilog-AMS is a Hardware Description Language (HDL) specifically designed to describe mixed-signal systems. It can be considered as an extension and combination of Verilog-A and Verilog-HDL [20]. In transistor level we have one differential equation for each transistor but in behavioral level we have one (less) differential equation for whole system. Less equations are good for efficiency prospective other advantage is less time. Less time is better.

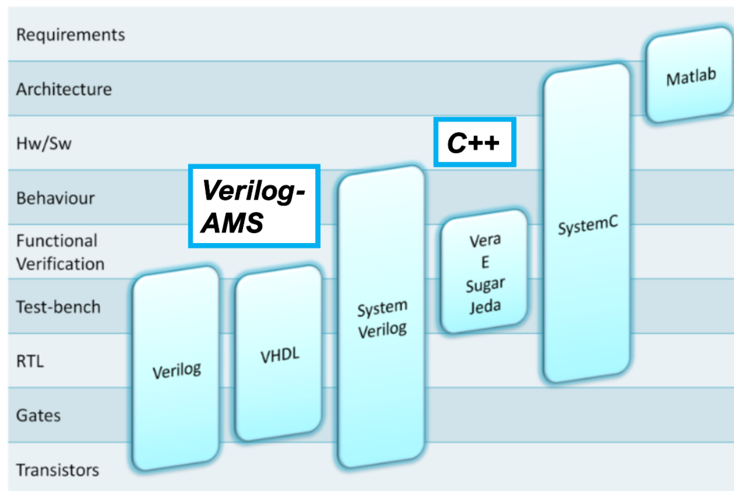


Fig. 8.1: Different Design and abstraction level [2].

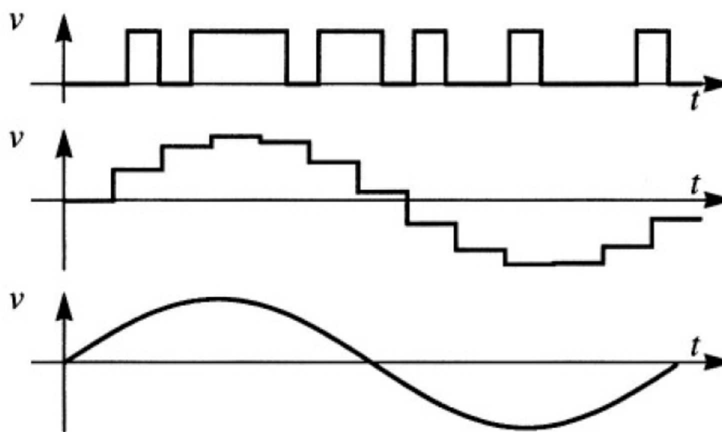


Fig. 8.2: Verilog-AMS Concepts.

Functional Level Simulation

Functional level simulations can be performed using C++ and SystemC-AMS TLM languages [23]. Functional level simulations are closer to actual hardware as compared to behavioral modeling. High abstraction level models are required to improve the simulation speedup, primarily when many simulations are used to test a large set of faults.

8.1.2 State of the art

Model Order Reduction (MOR) is a technique that can convert dynamic model of large size into small size model while preserving the behaviour of model. The resulting small size model improves the speed of statistical and functional simulation, verification, and control. Several MOR algorithms have been proposed for linear circuits such as RC and RLC networks. Application of these algorithms on non-linear circuits is a challenging task. Limited number of algorithms have been proposed for non-linear circuits. In MOR approach, computational complexity of the dynamic model is reduced for dynamic system while preserving

primary features. This technique has a potential of applying on analog circuit models. Therefore, a technique has been presented for modelling of non-linear analog circuits. We model the circuit using fuzzy differential equations and use qualitative simulation and K-means clustering to discretion efficiently its state space. MOR steps are refined using conformance check while guaranteeing accuracy and simulation acceleration[80].

Reduced-order types of models are a useful replacement for native element specifications since they minimize simulation complexities dramatically. While deriving lower order formulations for linear passive elements going through the system is a well-established method, finding similar concise representations for nonlinear parts is already a work in progress. The traditional Hammerstein–Wiener model structure identification [81], trajectory piecewise linear approaches [82], Volterra series-based methods [83], X Parameters [84], and neural networks [85], [86] are those of the modelling approaches which have been presented to approximation generic nonlinear circuit blocks (CBs). In terms of design complexity, precision, and creation time requirements, each one of these identification approaches has its own set of limitations and benefits. This means that, depending on the attributes of a reference circuit, the best modeling method for the applications of interest should be selected [87]. This work [87]provided a new method for generating reduced-order LPV macro models, which may be used to forecast the time-domain input/output behavior of slightly nonlinear analog CBs quickly. The circuit reaction may be reproduced using this structural model in non-stationary small-signal situations, in which the operating point varies periodically during execution. When compared to transistor-level circuit specifications, the system is formed as a reduced-order SPICE-compatible netlist, which promises a substantial speed boost in transient analysis. Numerous numerical simulations on voltage regulator CBs show that the model's effectiveness comes without losing its quality. Table 8.1 shows comparison of different data driven behavioral modeling present in literature.

Table 8.1: State of the art on Data driven Behavioral Modeling.

Criterion	Bradde'21[87]	Tarraf '19[80]	DeJonghe '12[88]
Source Simulator	CDS Spectre	CDS Spectre	MGC Eldo
Data Collection	AC transfer functions @ OP	MNA Matrices @ OP	MNA Matrices @ OP
Data Processing	Vector Fitting	PZ Extraction	Vector Fitting
Analog state	Continuous	Discrete (LTI regions)	Continuous
Replay Engine	Polynomial Interpolation	Piece wise (polyhedra)	AI generated Expressions
Target Simulator / HDL	LTSpice and C++	VerilogA	Eldo w. VHDL AMS and C++
Accuracy (NRMSE)	Good	Medium Glitches @ boundary X ings	Good
Speed up	50 -100x	30-120x	10-110x

8.1.3 Automatic data driven behavioral model generation flow

This section presents steps of the abstraction methodology shown in figure 8.3 used to abstract transistor level models into behavioral models. Proposed methodology is also data driven modeling approach. There are some constraints to show all the steps and results in details because this work is with the collaboration of Sydelity B.V., Kruisem, Belgium company and its not published yet and it is under testing phase and also under submission work. The steps of proposed methodology are :

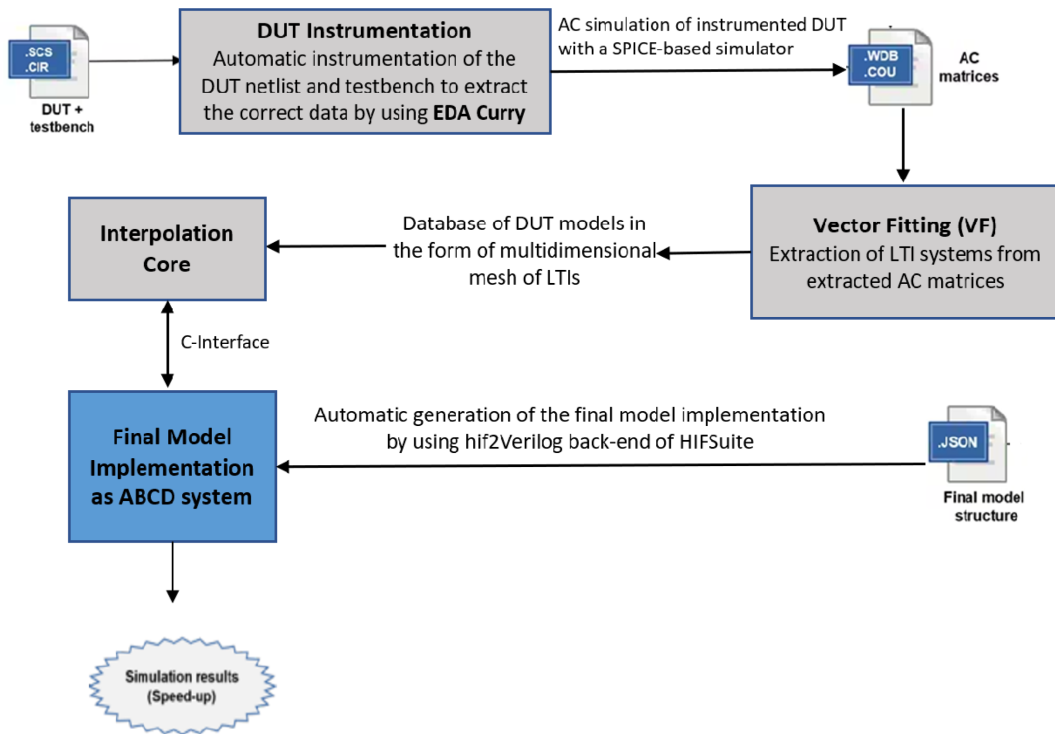


Fig. 8.3: Proposed methodological flow divided into its macro-block parts.

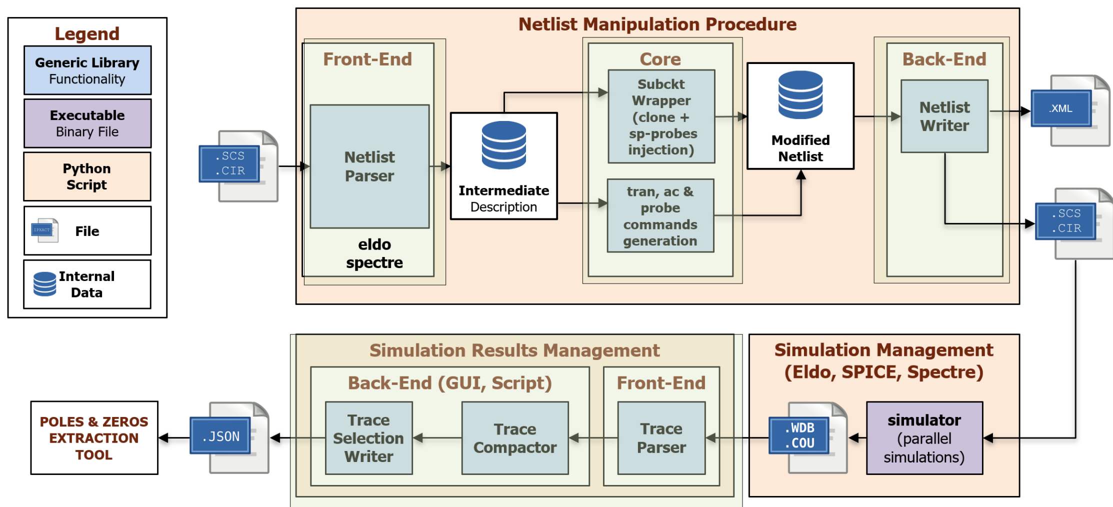


Fig. 8.4: Data extraction.

- EDACurry framework (previously discussed fault injection tool see 6.3) used to instrument the netlist with the probes and simulation commands.
- Simulation of the instrument netlist through a global transient simulation and one AC analysis for each O.P, with SPICE-based simulators see figure 8.4 ;

- Extraction of Linear Time Invariant (LTI) systems through the vector fitting applied on the AC simulation data (different sets of ordinary differential equations). By using vector fitting technique an ABCD system is built for each operating point;
- Creation of a mesh of LTI systems to discretize the space of the results, during the simulation based on the voltages on specific ports of the model the most suitable LTI system is selected and produce the output of the system, this procedure is implemented inside the interpolation core by using bycentric interplolation;
- Automatic generation of the final code that contains the calls to the interpolation core. This final model could be written in different languages: VHDL-AMS, Verilog-AMS, Modelica, C++ as it describe an ABCD model. To be able to run properly the final model the used simulator needs to have a support of the derivatives.

8.2 Abstraction of Piece-Wise linear models (Verilog-AMS to C++)

Functional level simulations can be performed using C++ and SystemC-AMS TLM languages [23]. Functional level simulations are closer to actual hardware as compared to behavioral modeling. High abstraction level models are required to improve the simulation speedup, primarily when many simulations are used to test a large set of faults. Proposed abstraction methodology can abstart linear , Piece-wise linear and as well as non-linear equation (for specific models) as shown in Figure 8.6 . In this section only details of abstraction of Piece-wise is explained.

8.2.1 State of the art

Most of the work present in literature is related to the abstraction from transistor to behavioral level or vice-versa. Moreover, many of these works are related to the functional level abstraction of only linear circuits [89]. Conversely, in proposed work, Piecewise Linear (PWL) models of analog circuits derived from non-linear descriptions are abstracted . This work is an extension of the methodology presented in [89] to enable the support of PWL descriptions. In [90] an automated abstraction technique has been presented. However, this technique was related to the abstraction of analog models described at transistor level to the behavioral level. Another work that proposes an abstraction method from transistor-level models was discussed in [91]. In [92] a hierarchical abstraction methodology was explained. An optimum specification of analog circuits was translated from higher to lower level abstractions to design the analog circuits. In [23] a SystemC-based simulator for functional simulations is presented. In electronic design and testing,the simulation speed of analog components is crucial. Moreover, the simulation of heterogeneous components embedded in a Virtual Platforms (VP) needs to be fast and accurate. Often, the analog components are non-linear, and simulating them is not easy to ensure the model's convergence. In this context, techniques for simulating linear circuits are stable and efficient, but there are still many research gaps for non-linear circuits. There are no systematic methods available to solve non-linear equations efficiently. One standard

method is to solve these non-linear equations by describing them as a PWL models. PWL techniques approximate non-linear functions with a set of linear functions. This is common to most solver methods: they linearize to compute an inverse matrix, finding which direction to move to satisfy the equations.

In this chapter, an abstraction methodology for PWL models is explained. By using this methodology, it is possible to abstract a piecewise model described with the Verilog-AMS language to the C++ language. These C++ models can be integrated into VPs. A half-wave rectifier, ideal relay and memristor model are selected to explain and validate the methodology. Furthermore, to show the effectiveness of the proposed technique, the abstracted models are abstacted into VPs . For example the half-wave rectifier is integrated into a MEMS accelerometer. Moreover, the accelerometer is integrated into a VP will discuss in chapter to show the effectiveness of the functional simulation.

8.2.2 Piecewise Linearization

Piecewise approximation is a method in which a function $g(x)$ is constructed to fits a objective function $f(x)$ that is non-linear as shown in Figure 8.5 . By adding extra variables (binary or continuous) and constraints, it is possible to reformulate the actual problem for the approximation of single-valued function in terms of linear segments sequence [93, 94]. A piecewise linear approximation will approximate a function $g(x)$ made up of a sequence of linear systems on the same intervals as the $f(x)$ function is defined. Usually, transistor-level simulators during the transient simulations compute the solution on the first time point and then linearize the system of equations to estimate the jump they will make to arrive at a subsequent time point. Using linearization, simulators can compute the direction towards which it has to move to satisfy all the equations at the next time point. For the first time, it usually takes the case with zero potential differences everywhere across the system (all inductors and capacitors discharged), which has a trivial solution of all currents equal zero. Then, the solver moves in small steps in the time domain. If the solver makes a step and then finds that at the new trial point it has computed, the circuit equations are not correctly solved (meaning there is a residual error due to the approximation in the linearization phase). Then, it will invalidate that trial point and try a new one, typically after reducing the time-step, making a smaller jump, and reducing the chance of making an error. It is fundamental to understand the behavior of the circuit at equilibrium points (DC operating Points) [95]. A circuit can be linearized at any point, but typically solvers choose to perform it at specific equilibrium points.

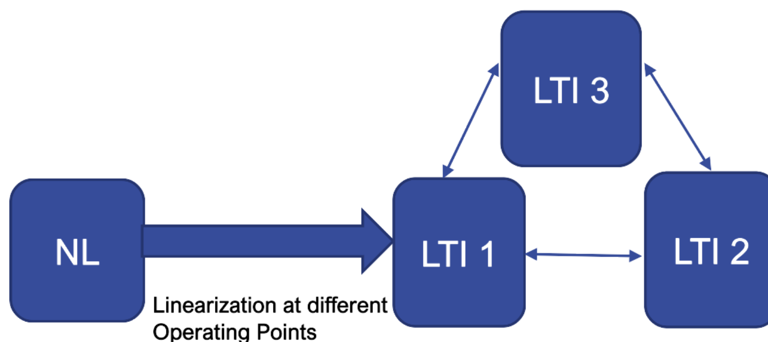


Fig. 8.5: Piece-Wise Linearization

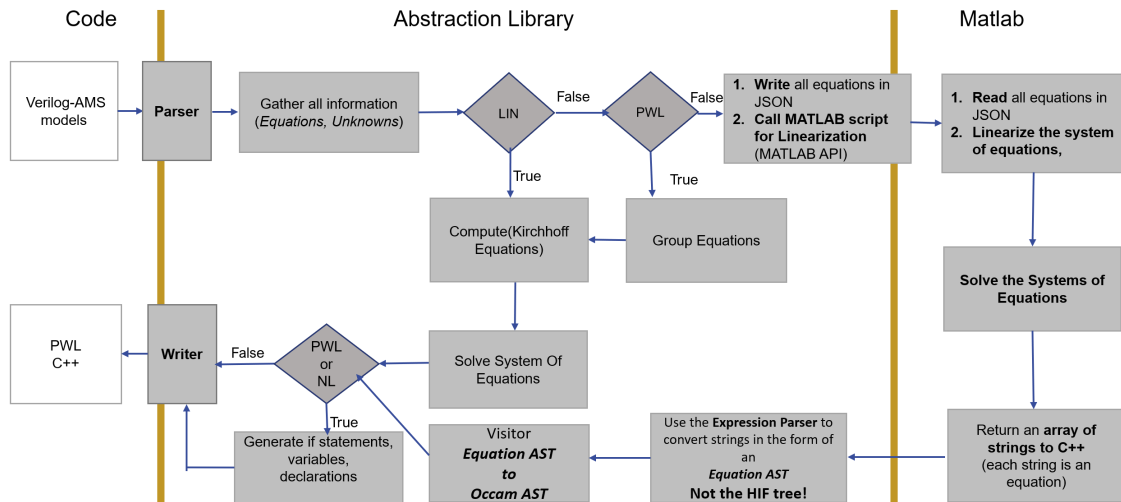


Fig. 8.6: Abstraction Methodology

8.2.3 Abstraction methodology

This section describes the steps required to abstract from a PWL Verilog-AMS model to a C++ functional model. In the first part, the software infrastructure is shown, while in the second, all the methodology steps are described in detail.

Software infrastructure

The software infrastructure required to abstract Verilog-AMS PWL models is described in Figure 8.7. The library that implements the methodology is built with the C++ language. The software infrastructure consists of the following steps built in different tools:

- The starting point is Verilog-AMS code that consists of both linear and non-linear equation models (see Listing 8.1). First, it linearizes the non-linear equations into piecewise linear models. Piecewise linear models can have both `if`/`else` and switch statements as shown in Listing 8.2.
- Then, the piecewise linear model and the linear models are parsed and transformed into an intermediate description. This intermediate description is an Abstract Syntax Tree (AST) written in XML that stores all the pieces of information of the original model by structuring them in a tree.
- When the information of the original model is stored inside the intermediate description, it is required to gather all the information to manipulate the model equations.
- Additional equations are computed through the Kirchhoff's current law (KCL) and Kirchhoff's voltage law (KVL) to generate the branch current and branch voltages equations.
- Then, the set of equations is simplified before solving them. Only *linearly independent equations* and their unknowns can be passed to the solver.
- In this step, we have the simplified set of equations and unknowns. Starting from it, the equations are symbolically solved through the GINAC C++ library [96]. Other libraries and tools can be used to solve the complete system of equations, like, MATLAB core or the SymPy library written in Python.

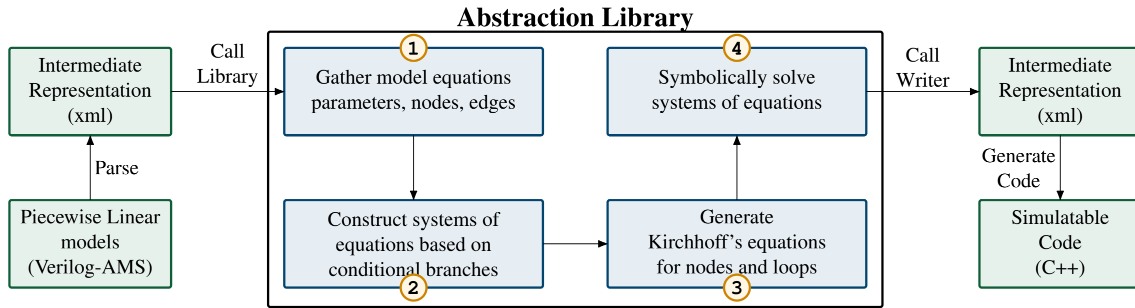


Fig. 8.7: Proposed methodology flow to abstract PWL descriptions.

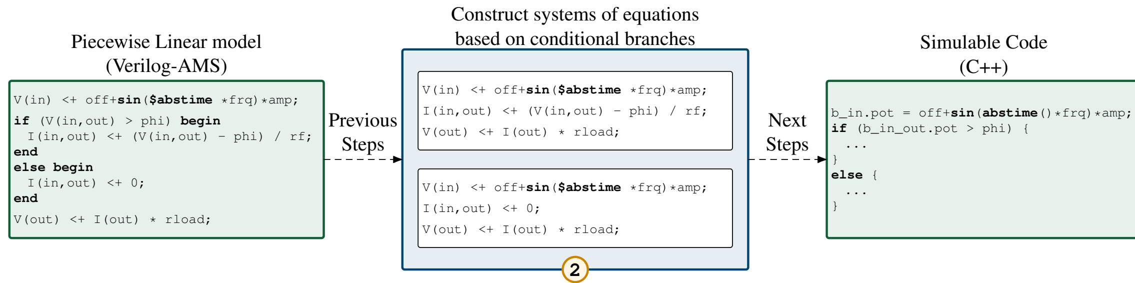


Fig. 8.8: Process flow used to abstract the model equations from Verilog-AMS to C++.

- The solved group of equations is re-written inside the intermediate description (XML file). Then, through a back-end, it is possible to generate the final abstracted C++ code. Moreover, the generated C++ code then is easily integrable into a VP.

Equations abstraction methodology

The entire abstraction methodology used to process and abstract the equations is schematized in the central part of Figure 8.7. Four main steps of the abstraction methodology can be identified:

- In the first step, the library takes in input all the equations, nodes, and branches of the original parsed model. The original model is written using Verilog-AMS language and contains an analog circuit's piecewise linear representation. An analog circuit can be seen as an electrical network composed of nodes, branches, and equations. Starting from the equations, all the `if – else` statements are identified and parsed. This is a fundamental step of the methodology because for each `if – else` statement identify a different electrical network, and for each electrical network different equations are associated. This means that the original circuit expressed as an electrical network change internally what an `if – else` statement is present in the model (Figure 8.7 step 1).
- For each `if – else` statement the library build an electrical network for each `if – else` statement. The idea is similar to the concept of unrolling a loop inside the C++ language through the `#pragma unroll` statement. With the `#pragma unroll` statement, it is possible to speed up the code many times. In the same manner, with the proposed process of building a different electrical network for each `if – else` statement, it is possible to speed up the simulation. At the simulation time, the final C++ code containing the abstracted model enables one electrical network when the dynamic requires to

change from a linear system to another (from one electrical network to another). While the commercial analog circuit simulator (based on SPICE) build the system of the equation at run time, this means that the system of equations can change at run time when the execution of the model arrives at `if – else` statements (Figure 8.7 step 2). This second step is the core of the proposed methodology and is described through the equations of a half-wave rectifier in Figure 8.8.

- In the third step, all the different electrical networks (composed of different equations) are symbolically solved one at a time through the GINAC library. The equations passed to the solver are the original, KCL and KVL, and the unknowns. The KCL and KVL equations are computed starting from the original equations for each node and branch of the electrical network. If the original model contains equations outside the `if – else` statement, these equations are duplicated inside each different electrical network; this is necessary when we solve the equations symbolically because the final model generated depends on all the components of the system (Figure 8.7 step 3).
- While, in the last step, the final C++ model is built starting from the solved equations. In the final code, a different system of equations for each electrical network is present. After the solving phase, if a component should be added to the electrical network, all the abstraction methodology should be re-run because the solved equations depend on the structure of the abstracted analog circuit. When the final C++ model is simulated, only one electrical network is active in a precise simulation time. This means that the final model is extremely fast because described with the C++ language and because all the possible system of equations of the model are explicated before the execution of the model (Figure 8.7 step 4).

Methodology Validation

Analyzed Models

The methodology is validated through different analog PWL models some of them are mentioned here:

- A half-wave rectifier taken from the book [97] that allows only one half-cycle of a voltage waveform to pass, blocking the other half-cycle;
- A memristor taken from the article [98] that model a non-linear component that link the electric charge and the magnetic flux natures. In appendices A Listing A.2 show verilog-AMS code of memristor that has been abstracted;
- An ideal relay taken from the book [97].

All of these models are simulated with a common test bench (see schematic in Figure 8.9). The testbench is modeled with a voltage source connected to the model and a load resistance to improve the convergence of the simulation.

Methodology validation

The entire methodology is validated with a model of a half-wave rectifier. The model is composed of a voltage source, a diode, and load resistance. The diode is the main component in this model, and its behavior

```
1 `include "disciplines.vams"
2 module hwr_nonlin(a, c);
3   electrical inout a, c;
4   parameter real is = 10f from (0:inf);
5   parameter real r  = 0   from [0:inf);
6   analog begin
7     I(a,c) <+ is * (limexp((V(a,c) -r*I(a,c)) / $vt) - 1);
8   end
9 endmodule
```

Listing 8.1: Non-linear model of a half-wave rectifier written in Verilog-AMS.

```
1 `include "disciplines.vams"
2 `include "constants.vams"
3 module hwr_lin(a, c);
4   electrical inout a, c;
5   parameter real rf  = 27      from [0:inf);
6   parameter real phi = 0.69629 exclude 0;
7   analog begin
8     if (V(a,c) >  phi)
9       I(a,c) <+ (V(a,c) - phi) / rf;
10    else
11      I(a,c) <+ 0;
12   end
13 endmodule
```

Listing 8.2: Verilog-AMS PWL model for a half-wave rectifier.

```
1 `include "disciplines.vams"
2 `include "constants.vams"
3 module top_level;
4   electrical in, out;
5   parameter real offset = 0.0  from [0:inf);
6   parameter real freq   = 1e06 from [0:inf);
7   parameter real ampl   = 3.3  from [0:inf);
8   parameter real rl     = 1e06  from [0:inf);
9   hwr_lin (in, out); // Half-Wave Rectifier instance.
10  analog begin
11    V(in) <+ offset + sin($abstime * freq) * ampl;
12    V(out) <+ I(out) * rload;
13  end
14 endmodule
```

Listing 8.3: Verilog-AMS top level.

Listing 8.4: C++ analog branch structure.

```
1 typedef struct analog_branch {
2   double pot, flw; ///< Potential and Flow value.
3 } analog_branch_t;
```

is non-linear. A diode works in two regions named: forward biased and reversed biased. A PWL is used for the approximations of the diode characteristic curve in the form of linear segments. First, we have checked the accuracy between both non-linear and PWL models of the half-wave rectifier to check the correctness of the PWL model. For this step, both models were simulated on a SPICE-based simulator. After that, the abstraction methodology is applied to the half-wave rectifier model. The abstraction flow applied on the model equations is schematized in Figure 8.8.

Listing 8.1 represents the Verilog-AMS non-linear description of a diode. This description contains the name of the module, the contribution statement (comprises non-linear equations), disciplines (electrical), parameters, and branches. The other two components, resistor, and voltage source, are connected with the diode, so two more linear equations are present. Listing 8.2 is the PWL model of the diode.The model presented in the Listing 8.2 is simulated by the top-level presented in the Listing 8.3 that stimulate the model. After applying all the steps of the methodology detailed in Figure 8.8, through the C++ back-end,

Listing 8.5: C++ abstracted code of half-wave rectifier. Variable name that start with 'b_' are of type *analog_branch_t*.

```
1  void Rectifier::simulate() {
2    constexpr double offset = 0.0, fre = 1e06, ampl = 3.3;
3    constexpr double rf = 27, phi = 0.69629;
4    // Circuit behaviour.
5    b_in.pot = off + std::sin(abstime() * freq) * ampl;
6    if (b_in_out.pot > phi) {
7      // Voltage source.
8      b_in.flw = (b_in.pot - phi) / (rf+rl);
9      // Diode.
10     b_in_out.pot = (b_in.pot * rf + phi * rl) / (rf+rl);
11     b_in_out.flw = (b_in.pot - phi) / (rf+rl);
12     // Load resistance.
13     b_out.pot = (b_in.pot - phi) * rl / (rf+rl);
14     b_out.flw = (b_in.pot - phi) / (rf+rl);
15   } else {
16     // Voltage source.
17     b_in.flw = 0;
18     // Diode.
19     b_in_out.pot = b_in.pot;
20     b_in_out.flw = 0;
21     // Load resistance.
22     b_out.pot = 0;
23     b_out.flw = 0;
24   }
25 }
```

Listing 8.6: C++ simulation manager.

```
1  static double &abstime() {
2      static double __abstime = 0.0;
3      return __abstime;
4  }
5  int main(int argc, char *argv[]) {
6      // Instantiate the analog component.
7      Rectifier r;
8      // Define simulation parameters.
9      double time_step      = 1e-06;
10     double simulated_time = 1;
11     // Simulate.
12     while (abstime() < simulated_time) {
13         // Simulate the model.
14         r.simulate();
15         // Advance the time.
16         abstime() += time_step;
17     }
18     return 0;
19 }
```

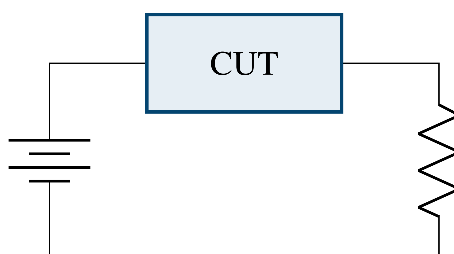


Fig. 8.9: Circuit simulation setup.

the final simulable code is generated. The final generated C++ code is represented in the Listing 8.5, while the analog branch structure is showed in the Listing 8.4. Moreover, this final C++ code is simulable through a simple C++ main code (see Listing 8.6).

Simulation Results

The models previously described are simulated with a PC equipped with an i7-9700 processor with 3.00 GHz, 16 Gb of RAM ddr4, and Ubuntu 20.04.2 LTS. A comparison of different simulations of the same

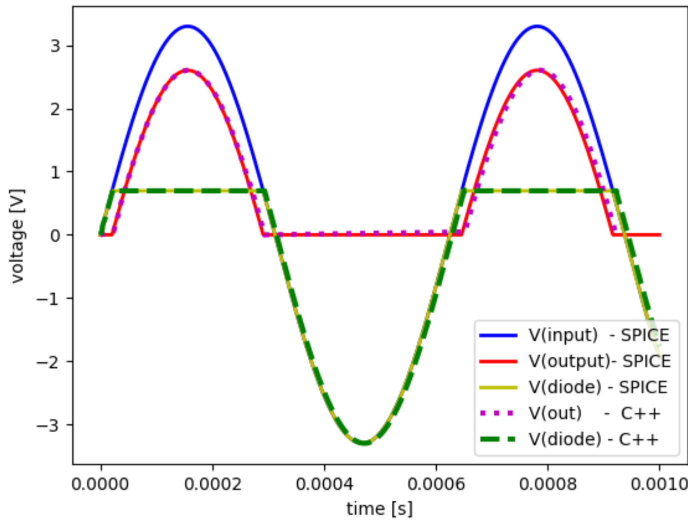


Fig. 8.10: Simulation results for the half-wave rectifier, the memristor, and the ideal relay.

model described at different abstraction levels and simulated with different modes is presented. For all the models, two different simulations are performed:

- The non-linear Verilog-AMS model simulated through a SPICE-based simulator;
- The abstracted C++ model built starting from a PWL description written in Verilog-AMS.

Both the simulations are run for 1 *ms* with fixed time-step set to 1 *us*. Figure 8.10 presents the simulation traces of the half-wave rectifier simulated with a SPICE-based simulator and with the proposed C++ model. The SPICE traces are obtained with the simulation of the non-linear Verilog-AMS model simulated with a SPICE-based simulator. While the C++ traces are obtained with the simulation of the PWL model abstracted and simulated in C++. Inside the graph, the V(input) waveform is the voltage input of the model (a *sin* wave), the V(output) waveform is the output voltage of the system, and the V(diode) waveform is the voltage on the diode. The condition that force the switch between the forward bias and the reverse bias in the diode is set to 0.69629 *V* (built-in junction potential (V)). The simulation traces of the two models are equivalent with an error around 0.1%. This implies that the abstracted model is fully equivalent with respect to the original non-linear model. Moreover, the simulation of C++ code has a speed-up around 2.65x in comparison with the simulation of Verilog-AMS code with a SPICE-based simulator. This speed-up obtained is not related only to the level of abstraction of the C++ language but also to how the methodology for the abstraction of PWL models is built. The methodology exploits the concept of creating before the simulation all the possible solved system of equations that allows describing all the model behaviors. Table 9.1 shows the simulation results for the half-wave rectifier, the memristor, and the ideal relay. The simulation time obtained by simulating the heterogeneous model (Verilog-AMS) and the abstracted model (C++) is presented for each model.

Table 8.2: Simulation results for the half-wave rectifier, the memristor, and the ideal relay.

Model Version	Half-wave rec.		Memristor		Ideal Relay	
	Time	Sp.	Time	Sp.	Time	Sp.
Heterogeneous	2.02s	-	0.57s	-	0.32s	-
Abstraction	0.76s	2.65x	0.005s	114x	0.002s	160x

8.3 Abstraction of Non-linear models (Verilog-AMS to C++)

8.3.1 Study of different linearization techniques for non-linear models

Linearization of non-linear models depends on how non-linear equations are formulated. Before starting the implementation of non-linear abstraction flow different linearization techniques are studied. If system is dx/dt = F(x(t), u(t)) then linearization is about taking first order taylor expansion of F(.,.) at (x0, u0) some operating point. In Matlab Simulink has an API for its models that it calls to get derivatives and then has a way to combine them taking connectivity into account and produce the final Jacobian of F(). Circuit solvers have that as well. And then a way is required to approximate dx/dt in function of past values of x and a future not yet known value (the latter being also used in the linearization of F()) and then a linear systems of equations can bulid allowing to solve for the new unknown values. Some solution methods prefer to integrate the equation over time instead of approximating dx/dt . The exact algorithm behind the linearization of the system in Simulink® Control Design software is implemented based on a block-by-block technique. The blocks are linearized individually and then the linearization of blocks is executed to achieve e overall linearized system. The state level and input of each block are determined by Jacobian and operating point. Derivatives are acquired from model APIs and they are combined to build the final Jacobian of F() while taking care of connectivity. In [99] a function is designed to partition any equation curve

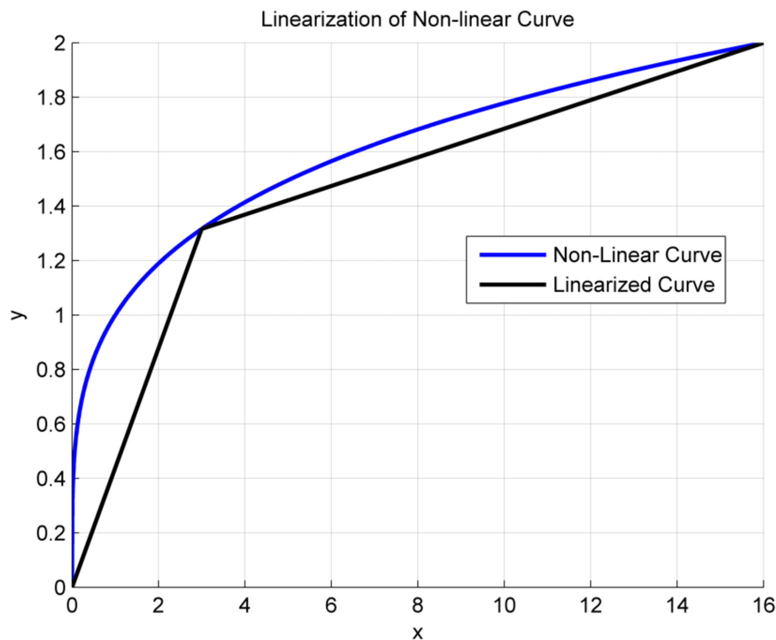


Fig. 8.11: Curve Linearization Example

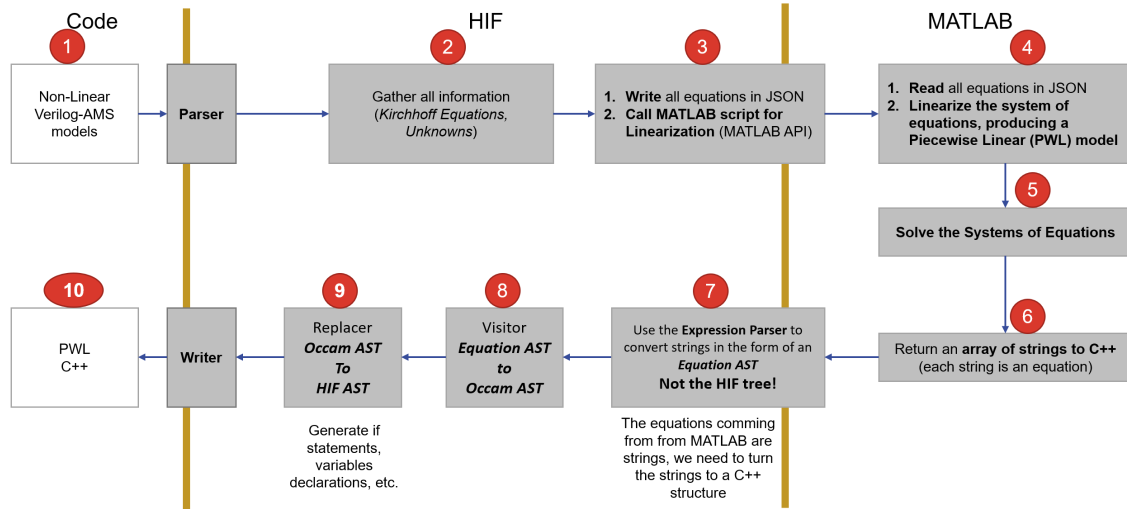


Fig. 8.12: Flow of abstraction of Non-linear Verilog-AMS models.

to considerable nonlinearity in to the user-defined linear categories depending on the points selected for linearization as shown in Figure 8.11. Nonlinear systems can also be linearized by using multiple piecewise linear equations as discussed in [100].

8.3.2 Simulation of Non-linear models by numerical techniques

To understand the behavior and to gain the knowledge of different differential equations i have simulated different Verilog-AMS models in C++ library by using different numerical solving methods. These methods are Euler, Rk4 , Adaptive Euler, Adaptive Rk4. These methods are used to solve the differential equations. Simulation code is available on GitHub:*https://github.com/Galfurian/stunning-chainsaw/tree/main/example*. One of the example of non-linear Verilog-AMS code is shown in appendices C in Listing C.1. Listing 8.7 is a piece of Verilog-A code from currently running project that i already mentioned its a collboration with Sydility. Numarical solving methods are computationally expensive so i moved to symbolically solving technique in proposed abstraction methodology..

8.3.3 Abstraction Methodology

This section describes the steps required to abstract from a non-linear Verilog-AMS model to a C++ functional model. Some steps are common and already discussed in piece-Wise linear abstraction methodology because its the same library that can extract non-linear models.

Software infrastructure

The software infrastructure required to abstract Verilog-AMS non-linear models is described in Figure 8.12. The library that implements the methodology is built with the C++ language and implemented on HIF suite [101] . The software infrastructure consists of the following steps built in different tools:

- The starting point is Verilog-AMS code that consists of both linear and non-linear equation models .

- Then, the non-linear and the linear models are parsed and transformed into an intermediate description. This intermediate description is an Abstract Syntax Tree (AST) written in XML that stores all the pieces of information of the original model by structuring them in a tree.
- When the information of the original model is stored inside the intermediate description, it is required to gather all the information to manipulate the model equations.
- Additional equations are computed through the Kirchhoff's current law (KCL) and Kirchhoff's voltage law (KVL) to generate the branch current and branch voltages equations.
- Then all equations are stored in JASON file to send matlab Engine [102] to linearize the Non-linear equations . The link of this Matlb Api is : *https://www.mathworks.com/help/matlab/cpp-engine-api.html* . Buy using this Matlab engine Matlab functions can called in C++. For Example to start the matlab in C++ startmatlab() function is required and to call the matlab functions in C++ feval() is used. In this step linearized equations are generated .
- In this step, we have the simplified set of equations and unknowns. Starting from it, the equations are symbolically solved through solve() function . this function sybbolically solve the equations .
- Then some steps are required to convert matlab format into our C++ library format.
- The solved group of equations is re-written inside the intermediate description (XML file). Then, through a back-end, it is possible to generate the final abstracted C++ code. Moreover, the generated C++ code then is easily integrable into a VP.

```
1
2 `include "constants.vams"
3 `include "disciplines.vams"
4
5 module Case_Gm_LinDyn2P( GND, U1, U2 );
6
7 // U1terface nodes
8 inout GND, U1, U2;
9 electrical GND, U1, U2;
10
11 // Internal nodes
12 electrical X1, X2;
13
14 // Interface parameters
15 parameter real R2bValue = 1.0 ;
16 parameter real R2aValue = 1.0 ;
17 parameter real R1bValue = 1.0 ;
18 parameter real R1aValue = 1.0 ;
19 parameter real GmValue = 0.0 ;
20 parameter real C2bValue = 0.0 ;
21 parameter real C1bValue = 0.0 ;
22
23 // Local variables:
24 // A-matrix
25 real A11, A12, A21, A22;
26
27 // B-matrix
28 real B11, B12, B21, B22;
29
30 // C-matrix
31 real C11, C12, C21, C22;
32
33 // D-matrix
34 real D11, D12, D21, D22;
35
36 // E-matrix
37 real E11, E22;
38
39
40 // Initialize all static variables
41 analog initial begin
42   A11 = -1.0*(R1aValue + R1bValue)/(R1aValue*R1bValue);
43   A12 =  0.0;
44   A21 = -1.0*GmValue;
45   A22 = -1.0*(R2aValue + R2bValue)/(R2aValue*R2bValue);
46
47   B11 =  1.0/R1aValue;
48   B12 =  0.0;
49   B21 =  0.0;
50   B22 =  1.0/R2aValue;
51
52   C11 = -1.0/R1aValue ;
53   C12 =  0.0;
54   C21 =  0.0;
55   C22 = -1.0/R2aValue;
56
57   D11 =  1.0/R1aValue;
58   D12 =  0.0;
59   D21 =  0.0;
60   D22 =  1.0/R2aValue;
61
62   E11 = C1bValue;
63   E22 = C2bValue;
64
65 end
66
67 // Update variables at all time-steps
68 analog begin
69   // Equation yielding the first pole  / first state variable
70   V(X1): ddt(V(X1)) == (A11*V(X1) + A12*V(X2) + B11*V(U1) + B12*V(U2))/E11;
71
72   // Equation yielding the second pole
73   V(X2): ddt(V(X2)) == (A21*V(X1) + A22*V(X2) + B21*V(U1) + B22*V(U2))/E22;
74
75   // Equations yielding the first output
76   I(U1) <+ C11*V(X1);
77   I(U1) <+ C12*V(X2);
78   I(U1) <+ D11*V(U1);
79   I(U1) <+ D12*V(U2);
80
81   // Equations yielding the first output
82   I(U2) <+ C21*V(X1);
83   I(U2) <+ C22*V(X2);
84   I(U2) <+ D21*V(U1);
85   I(U2) <+ D22*V(U2);
```

Listing 8.7: Example of Verilog-AMS model simulated in C++ using numerical methods.

9

Application Of Methodology

In previous chapters we saw how proposed methodologies are validated on single components or on single circuits. In this chapter i will discuss the effectiveness and applications of proposed methodologies on complete systems.

9.1 Integration into a Virtual Platform (VP)

The primary goal of virtual platforms is development of software instead of designing hardware; simulation efficiency can achieved by increasing the level of abstraction in the hardware description. Unfortunately,The existence of analog components, however, creates new challenges for the development of virtual platforms. Because tools that are used to develop analog components generally produce descriptions based on algebraic and differential equations, that require equation resolution rather than event-driven simulation. As a result, simulation time becomes a crucial factor. Even worse, environments that represent analog hardware like Verilog-AMS Matlab, SystemC-AMS, and SPICE support co-simulation of digital and analog hardware, but at the cost of increasing simulation time . Therefore, to reduce simulation time and simplify the construction of virtual platforms, an abstraction methodology for analog models must be developed. Integrating analog components inside Virtual Platforms (VPs) requires modeling with a high-level description language Usually SystemC RTL/TLM, SystemC-AMS, C, C++. Finding these descriptions is hard, and most of the time for these descriptions of Piece-Wise Linear and non-linear (PWL) models are not provided. So a methodology for building and simulating VPs containing PWL models is requires. In this context this Section shows how the proposed abstraction methodology that has been discussed in previous chapter is effective on the simulation of a complete smart system shown in 9.1 . The C++ description of half-wave rectifier that is abstracted in previous chapter is embedded into a reference Virtual Platforms (VP) described in Figure 9.1. The rectifier is connected into the VP to the MEMS to provide the current DC voltage to guarantee the correct functionality of the MEMS. This VP is composed of an Advanced Microcontroller Bus Architecture (AMBA) bus that interconnects all the components of the VP internally. The components connected to the AMBA bus inside the VP are a com6502 microprocessor, SRAM, UART, a radio-frequency transceiver, and a Micro Electro Mechanical Systems (MEMS) accelerometer composed of Analog to Digital Converters (ADCs), Digital to Analog converters (DACs), and our half-wave rectifier to

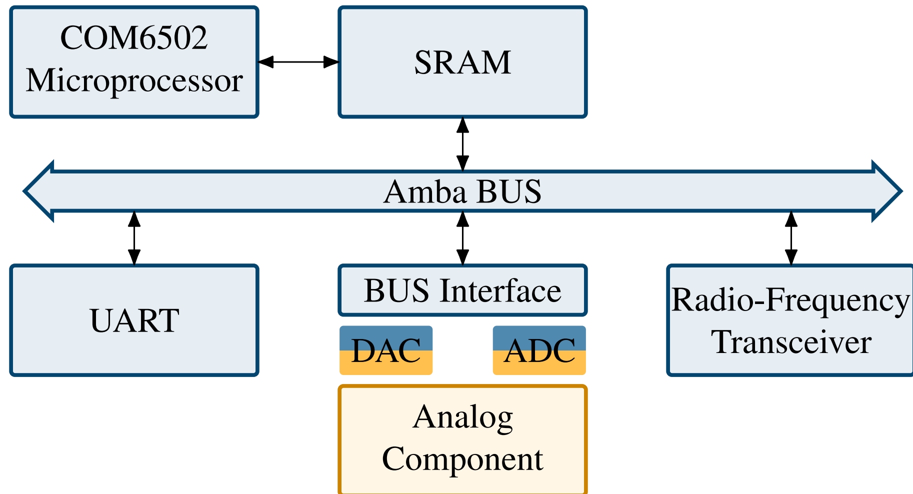


Fig. 9.1: Heterogeneous Platform.

provide the correct DC voltage to the MEMS results are shown in 9.1. The advantage of integrating several heterogeneous components within the same VP is that a validation of the functionality of an entire system can be performed before taking it to the production stage [103]. In this way, it is possible to obtain an efficient functional simulation using the C++ language. Moreover, it is possible to reduce simulation times and modify the various designs quickly if necessary to obtain a complete system in which the functionality is validated. Furthermore, when the system is complete, it is also possible to inject faults into the digital and analog parts to verify the robustness of the designs created to verify functional safety.

Table 9.1: Simulation results for the half-wave rectifier, the memristor, and the ideal relay.

Model	Half-wave rec.		Memristor		Ideal Relay	
Version	Time	Sp.	Time	Sp.	Time	Sp.
Heterogeneous	2.02s	-	0.57s	-	0.32s	-
Abstraction	0.76s	2.65x	0.005s	114x	0.002s	160x

9.2 Abstraction Of Faults

ANABASIS is a project collaboration with collaboration of Sydelity B.V., Kruisem, Belgium company. All the proposed work is applied on the different part of this project with respect to functional safety. Abstraction of faults can be done from this technique. The ANABASIS flow relies on the collection of AC matrices at the circuit ports and applies VF regression to them in order to build a set of LTI systems modelling the circuit around specific operating points. Interpolation in the responses is used to fill the gaps between the O.P. at which the LTI systems were extracted. This procedure removes any structural information related to circuit implementation. It only keeps sampled data describing the circuit behaviour.We can use this model for the abstraction of faults. the impact of faults can only be assessed by tracing the resulting numeric modifications on AC matrices and then on poles and residues. To trace the impact of Faults on AC Matrices We

can activate the fault in the circuit and simulate it as we do to extract data for the fault-free case in ANABASIS. This is a process to repeat for each fault-site. We can use sensitivity analysis (based on the adjoint circuit matrix) to assess the impact of all faults at once on the AC matrices. To do the sensitivity analysis of AC matrices : Insert zero current or voltage sources at each fault-site such that the fault-free behaviour is not affected, but the proper sensitivities can be computed. Insert fault models with "dormant parameters" (e.g. a short with infinite resistance or an open with zero series resistance). This work is in progress figure 9.2 and 9.3 presents the working flow of this work. Proposed faults in chapter 4 are injected. For the injection and manipulation proposed analog netlist manipulation tool in chapter 6 is used and proposed fault grouping technique in chapter 5 is used to categorize the faults to reduce the simulation time.

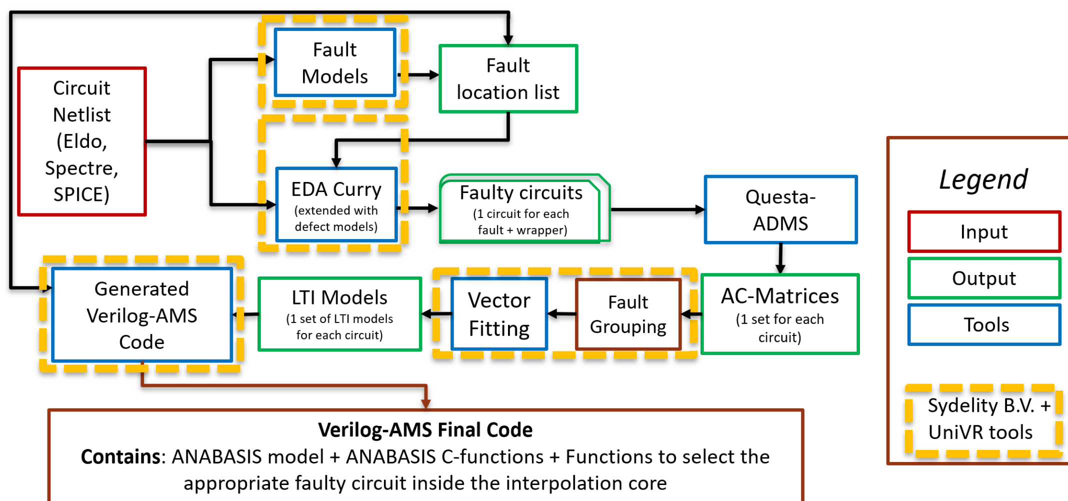


Fig. 9.2: Abstraction Of faults.

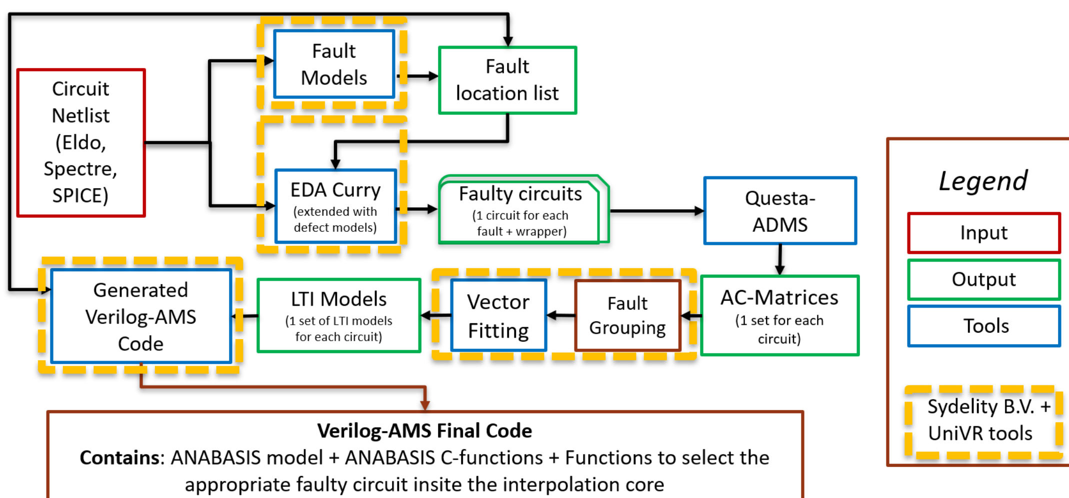


Fig. 9.3: Abstraction Of faults using fault grouping.

10

Conclusion and suggestions for future research

In the electronic design and testing field, functional safety is crucial in every physical context. The effective fault modeling, simulation solutions for analog circuits are lagging as compared to digital circuits. Digital circuits have efficient cost-effective solutions for structural testing in the presence of faults. Testing of digital circuits is different from Analog and Mixed-Signal (AMS) circuits because we also need to look at the parametric deviation of circuits and the presence of continuous signals. IEEE P2427 research group is working to provide a draft standard for assessing the functional safety in the analog domain bit it still on-going work. In this context, new defect modeling technique in chapter 4, fault grouping for simulation in chapter 5, fault injection and manipulation in chapter 6, new fault templates in chapter 7and abstraction methodologies in chapter 8 are proposed in this thesis to contribute in the research field. These methodologies are developed by using following the IEEE P2427 draft standard. Proposed methodologies are tested on proper IEEE analog benchmark circuits. With respect to testing on large circuits these methodologies are tested different parts of industrial project named ANABASIS and on complete Virtual Platforms (VP) of smart systems.

In future work, the proposed fault templates will be extended with a mechanism to compute fault-injection "weights" and pass them to the simulation scheduler. Abstraction methodology from transistor level to Verilog-AMS can be use for the abstraction of faults. The next step is towards the abstraction of faults and this work is in progress. The flow of methodology relies on the collection of AC matrices at the circuit ports and applies VF regression to them in order to build a set of LTI systems modelling the circuit around specific operating points. Interpolation in the responses is used to fill the gaps between the O.P. at which the LTI systems were extracted. This procedure removes any structural information related to circuit implementation. It only keeps sampled data describing the circuit behaviour. The impact of faults can only be assessed by tracing the resulting numeric modifications on AC matrices and then on poles and residues.

Summary of the proposed innovative contributions

10.1 Thesis related publications

Most of the work explained in thesis is related to these papers:

10.1.1 Fault Modeling and Analysis

Published Papers

- Sadia Azam, Nicola Dall'Ora, Enrico Fraccaroli, Andre Alberts, Renaud Gillon & Franco Fummi. *Investigation on realistic stuck-on/off defects to complement IEEE p2427 draft standard*.
  In 23rd International Symposium on Quality Electronic Design (ISQED), IEEE, pages 51-57, 2022
- Nicola Dall'Ora, Sadia Azam, Enrico Fraccaroli, André Alberts, & Franco Fummi. *Predictive fault grouping based on faulty ac matrices*.
  In 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), IEEE, pages 11–16, 2021

Submitted Papers

- Nicola Dall'Ora, Sadia Azam, Enrico Fraccaroli, Renaud Gillon & Franco Fummi , *Verilog-A Implementation of Generic Fault Templates*.
  In 60th Design and Automation conference 2023.

10.1.2 Fault Injection

Published Papers

- Nicola Dall'Ora, Sadia Azam, Enrico Fraccaroli, André Alberts and Franco Fummi. *A common manipulation framework for transistor-level languages*.
  In Forum on specification Design Languages (FDL), IEEE, pages 01–07, 2021

Submitted Papers

- Sadia Azam, Nicola Dall'Ora,Enrico Fraccaroli, Renaud Gillon & Franco Fummi , *Analog Fault Injection and Simulation Techniques: A Systematic Literature Review*, IEEE TCAD (Journal)

Under Submission

- A Language Independent Framework to Manipulate Transistor-level Netlists, IEEE TCAD (Journal)

10.1.3 Model Abstraction

Published Papers

- Sadia Azam, Nicola Dall'Ora, Enrico Fraccaroli, & Franco Fummi.*Functional level abstraction and simulation of verilog-ams piecewise linear models*.
  In 23rd Inter-national Symposium on Quality Electronic Design (ISQED), IEEE ,pages 39–44, 2022

Under Submission

- A Complete Framework Moving from Transistor-level to Behavioral Models, IEEE TCAD(Journal)

10.1.4 Simulation

Published Papers

- Sadia Azam, Nicola Dall'Ora, Enrico Fraccaroli, & Franco Fummi.*Functional level abstraction and simulation of verilog-ams piecewise linear models*.
  In 23rd Inter-national Symposium on Quality Electronic Design (ISQED), IEEE ,pages 39–44, 2022

References

1. S. Sunter and P. Sarson, “A/MS benchmark circuits for comparing fault simulation, DFT, and test generation methods,” in *IEEE Int. Test Conference (ITC)*. IEEE, Oct. 2017.
2. “System Level Modeling.” [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/System-Level_Modeling_in_SystemC
3. F. Su and P. Goteti, “Improving analog functional safety using data-driven anomaly detection,” in *2018 IEEE International Test Conference (ITC)*, 2018, pp. 1–10.
4. *ISO 26262-2 – Road vehicles – Functional safety*, ISO, 2018.
5. *IEEE-SA Standards Board P2427/D0.13 Draft Standard for Analog Defect Modeling and Coverage*. IEEE, 2019. [Online]. Available: https://standards.ieee.org/project/2427.html
6. 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*. IEEE Conference Publications, May 2013, pp. 565–570.
7. R. Pratap, V. Agarwal, and R. K. Singh, “Review of various available spice simulators,” in *2014 International Conference on Power, Control and Embedded Systems (ICPCES)*, 2014, pp. 1–6.
8. “Spice users guide,” https://irsa.ipac.caltech.edu/data/SPITZER/docs/data analysistools/tools/spice/spiceusersguide/SPICE_Users_Guide.pdf, v.2.3.1.
9. K. S. Kundert and P. Gray, *The Designer’s Guide to Spice and Spectre*. USA: Kluwer Academic Publishers, 1995.
10. L.-R. Zheng, M. Shen, X. Duo, and H. Tenhunen, “Embedded smart systems for intelligent paper and packaging,” in *Proceedings Electronic Components and Technology, 2005. ECTC ’05.*, 2005, pp. 1776–1782 Vol. 2.
11. “Cadence virtual system platform,” https://www.cadence.com/ko_KR/home /tools/system-design-and-verification/software-driven-verification/virtual-system-platform.html, 2021.
12. A. Nardi, S. Camdzic, A. Armato, and F. Lertora, “Design-For-Safety For Automotive IC Design: Challenges And Opportunities,” in *2019 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, Apr. 2019.
13. S. Sunter, “Efficient Analog Defect Simulation,” in *2019 IEEE International Test Conference (ITC)*. IEEE, Nov. 2019.
14. S. Sunter and P. Sarson, “A/MS benchmark circuits for comparing fault simulation, DFT, and test generation methods,” in *2017 IEEE International Test Conference (ITC)*, 2017, pp. 1–7.
15. I. S. Esqueda and H. J. Barnaby, “A defect-based compact modeling approach for the reliability of CMOS devices and integrated circuits,” *Solid-State Electronics*, vol. 91, pp. 81–86, Jan. 2014.

16. Y. Zhou, H. Liu, S. Mu, Z. Chen, and B. Wang, "Short-circuit failure model of sic mosfet including the interface trapped charges," *IEEE Trans. Emerg. Sel. Topics Power Electron.*, vol. 8, no. 1, pp. 90–98, 2020.
17. S. Rashkeev, C. Cirba, D. Fleetwood, R. Schrimpf, S. Witczak, A. Michez, and S. Pantelides, "Physical model for enhanced interface-trap formation at low dose rates," *IEEE Trans. Nucl. Sci.*, vol. 49, no. 6, pp. 2650–2655, 2002.
18. M. Graphics, *Defectsim*. [Online]. Available: https://www.mentor.com/products/ic_nanometer_design/analog-mixed-signal-verification/eldo-platform.
19. S. Sanyal, S. P. P. K. Garapati, A. Patra, P. Dasgupta, and M. Bhattacharya, "Fault classification and coverage of analog circuits using DC operating point and frequency response analysis," in *Proc. of the 2019 on Great Lakes Symposium on VLSI*. ACM, May 2019, pp. 123—-128.
20. F. Pecheux, C. Lallement, and A. Vachoux, "Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 204–225, 2005.
21. "Spectre circuit simulator reference manual," http://web.engr.uky.edu/ elias/tutorials/Spectre/spectre_refManual.pdf, 2020.
22. H. Mantooth, L. Ren, X. Huang, Y. Feng, and W. Zheng, "A survey of bottom-up behavioral modeling methods for analog circuits," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, vol. 3, 2003, pp. III–III.
23. T. Bonnerud, B. Hernes, and T. Ytterdal, "A mixed-signal, functional level simulation framework based on systemc for system-on-a-chip applications," in *Proceedings of the IEEE 2001 Custom Integrated Circuits Conference (Cat. No.01CH37169)*, 2001, pp. 541–544.
24. S.-N. Ahmadian and S.-G. Miremadi, "Fault injection in mixed-signal environment using behavioral fault modeling in Verilog-A," in *2010 IEEE International Behavioral Modeling and Simulation Workshop*, 2010, pp. 69–74.
25. Y. Kiliç and M. Zwoliński, "Behavioral fault modeling and simulation using VHDL-AMS to speed-up analog fault simulation," *Analog Integrated Circuits and Signal Processing*, vol. 39, no. 2, pp. 177–190, May 2004.
26. E. Yilmaz, A. Meixner, and S. Ozev, "An industrial case study of analog fault modeling," in *29th VLSI Test Symposium*, 2011, pp. 178–183.
27. 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)*, 2016, pp. 1–9.
28. S. Sunter, "Analog fault simulation - a hot topic!" in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–5.
29. V. Gutiérrez Gil, A. J. Gines Arteaga, and G. Léger, "Assessing AMS-RF Test Quality by Defect Simulation," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 55–63, 2019.
30. M. Graphics, *Eldo Platform*. [Online]. Available: https://www.mentor.com/products/ic_nanometer_design/analog-mixed-signal-verification/eldo-platform.
31. A. Technologies, "Netlist translator for spice and spectre," https://pages.jh.edu/aandreo1/495/Archives/Bibliography/CAD/SPICE_Models.Equivalency.pdf, 2002.
32. D. Batas and H. Fiedler, "A python interface for spice-based simulations," in *ICSES 2010 International Conference on Signals and Electronic Circuits*, 2010, pp. 161–164.
33. O. Karaca *et al.*, "Fault grouping for fault injection based simulation of AMS circuits in the context of functional safety," in *Int. Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*. IEEE, Jun. 2016.
34. G. Leger and A. Gines, "Likelihood-sampling adaptive fault simulation," in *2017 International Mixed Signals Testing Workshop (IMSTW)*, 2017, pp. 1–6.

35. S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, "Practical random sampling of potential defects for analog fault simulation," in *2014 International Test Conference*, 2014, pp. 1–10.
36. J. Hou and A. Chatterjee, "Analog transient concurrent fault simulation with dynamic fault grouping," in *Proc. Int. Conference on Computer Design*. IEEE Comput. Soc, Sep 2000, pp. 35–41.
37. N. Guerreiro, M. Santos, and P. Teixeira, "Fault list compression for efficient analogue and mixed-signal production test preparation," in *Design of Circuits and Integrated Systems*. IEEE, Nov 2014, pp. 1–6.
38. Y.-H. Chang, "Frequency-domain grouping robust fault diagnosis for analog circuits with uncertainties," *Int. Journal of Circuit Theory and Applications*, vol. 30, no. 1, pp. 65–86, Jan 2002.
39. M. W. Tian and C.-J. R. Shi, "Rapid frequency-domain analog fault simulation under parameter tolerances," in *Proceedings of the 34th annual conference on Design automation conference - DAC '97*. ACM Press, Aug 1997, pp. 275–280.
40. S. Ozev and A. Orailoglu, "An integrated tool for analog test generation and fault simulation," in *Proceedings International Symposium on Quality Electronic Design*, 2002, pp. 267–272.
41. V. Zivkovic and A. Schaldenbrand, "Requirements, for Industrial Analog Faulf-Simulator," in *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2019, pp. 61–64.
42. N. Dall'Ora, t. **Azam**, E. Fraccaroli, A. Alberts, and F. Fummi, "A common manipulation framework for transistor-level languages," in *2021 Forum on specification Design Languages (FDL)*, 2021, pp. 01–07.
43. T. Gao, Y. Sun, and G. Zhao, "Analog Circuit Fault Simulation Based on Saber," in *2010 International Conference on Computational and Information Sciences*, 2010, pp. 388–391.
44. R. Bhattacharya, S. H. M. Ragamai, and S. Kumar, "SFG Based Fault Simulation of Linear Analog Circuits Using Fault Classification and Sensitivity Analysis," in *VLSI Design and Test*, B. K. Kaushik, S. Dasgupta, and V. Singh, Eds. Singapore: Springer Singapore, 2017, pp. 179–190.
45. W. C. Tam and R. D. Blanton, "SLIDER: Simulation of Layout-Injected Defects for Electrical Responses," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 918–929, 2012.
46. V. Huard, "Two independent components modeling for negative bias temperature instability," in *2010 IEEE International Reliability Physics Symposium*, 2010, pp. 33–42.
47. D. De Jonghe and G. Gielen, "Characterization of analog circuits using transfer function trajectories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1796–1804, 2012.
48. T. Bradde, S. Grivet-Talocia, P. Toledo, A. V. Proskurnikov, A. Zanco, G. C. Calafiore, and P. Crovetti, "Fast simulation of analog circuit blocks under nonstationary operating conditions," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 9, pp. 1355–1368, 2021.
49. K. Jung, W. Eisenstadt, and R. Fox, "Spice-based mixed-mode s-parameter calculations for four-port and three-port circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 5, pp. 909–913, 2006.
50. C. Grimm and M. Rathmair, "Dealing with uncertainties in analog/mixed-signal systems," in *Proc. of the 54th Annual Design Automation Conference 2017*. ACM, Jun. 2017.
51. T. Grasser, B. Kaczer, W. Goes, T. Aichinger, P. Hehenberger, and M. Nelhiebel, "A two-stage model for negative bias temperature instability," in *2009 IEEE International Reliability Physics Symposium*, 2009, pp. 33–44.
52. N. Dall'Ora, S. Azam, E. Fraccaroli, A. Alberts, and F. Fummi, "Predictive fault grouping based on faulty ac matrices," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2021, pp. 11–16.

53. C. Tenorio de Carvalho and L. de Menezes, “Extraction of s-parameter using the three-dimensional transmission-line matrix (tlm) method,” in *Proc. of the 2003 SBMO/IEEE MTT-S Int. Microwave and Optoelectronics Conference - IMOC 2003. (Cat. No.03TH8678)*, vol. 2, 2003, pp. 967–969.
54. S. Tian, C. Yang, F. Chen, and Z. Liu, “Circle equation-based fault modeling method for linear analog circuits,” *IEEE Trans. Instrum. Meas.*, vol. 63, no. 9, pp. 2145–2159, Sep. 2014.
55. A. Al-Sharadqah and N. Chernov, “Error analysis for circle fitting algorithms,” *Electronic Journal of Statistics*, vol. 3, no. 0, pp. 886–911, Jul 2009.
56. S. Sunter, “Efficient analog defect simulation,” in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–10.
57. K. Jung, W. Eisenstadt, and R. Fox, “SPICE-based mixed-mode s-parameter calculations for four-port and three-port circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 5, pp. 909–913, May 2006.
58. S.-S. Lu, T.-W. Chen, H.-C. Chen, and C.-C. Meng, “A novel interpretation of transistor s-parameters by poles and zeros for RF IC circuit design,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 49, no. 2, pp. 406–409, 2001.
59. Y.-J. Kim, O.-K. Kwon, and C.-H. Lee, “Equivalent circuit extraction from the measured s-parameters of electronic packages,” in *ICVC '99. 6th International Conference on VLSI and CAD (Cat. No.99EX361).* IEEE, Aug 1999, pp. 415–418.
60. J.-C. G. Santos and R. Torres-Torres, “Assessing the accuracy of the open, short and open-short de-embedding methods for on-chip transmission line s-parameters measurements,” in *2017 International Caribbean Conference on Devices, Circuits and Systems (ICCDCS).* IEEE, Jun 2017, pp. 57–60.
61. “Edacurry – unified mixed-signal netlist parser framework,” https://github.com/sydelity-net/EDACurry, 2021.
62. “Pybind11 - seamless operability between c++11 and python,” https://pybind11.readthedocs.io/en/stable/index.html, 2021.
63. S. Sunter, K. Jurga, P. Dingenen, and R. Vanhooren, “Practical random sampling of potential defects for analog fault simulation,” in *2014 International Test Conference*, 2014, pp. 1–10.
64. J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, “Graphviz and dynagraph – static and dynamic graph drawing tools,” in *GRAPH DRAWING SOFTWARE.* Springer-Verlag, 2003, pp. 127–148.
65. L. Xia, M. U. Farooq, I. M. Bell, F. A. Hussin, and A. S. Malik, “Survey and Evaluation of Automated Model Generation Techniques for High Level Modeling and High Level Fault Modeling,” *Journal of Electronic Testing*, vol. 29, no. 6, pp. 861–877, Aug. 2013.
66. M. Zwolinski and A. Brown, “Behavioural modelling of analogue faults in VHDL-AMS - a case study,” in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, vol. 5, 2004, pp. V–V.
67. Y. Kiliç and M. Zwoliński, “Behavioral fault modeling and simulation using VHDL-AMS to speed-up analog fault simulation,” *Analog Integrated Circuits and Signal Processing*, vol. 39, no. 2, pp. 177–190, may 2004.
68. P. Wilson, Y. Kilic, J. Ross, M. Zwolinski, and A. Brown, “Behavioural modelling of operational amplifier faults using analogue hardware description languages,” in *Proceedings of the Fifth IEEE International Workshop on Behavioral Modeling and Simulation. BMAS 2001 (Cat No.01TH8601).* IEEE, 2001, pp. 106–112.
69. ——, “Behavioural modelling of operational amplifier faults using VHDL-AMS,” in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition.* IEEE Comput. Soc, 2002, pp. 1133–.
70. W. Zheng, Y. Feng, X. Huang, and H. Mantooth, “Ascend: automatic bottom-up behavioral modeling tool for analog circuits,” in *2005 IEEE International Symposium on Circuits and Systems.* IEEE, 2005, pp. 5186–5189 Vol. 5.

71. S. Puthiyottil and E. Sureshkumar, “Speed Enhanced Mixed Signal Design-for-Test Using Hybrid Fault Based Testing Algorithms,” in *2010 Second International Conference on Computer Modeling and Simulation*, vol. 4. IEEE, Jan. 2010, pp. 270–274.
72. J. Parky, S. Madhavapeddiz, A. Paglieri, C. Barrz, and J. A. Abrahamy, “Defect-based analog fault coverage analysis using mixed-mode fault simulation,” in *2009 IEEE 15th International Mixed-Signals, Sensors, and Systems Test Workshop*. IEEE, Jun. 2009, pp. 1–6.
73. L. Fang, G. Gronthoud, and H. Kerkhoff, “Reducing analogue fault-simulation time by using ifigh-level modelling in dotss for an industrial design,” in *IEEE European Test Workshop, 2001.* IEEE, 2001, pp. 61–67.
74. X. Li, X. Zeng, D. Zhou, and X. Ling, “Behavioral modeling of analog circuits by wavelet collocation method,” in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*. IEEE, 2001, pp. 65–69.
75. D. Shaw, D. Al-Khalili, and C. Rozon, “Automatic generation of defect injectable VHDL fault models for ASIC standard cell libraries,” *Integration*, vol. 39, no. 4, pp. 382–406, Jul. 2006.
76. N. Zain Ali, M. Zwolinski, and A. Ahmadi, “Delay fault modelling/simulation using VHDL-AMS in multi-Vdd systems,” in *2008 26th International Conference on Microelectronics*, 2008, pp. 413–416.
77. E. Romero, G. Peretti, and C. Marqués, “An operational amplifier model for evaluating test strategies at behavioural level,” *Microelectronics Journal*, vol. 38, no. 10-11, pp. 1082–1094, Oct. 2007.
78. B. Straube, W. Vermeiren, and V. Spenke, “Multi-level hierarchical analogue fault simulation,” *Microelectronics Journal*, vol. 33, no. 10, pp. 815–821, Oct. 2002.
79. S. Sunter *et al.*, “Using mixed-signal defect simulation to close the loop between design and test,” *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2313–2322, Dec. 2016.
80. A. Tarraf and L. Hedrich, “Behavioral modeling of transistor-level circuits using automatic abstraction to hybrid automata,” in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1451–1456.
81. M. Sano and L. Sun, “Identification of hammerstein-wiener system with application to compensation for nonlinear distortion,” in *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, vol. 3, 2002, pp. 1521–1526 vol.3.
82. M. Rewienski and J. White, “A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 2, pp. 155–170, 2003.
83. J. Pedro and S. Maas, “A comparative overview of microwave and wireless power-amplifier behavioral modeling approaches,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 4, pp. 1150–1163, 2005.
84. D. Root, J. Verspecht, D. Sharrit, J. Wood, and A. Cognata, “Broad-band poly-harmonic distortion (phd) behavioral models from fast automated simulations and large-signal vectorial network measurements,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 53, no. 11, pp. 3656–3664, 2005.
85. S. Nagaraj, D. Seshachalam, and S. Hucharaddi, “Model order reduction of nonlinear circuit using proper orthogonal decomposition and nonlinear autoregressive with exogenous input (narx) neural network,” in *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2018, pp. 1–4.
86. Z. Naghibi, S. A. Sadrossadat, and S. Safari, “Time-domain modeling of nonlinear circuits using deep recurrent neural network technique,” *AEU - International Journal of Electronics and Communications*, vol. 100, pp. 66–74, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1434841118318235

87. T. Bradde, S. Grivet-Talocia, P. Toledo, A. V. Proskurnikov, A. Zanco, G. C. Calafiore, and P. Crovetti, "Fast simulation of analog circuit blocks under nonstationary operating conditions," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 9, pp. 1355–1368, 2021.

88. D. De Jonghe and G. Gielen, "Characterization of analog circuits using transfer function trajectories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1796–1804, 2012.

89. M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.

90. A. Tarraf and L. Hedrich, "Behavioral modeling of transistor-level circuits using automatic abstraction to hybrid automata," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2019.

91. E. Tlelo-Cuautle and M. Duarte-Villaseñor, "Designing chua's circuit from the behavioral to the transistor level of abstraction," *Applied Mathematics and Computation*, vol. 184, no. 2, pp. 715–720, Jan. 2007.

92. S. Dam and P. Mandal, "Modeling and design of CMOS analog circuits through hierarchical abstraction," *Integration*, vol. 46, no. 4, pp. 449–462, Sep. 2013.

93. J. L. V. S. de la Vega and E. Tlelo-Cuautle, "Simulation of piecewise-linear one-dimensional chaotic maps by verilog-a," *IETE Technical Review*, vol. 32, no. 4, pp. 304–310, Mar. 2015.

94. Y. Zhang, S. Sankaranarayanan, and F. Somenzi, "Piecewise linear modeling of nonlinear devices for formal verification of analog circuits," in *2012 Formal Methods in Computer-Aided Design (FMCAD)*, 2012, pp. 196–203.

95. J. Goncalves, A. Megretski, and M. Dahleh, "Global analysis of piecewise linear systems using impact maps and surface lyapunov functions," *IEEE Transactions on Automatic Control*, vol. 48, no. 12, pp. 2089–2106, 2003.

96. "Introduction to the ginac framework for symbolic computation within the c++ programming language," 2002. [Online]. Available: https://www.ginac.de/csSC-0004015.pdf

97. K. Kundert and O. Zinke, *The designer's guide to Verilog-AMS*. Springer Science & Business Media, 2006.

98. Z. Biolek, D. Biolek, and B. V, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, 06 2009.

99. Ahmed ElTahan , "Curve Linearization," 2022. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/57112-curve-linearization

100. G. K. Lowe and M. A. Zohdy, "Modeling nonlinear systems using multiple piecewise linear equations," vol. 15, no. 4, pp. 451–458, Oct. 2010. [Online]. Available: https://doi.org/10.15388/na.15.4.14317

101. N. Bombieri, G. Di Guglielmo, F. Michele, F. Fummi, G. Pravadelli, F. Stefanni, and V. Alessandro, "Hifsuite: Tools for hdl code conversion and manipulation," *EURASIP Journal on Embedded Systems*, vol. 2010, 01 2010.

102. "MATLAB Engine." [Online]. Available: https://www.mathworks.com/help/matlab/cpp-engine-api.html

103. M. Monton, J. Engblom, and M. Burton, "Checkpointing for virtual platforms and systemc-tlm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 133–141, 2013.

A

Appendix A

```
1  module vdba(in, out);
2     input in ;
3     output out ;
4     electrical in, out ;
5     parameter real vin_low =  -2.0 ;
6     parameter real vin_high =  2.0 ;
7     parameter real gain = 1 from (0:inf);
8
9     analog begin
10
11    if (V(in) >= vin_high) begin
12
13       V(out) <+ gain*(V(in) - vin_high) ;
14    end
15
16    else if (V(in) <= vin_low) begin
17
18       V(out) <+ gain*(V(in) - vin_low) ;
19    end
20
21    else begin
22       $strobe("Third if statement");
23
24       V(out) <+ 0 ;
25
26       end
27    end
28
29 endmodule
```

Listing A.1: Volatge deadband amplifier.

```

`include "disciplines.vams"
`include "constants.vams"

`define X_BORDER_BUMP 10e-18

`define attr(txt) (*txt*)

module memristor1 (p,n);

    inout p,n;
    electrical p,n;

    parameter real Roff=16000 from (0:inf) `attr(info="Roff" type="instance");
    parameter real Ron=100 from (0:inf) `attr(info="Ron" type="instance");
    parameter real Rinit=11000 from (0:inf) `attr(info="Rinit" type="instance");
    parameter real D=10n from (0:inf) `attr(info="D" type="instance");
    parameter real uv=10e-15 from (0:inf) `attr(info="uv" type="instance");
    parameter real p_coeff=1.0 from (0:inf) `attr(info="p_coeff" type="instance");

    // local variables that should persist over time steps
    real w_last;
    real time_last;

    // local variables that hold temporary values
    real G;
    real window_function;
    real w;
    real dw;
    real R;
    real direction;
    real current;
    real time;
    real time_delta;
   analog
     begin

        @(initial_instance)
        begin
          w_last = ((Roff - Rinit) / (Roff - Ron)) * D;
          time_last = 0;
        end

        // calculate conductance
        G = 1 / (Ron * w_last / D + Roff * (1 - w_last / D));
        current = G * V(p,n);


        direction = 0;
        if (current > 0) begin
          if(w_last<=0) begin
              direction=1;
          end
        end
        else begin
          if(w_last>=D) begin
                direction=-1;
          end
        end

        time = $realtime;
        time_delta = time - time_last;
        window_function = (1.0 - (pow(2 * w_last / D - 1, 2 * p_coeff)));
        dw = uv * Ron / D * current * window_function * time_delta;
        w = w_last + dw + direction * `X_BORDER_BUMP;

        if(w >= D) begin
          w = D;
        end
        if(w <= 0) begin
          w = 0;
        end

        // calculate conductance
        G = 1 / (Ron * w / D + Roff * (1 - w / D));

        // set the current
        I(p,n) <+ -1.0*G*V(p,n);

        // persist variables
        w_last = w;
        time_last = time;

     end
endmodule
```

Listing A.2: memristor.

B

Appendix B

```
1
2  // Macro for the underlying model.
3  `ifndef ORIGINAL_MOSFET_MODEL
4      `define ORIGINAL_MOSFET_MODEL `ORIGINAL_MOSFET_MODEL
5  `endif
6  // Macro for the wrapper name.
7  `ifndef TARGET_MOSFET_MODEL
8      `define TARGET_MOSFET_MODEL anabasis_`ORIGINAL_MOSFET_MODEL
9  `endif
10 // Fault injection wrappers for MOSFET components.
11 module `TARGET_MOSFET_MODEL (D, G, S, B);
12   // Interface nodes.
13   inout D, G, S, B;
14   electrical D, G, S, B;
15   // Geometrical instance parameters.
16   parameter real l   = 0.35E-6;
17   parameter real w   = 10.0E-6;
18   parameter real ad  = 0.0;
19   parameter real as  = 0.0;
20   parameter real pd  = 0.0;
21   parameter real ps  = 0.0;
22   parameter real nrd = 0.0;
23   parameter real nrs = 0.0;
24   // Failure mode parameters.
25   parameter integer failmode = 0;
26   parameter real ropen = 1.0E9;
27   parameter real rshort = 0.1;
28   // Use genvar to instantiate the right model.
29   genvar mode;
30   for (mode = failmode; mode <= failmode; mode = mode + 1) begin
31     case (mode)
32           ( 1 ): begin
33                 // DRAIN OPEN
34                 // ----------
35                 // Internal nodes
36                 electrical DD;
37                 // Instanciating the fault-free instance
38                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
39                  core(DD, G, S, B) ;
40                 // Drain open
41                 resistor #(.r(ropen)) defect (D, DD) ;
42             end
43             ( 2 ): begin
44                 // GATE OPEN
45                 // ---------
46                 // Internal nodes
47                 electrical GG;
48                 // Instanciating the fault-free instance
49                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
50                  core(D, GG, S, B) ;
51                 // Gate open
52                 resistor #(.r(ropen)) defect (G, GG) ;
53             end
54             ( 3 ): begin
55                 // SOURCE OPEN
56                 // -----------
57                 // Internal nodes
58                 electrical SS;
59                 // Instanciating the fault-free instance
60                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
61                  core(D, G, SS, B) ;
62                 // Gate open
63                 resistor #(.r(ropen)) defect (S, SS) ;
64             end
65             ( 4 ): begin
66                 // BULK OPEN
67                 // ----------
68                 // Internal nodes
69                 electrical BB;
70                 // Instanciating the fault-free instance
71                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
72                  core(D, G, S, BB) ;
73                 // Gate open
74                 resistor #(.r(ropen)) defect (B, BB) ;
75             end
76             ( 5 ): begin
77                 // GATE-DRAIN SHORT
78                 // ----------------
79                 // Instanciating the fault-free instance
80                 `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
81                  core(D, G, S, B) ;
82
83                 // Gate-Drain short
84                 resistor #(.r(rshort)) defect (G, D) ;
85             end
```

Listing B.1: Templates for MOSFET fault models Part 1.

```
1               ( 6 ): begin
2                   // GATE-SOURCE SHORT
3                   // ----------------
4                   // Instanciating the fault-free instance
5                   `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
6                    core(D, G, S, B) ;
7
8                   // Gate-Source short
9                   resistor #(.r(rshort)) defect (G, S) ;
10              end
11
12              ( 7 ): begin
13                  // GATE-BODY SHORT
14                  // ----------------
15                  // Instanciating the fault-free instance
16                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
17                    core(D, G, S, B) ;
18
19                  // Gate-Source short
20                  resistor #(.r(rshort)) defect (G, B) ;
21              end
22
23              ( 8 ): begin
24                  // BODY-DRAIN SHORT
25                  // ----------------
26                  // Instanciating the fault-free instance
27                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
28                   core(D, G, S, B) ;
29
30                  // Body-Drain short
31                  resistor #(.r(rshort)) defect (B, D) ;
32              end
33
34              ( 9 ): begin
35                  // BODY-SOURCE SHORT
36                  // ----------------
37                  // Instanciating the fault-free instance
38                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
39                    core(D, G, S, B) ;
40
41                  // Body-Drain short
42                  resistor #(.r(rshort)) defect (B, S) ;
43              end
44
45              ( 10 ): begin
46                  // DRAIN-SOURCE SHORT
47                  // ----------------
48                  // Instanciating the fault-free instance
49                  `ORIGINAL_MOSFET_MODEL # (.l(l),.w(w),.ad(ad),.as(as),.pd(pd),.ps(ps),.nrd(nrd),.nrs(nrs))
50                   core(D, G, S, B) ;
51
52                  // Body-Drain short
53                  resistor #(.r(rshort)) defect (D, S) ;
54
55          end
56        endcase
57      end
58  endmodule
```

Listing B.2: Templates for MOSFET fault models Part 2.

C

Appendix C

```
1
2
3 `include "disciplines.vams"
4 `include "constants.vams"
5 `timescale 1ms / 1ms
6
7 module motor(in1, in2, out);
8   inout in1, in2;
9   electrical in1, in2;
10  rotational_omega out;
11  output out;
12  //ground in2;
13
14  branch (in1, in2) input_voltage;
15
16  parameter real K1 = 0.052,   K2 = 0.052;
17  parameter real Ra = 1.3 ;
18  parameter real La = 1.8e-03;
19  parameter real J = 2.02e-5;
20  parameter real B = 0.01 ;
21  parameter real l = 0.24 ;
22  parameter real m = 0.1 ;
23  parameter real g = 1;
24
25  analog begin
26
27      Omega(out) <+ (m * pow(l,2) + J*ddt(Omega(out)))/B + (K2*I(input_voltage))/B - m*g*l*sin(idt(Omega(out)))/B;
28
29      I(input_voltage): ddt(I(input_voltage))  == ((-Ra/La) * I(input_voltage))-(K1/La)*Omega(out)-((1/La)*V(in1,in2)
        ); // or *I(in1,out)
30  end
31 endmodule
```

Listing C.1: motor driven pendulum .