Université Claude Bernard Lyon 1

THÈSE DE DOCTORAT DE L'UNIVERSITÉ CLAUDE BERNARD LYON 1

École Doctorale N° 512
InfoMaths

Discipline : Informatique

Soutenue publiquement le 21/11/2025, par :

Oussama Oulkaid

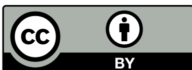
Modèles formels des circuits intégrés pour la vérification électrique au niveau transistor

Devant le jury composé de :

Emmanuelle Encrenaz	Professeure	Sorbonne Université	Rapporteure
Katell Morin-Allory	Professeure	Grenoble INP – Phelma	Rapporteure
Arnaud Virazel	Professeur	Université de Montpellier	Examinateur
Lars Hedrich	Professeur	Goethe-Universität	Examinateur
Xavier Urbain	Professeur	Université Claude Bernard Lyon 1	Président du jury
Matthieu Moy	Maître de Conférences	Université Claude Bernard Lyon 1	Directeur de thèse
Pascal Raymond	Chargé de Recherche	CNRS	Co-encadrant de thèse
Bruno Ferres	Maître de Conférences	Université Grenoble Alpes	Co-encadrant de thèse
Mehdi Khosravian	Ingénieur de Recherche	Aniah	Co-encadrant de thèse

Formal Models of Integrated Circuits for Transistor Level Electrical Verification

Oussama Oulkaid



Prepared by Oussama Oulkaid in 2022–2025 at the Laboratoire de l'Informatique du Parallélisme (LIP—Lyon, France), VERIMAG Laboratory and ANIAH (Grenoble, France). This work is funded by the Convention Industrielle de Formation par la Recherche (CIFRE) program.

Colophon
This document was typeset using LaTeX, BibTeX, and TeXindy. It is built using the LaTeX memoir document class. The distribution used for compilation of the main document is LuaLaTeX. Figures are made with PGF/Ti*k*Z and compiled with either pdfLaTeX, LuaLaTeX, or XeLaTeX. In particular, circuit schematics were produced thanks to the CircuiTi*k*Z package. Plots were essentially processed with the R language and produced as TeX files using the R tikzDevice library.

The lowest level? Well, not really, for I am not going to talk about elementary particles—but it is the lowest level which we wish to think about.

Douglas R. Hofstadter
Gödel, Escher, Bach: an Eternal Golden Braid

Mots de remerciement

J'ai eu énormément de plaisir à rédiger ce manuscrit. Le plaisir que j'ai eu en travaillant sur ce qui deviendra le contenu de ses chapitres n'est pas moindre. À toutes les personnes ayant rendu ces trois années de thèse si agréables et riches d'expériences, je dis simplement

Merci !

À Matthieu Moy, mon directeur de thèse, pour l'encadrement de qualité dont j'ai pu bénéficier tout au long de cette thèse. Je suis très reconnaissant d'avoir eu un encadrant comme toi, si compétent, présent, et bienveillant. J'ai beaucoup appris de toi. Merci pour toutes les idées que tu as contribué à la thèse, pour nos échanges très fréquents et vraiment qualitatifs même en visio, et pour les nombreuses relectures de ce manuscrit ! Tes mots d'encouragement m'ont toujours donné la force pour avancer.

À Pascal Raymond, pour m'avoir donné envie de faire la recherche depuis que j'étais encore étudiant à l'Ensimag, pour m'avoir d'abord encadré dans mon stage à Verimag ayant précédé cette thèse, et d'avoir continué à m'encadrer pendant la thèse. On a tant discuté de sujets passionnants devant un tableau, des BDDs aux solveurs SAT. Merci également pour les relectures, pour ton soutien, et pour ton encouragement.

À Bruno Ferres, pour ton soutien permanent, pour ton implication forte dans mon encadrement, et pour la bonne humeure au quotidien. Merci pour tes promptes relectures (!), et pour tes retours toujours constructifs. Merci encore pour les coups de main pour débogguer du code OCaml. Tu m'a aussi énormément aidé en matière d'enseignement, et tu as toujours su m'orienter vers les bonnes personnes à contacter. J'en suis très reconnaissant !

À Mehdi Khosravian, pour l'encadrement industriel de la thèse, et pour m'avoir initié au problème de la vérification électrique. Merci également pour ton aide à la préparation des benchmarks industriels utilisés dans la thèse. Merci pour tous les bons moments partagés en entreprise.

À Emmanuelle Encrenaz, et à Katell Morin-Allory, rapporteures de cette thèse, pour le temps qu'elles ont consacré à lire ce manuscrit, et pour leurs retours constructifs. Merci également à Arnaud Virazel, à Lars Hedrich, et à Xavier Urbain, examinateurs de cette thèse.

À Laurence Pierre, et à Russ Harmer, membres de mon comité de suivi individuel. Merci pour votre participation aux comités annuels, et pour le temps consacré au suivi de l'avancement de la thèse.

À Gabriel Radanne, et à Ludovic Henrio, pour la collaboration scientifique. Merci notamment pour les discussions qui ont conduit à l'élaboration de ce qui deviendra la sémantique switch-based.

À mes collègues de Verimag, permanents et non permanents, administratifs et chercheurs, pour faire de ce labo un lieu si agréable pour travailler. Merci aux doctorants et autres non permanents qui rythment le quotidien au labo avec de bons moments conviviaux : les déjeuners groupés au Crous, les *Frookies* à Eve, les cafés en terrasse sous le charme de Belledonne, les soirées jeux de société, etc. Merci à Dr. Aina, Dr. Akshay, Alban (qui connait tout Wikipédia !), Alexandre B (qui sait faire cohabiter Haskell et OCaml !), Dr. Alexandre H, Ana, Andrei, Arno (le co-bureau qui sait à la fois m'expliquer la physique et m'éclairer sur des choix esthétiques), Baptiste (mon co-bureau qui m'a tant appris sur la didactique et autres affaires trop obscures pour moi simple hardeux), Basile G (et sa culture impressionnante en Informatique et au-delà), Dr. Basile P, Benjamin B, Benjamin G (bon courage pour la thèse !), Christoffer, Etienne (mon petit frère de thèse), Dr. Hadi, Hichem, Hugo,

Dr. Lucas, Dr. Léo, Neven (ou magicien typst, entre autres), Dr. Nicolas (salutations Lyonnaises et Grenobloises ;)), Sirine (bon courage pour la thèse !), Dr. Thomas M, Dr. Thomas V, Timothée, et Dr. Vincent. Merci également à tous les stagiaires qui sont passés par là.

À mes collègues du LIP, que je voyais moins souvent vu que j'osais rarement prendre le train pour Lyon, Merci pour tous les moments partagés au fil de ces années.

À mes collègues d'Aniah, Merci pour votre soutien durant ces années de thèse. J'ai beaucoup apprécié de travailler à vos côtés. Je garde de très beaux souvenirs des moments de convivialité, en entreprise ou à la montagne ! Merci beaucoup aux ingénieurs d'application qui m'ont toujours éclairé sur des sujets que je maitrisais peu. Merci notamment à Ahmed, Bedier, Diego, Kenza, Meryam, Vincent B, et Vincent S. Merci à Danaël, Nicolas, et Tom, pour leur support sur des sujets IT. Merci également à Adrien, Antoine, Blandine, Catarina, Clément, Constant, Hermann, Jean-Pierre, Joana, Kevin, Nessrine, Pierre-Charles, Rémi, Samy, Virginie, et tant d'autres.

À mes parents, et à toute ma famille, Merci pour tout.

Abstract

CHIP DESIGN is a highly complex task. It involves large teams of engineers with a wide range of skills. Their collective goal is to build chips that conform to their respective specifications, and that are bug-free. Despite the efforts made, it is not uncommon for manufactured chips to contain design errors that went unnoticed. Such scenario is a nightmare for semiconductor companies, for the cost of post-fabrication bug fixing is enormous. Companies hence tend to put even more resources toward test and verification than is spent on other design stages. One can clearly see the importance of early error detection in a circuit design project; the earlier an error is discovered, the less it would cost to fix it.

This thesis investigates the problem of error verification in integrated circuits (ICs). The design errors of interest are the so-called electrical errors, which are caused by violations of design rules specific to electrical properties. We study existing circuit verification techniques, to understand their limitations. We then introduce our own approaches and demonstrate their use to detect electrical properties violations in industrial circuits. This is achieved by first defining circuit semantics at transistor level, which can be dealt with as satisfiability modulo theories (SMT) problems. Properties (e.g., absence of an error) are then formalized and checked against the circuit formula using an SMT solver, namely Z3.

In addition to detecting electrical errors, we address the problem of the analysis of circuits reliability, and more specifically, that of finding the greatest lower bound for reliability of a circuit and the associated electrical state. In fact, the conditions in which a circuit is used determine its degradation rate. To each circuit state is associated a reliability as a function of age, for a given failure mechanism and related model parameters. Hence, finding the greatest lower bound for reliability boils down to identifying the least reliable circuit state at a given age. Such goal is reached by solving the corresponding optimization modulo theories (OMT) problem. The resulting information gives circuit designers solid guarantees on the lifetime of their circuits. These guarantees must be safe. Yet, the more realistic such information is, the better.

The approaches introduced in this work use circuit semantics that are derived from the actual circuit behavior. Different semantics, based on different levels of detail in modeling, are explored, implemented, validated against simulation (which is industry-standard in verifying the properties of interest), and compared with each other. The experimental evaluations conducted provide valuable insights on analysis precision-versus-performance tradeoffs.

Keywords. Circuit semantics, electrical errors, electrical rule checking, formal verification, integrated circuits, optimization modulo theories, reliability, satisfiability modulo theories, transistor level.

Résumé

La conception de puces électroniques est une tâche extrêmement complexe. Elle implique de vastes équipes d'ingénieurs dotés d'un large socle de compétences. Leur objectif collectif est de fabriquer des puces conformes à leurs spécifications respectives et exemptes de bogues. Malgré les efforts déployés, il n'est pas rare que les puces fabriquées contiennent des erreurs de conception qui sont passées inaperçues. Un tel scénario est un cauchemar pour les entreprises de semiconducteurs, le coût de correction des bogues post-fabrication étant énorme. Les entreprises consacrent ainsi plus de ressources aux tests et à la vérification qu'elles n'en consacrent à d'autres étapes de la conception. Il est donc important que les erreurs de conception soient détectées le plus tôt possible; le plus tôt une erreur est découverte, moins il en coûtera de la réparer.

Cette thèse étudie le problème de la vérification de certaines erreurs dans les circuits intégrés. Nous nous intéressons en particulier aux erreurs de conception dites électriques. Celles-ci sont causées par des violations des règles de conception propres aux propriétés électriques. Nous présentons les techniques de vérification de circuits existantes afin d'en comprendre les limites. Nous présentons ensuite notre propre approche et démontrons son utilisation pour détecter des violations de propriétés électriques dans des circuits industriels. Pour ce faire, nous définissons d'abord la sémantique de circuits au niveau transistor, que l'on peut considérer comme un problème de satisfiabilité modulo théories (SMT). Les propriétés (e.g., l'absence d'une erreur) sont ensuite formalisées et vérifiées par rapport à la formule du circuit à l'aide d'un solveur SMT, en l'occurence Z3.

En plus de la détection des erreurs électriques, nous abordons le problème de l'analyse de la fiabilité des circuits, et plus précisément, celui de déterminer la borne inférieure pour la fiabilité d'un circuit et de l'état électrique lui correspondant. En effet, les conditions d'utilisation d'un circuit déterminent son taux de dégradation. À chaque état du circuit est associée une fiabilité en fonction de l'age, et cela pour un mécanisme de défaillance donné et ses paramètres de modèle. Donc, rechercher la borne inférieure pour la fiabilité revient à identifier l'état du circuit le moins fiable à un moment donné. Cet objectif est atteint en résolvant le problème d'optimisation modulo théories (OMT) correspondant. Les informations qui en résultent donnent aux concepteurs de circuits de solides garanties sur la durée de vie de leurs circuits. Ces garanties doivent être sûres. Toutefois, il est également important que ces informations soient les plus réalistes possibles.

Les techniques présentées dans cette thèse utilisent des sémantiques de circuits dérivées du comportement réel du circuit. Différentes sémantiques, basées sur différents niveaux de détail dans la modélisation, sont explorées, mises en œuvre, validées par rapport à la simulation (qui représente un standard pour la vérification des propriétés considérées), et comparées entre elles. Les résultats empiriques fournissent une idée sur le compromis entre la performance et la précision de l'analyse.

Mots clés. Circuits intégrés, erreurs électriques, fiabilité, niveau transistor, optimisation modulo théories, satisfiabilité modulo théories, sémantique de circuits, vérification des règles électriques, vérification formelle.

Note. Un résumé substantiel en français de cette thèse commence dès la page 123.

Contents

1 Introduction **1**
- 1.1 Motivation and Objectives . . . 2
- 1.2 Industrial Context of this Thesis . . . 3
- 1.3 Structure of this Thesis . . . 3

2 Formal Methods **5**
- 2.1 A Tour of Formal Methods . . . 5
  - 2.1.1 Model Checking . . . 5
  - 2.1.2 Abstract Interpretation . . . 9
  - 2.1.3 Theorem Proving . . . 11
- 2.2 Decision Procedures . . . 11
  - 2.2.1 Satisfiability . . . 11
  - 2.2.2 Satisfiability Modulo Theories . . . 12
  - 2.2.3 Optimization Modulo Theories . . . 14
  - 2.2.4 System Modeling and Verification . . . 15
  - 2.2.5 Discussion on Soundness and Completeness . . . 15
- 2.3 Methods Used in this Thesis . . . 16

3 Transistor Level Electric Circuit Verification **17**
- 3.1 Transistor Level and Circuit Modeling . . . 17
  - 3.1.1 Circuit Design Flow . . . 18
  - 3.1.2 Transistor Level Terminology and Circuit Behavior . . . 19
  - 3.1.3 Transistor Level Circuit Modeling . . . 22
- 3.2 Electrical Errors . . . 24
  - 3.2.1 Floating Gates . . . 24
  - 3.2.2 Electrical Overstress . . . 25
  - 3.2.3 Missing Level-Shifter . . . 25
  - 3.2.4 Current Leakage . . . 28
  - 3.2.5 Electrostatic Discharge . . . 28
- 3.3 State of the Art in Electrical Rule Checking at Transistor Level . . . 28
  - 3.3.1 Simulation . . . 30
  - 3.3.2 Wire-Based Approaches . . . 30
  - 3.3.3 Ad-Hoc Patterns . . . 32
  - 3.3.4 Switch-Based Abstractions . . . 33
  - 3.3.5 Quantitative Approaches . . . 37
  - 3.3.6 Discussion of Related Work . . . 37
- 3.4 Overview of Contributions and Positioning . . . 37

4 Circuit Semantics **41**
- 4.1 Overview of the Analysis Framework . . . 41
- 4.2 Common Formalism . . . 43
  - 4.2.1 Notation System . . . 43
  - 4.2.2 Circuit Environment . . . 50
- 4.3 Intuition Behind the Semantics of Circuits . . . 53

4.4 First Attempt: Oriented Signal Semantics ($\mathcal{S}^o$) . . . 53
4.4.1 Intuition . . . 55
4.4.2 Limitations . . . 55
4.5 Relational Switch-Based Semantics ($\mathcal{S}^s$) . . . 58
4.5.1 Switch-Based Abstraction Put in Perspective Against Simulation . . . 58
4.5.2 Switch-Based Semantics for Transistors . . . 59
4.5.3 Extension of Switch-Based Semantics to Other Devices . . . 62
4.5.4 Variant of Switch-Based Semantics with Non-null Threshold ($\mathcal{S}^t$) . . . 62
4.6 Quantitative Semantics ($\mathcal{S}^q$) . . . 63
4.6.1 Intuition . . . 63
4.6.2 Device Characteristics . . . 65
4.7 Formal Comparison of Circuit Semantics . . . 71
4.7.1 Illustration of Semantics Adequacy with SPICE . . . 72
4.7.2 Comparison of $\mathcal{S}^s$ with $\mathcal{S}^t$ . . . 73
4.7.3 Comparison of $\mathcal{S}^s$ with $\mathcal{S}^q$ . . . 73
4.7.4 Comparison of $\mathcal{S}^q$ with $\mathcal{S}^t$ . . . 73
4.8 Extensions . . . 77
4.8.1 Alternate Definition of Inputs . . . 77
4.8.2 Towards Sound Quantitative Modeling . . . 78
4.9 Conclusion . . . 78

5 Leveraging Circuit Semantics for Error Detection 81
5.1 Missing Level-Shifter . . . 81
5.1.1 Error Formalization . . . 82
5.1.2 Industrial Case Study . . . 83
5.1.3 Performance Evaluation on Synthetic Benchmarks . . . 86
5.2 Electrical Overstress . . . 87
5.2.1 Property Formulation . . . 87
5.2.2 Experimental Evaluation . . . 89
5.2.3 Future Extension: Optimizing Enumeration of Errors . . . 93
5.3 Conclusion . . . 94

6 Time-Dependent Dielectric Breakdown Worst Case Reliability Analysis 95
6.1 Overview of the Approach . . . 95
6.2 Time-Dependent Dielectric Breakdown Failure Mechanism and JEDEC Model . . . 97
6.2.1 Stress Voltage Applied on Transistors . . . 97
6.2.2 Time to Failure . . . 97
6.2.3 Device Reliability . . . 98
6.2.4 Circuit Reliability . . . 98
6.2.5 Applications and Related Work . . . 99
6.3 Circuit Reliability Analysis . . . 100
6.3.1 Formulation of the Worst Case Reliability . . . 100
6.3.2 Minimizing Reliability Under SMT Constraints . . . 101
6.3.3 Towards Robust Implementation of Circuit Reliability Minimization . . . 103
6.4 Industrial Case Study . . . 104
6.4.1 Experimental Setup . . . 105
6.4.2 Experimental Evaluation . . . 106
6.4.3 Performance Versus Precision Tradeoffs . . . 108
6.4.4 Experimental Validation of Circuit Semantics Against SPICE . . . 109
6.4.5 Impact of the Linearisation Strategy for the Reliability Function . . . 113
6.4.6 Comparison of Our Approach Against the Baseline . . . 113
6.4.7 Alternate Use Case: Minimizing Circuit Lifetime . . . 116
6.5 Conclusion and Future Work . . . 117

7 Conclusion and Future Directions 119
7.1 Contributions and Lessons Learned . . . 119

7.1.1 Circuit Semantics 119
7.1.2 Electrical Error Identification 119
7.1.3 Circuit Reliability Analysis 120
7.1.4 Feedback on the Industrial Context of this Thesis 120
7.2 Future Research Perspectives 121
7.2.1 Modular Analysis Techniques 121
7.2.2 Error Diagnosis: Identifying the Minimal Error Model 121

A Résumé Substantiel **123**

B Oriented Signal Circuit Semantics **127**
B.1 Virtual Nets 127
B.2 Voltage Abstraction 127
B.3 Net Status 128
B.4 Transistor States 128
B.5 Net Drivers 129
B.6 Circuit Formula 131
B.7 Example 131

C Various Attempts of Quantitative Semantics **133**
C.1 $r_{\mathtt{ON}}/r_{\mathtt{OFF}}$ without Threshold ($\mathcal{S}_1^q$) 133
C.2 $r_{\mathtt{ON}}/r_{\mathtt{OFF}}$ with Threshold ($\mathcal{S}_2^q$) 134
C.3 Variable $r_{\mathtt{ON}}/r_{\mathtt{OFF}}$ ($\mathcal{S}_3^q$) 137
C.4 A Three Domains Based Modeling ($\mathcal{S}_4^q$) 138
C.5 Hyperplanes Based Approximation of Device Characteristics ($\mathcal{S}_5^q \triangleq \mathcal{S}^q$) 140

Publications **141**

Bibliography **143**

Glossary of Notations **153**

Index **159**

List of Figures

1.1 SMT based error detection analysis flow. 2

2.1 Concrete system states ($\mathcal{C}$), bad states ($\mathcal{B}$), and *an* upper approximation ($\mathcal{A}$). 6
2.2 A transition diagram of a system where some states (q_2, q_3, and q_5) are not reachable. . 7
2.3 Use of LTL fundamental operators in program traces. 7
2.4 Use of LTL 'Eventually' and 'Always' operators in program traces. 7
2.5 CEGAR framework. 8
2.6 Expanded BDD for $(a \vee b) \wedge c$ using ordering $a < b < c$. 9
2.7 Reduced BDD for $(a \vee b) \wedge c$ using ordering $a < b < c$. 9
2.8 Bounded model checking (BMC) framework. 9
2.9 Galois connection $\langle \mathcal{C}, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{A}, \preccurlyeq \rangle$. 10
2.10 Abstract interpretation of integers by signs [Burghardt, 2013]. 10
2.11 The eager SMT solving approach. 13
2.12 The lazy SMT solving approach. 13
2.13 Graphical representation of the solution space (colored area) to the SMT problem of E_1. 14

3.1 A typical design and integration flow. 18
3.2 Inverter circuit schematic. 20
3.3 Three states of the Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$. 21
3.4 Transistor characteristics curves obtained from simulation, and their switch-based abstraction. With $V_t = 0.7\,\text{V}$. 22
3.5 Circuit states of the two-input Inverter for different states of net `Z`. With $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{VDD})$. 24
3.6 Current-mirror schematic. 24
3.7 One-stage buffer circuit schematic. 25
3.8 Buffer states for different input (`A`) values. With $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{VDDL}) < \mathcal{V}(\texttt{VDDH})$. 26
3.9 A fix of the buffer circuit of Figure 3.7 using a level-shifter. 26
3.10 An example of a level-shifter to be used in Figure 3.9. 27
3.11 Level-shifter states. 27
3.12 Taxonomy for transistor level electrical rule checking approaches. 29
3.13 Wire-based abstraction of an Inverter. 30
3.14 Inverter circuit with a power-down mode. 35
3.15 A three-transistor based disposition constituting an electrical path from supply (`VDD`) to ground (`GND`). 36

4.1 (Reminder) SMT based error detection flow. 42
4.2 OMT based worst-case reliability analysis flow. 43
4.3 Two-stage buffer circuit schematic. 44
4.4 Transistor terminals. 45
4.5 Resistor terminals. 46
4.6 Diode terminals. 46
4.7 Example circuit schematic containing different device types. 48
4.8 Two-input multiplexer schematic. 50
4.9 Buffer circuit input environment. 51
4.10 AND gate. 51

4.11 AND gate environment, assuming pin NAND can be used as both input and output. . . . 52
4.12 Inverter circuit components. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$. 54
4.13 Drain values (Z1 and Z2) of devices M1 and M2 from Figure 4.12. 54
4.14 (Reminder) Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$. 54
4.15 Inverter circuit (Figure 4.14) simulation output. 54
4.16 (Reminder) Inverter circuit. 55
4.17 Inverter circuit (Figure 4.16) output construction with the oriented signal abstraction. . 55
4.18 Three states of the Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$. 56
4.19 CMOS-based NAND gate. 57
4.20 Variant of a non CMOS based NAND circuit. 57
4.21 NAND gate (Figure 4.19) encoding flow using oriented signal semantics. 58
4.22 NAND gate (Figure 4.19) encoding flow using relational switch-based semantics. 58
4.23 Nets voltages of the buffer circuit (Figure 3.7, page 25) with respect to $\mathcal{V}(\texttt{A})$. 59
4.24 Two device gate configurations of the PMOS transistors chain. With $\mathcal{V}(\texttt{GND})+V_t < \mathcal{V}(\texttt{VDD})$. 61
4.25 Diode-connected transistor in series with a resistor. 64
4.26 I-V relations of devices R0 (in blue) and M1 (in red) of the circuit of Figure 4.25, where the space of solutions of the circuit is empty. 65
4.27 Comparison of nets' voltage modeling of the buffer circuit (Figure 3.7, page 25) with respect to $\mathcal{V}(\texttt{A})$ between SPICE and quantitative semantics $\mathcal{S}^q$. 66
4.28 Level 1 SPICE model based I-V characteristics of a device X, for NMOS (left) and PMOS (right). 67
4.29 Linear approximation of I-V characteristics of a device X, for NMOS (left) and PMOS (right). 68
4.30 Abstraction of NMOS regions of operation for a transistor X. 68
4.31 Abstraction of PMOS regions of operation for a transistor X. 68
4.32 Linear approximation of the current of a diode $D \in \mathbf{D}_{\texttt{DIO}}$. 70
4.33 Graphical representation of the relation between circuit models in the switch-based semantics ($\mathcal{S}^s$), switch-based semantics with threshold ($\mathcal{S}^t$), and quantitative semantics ($\mathcal{S}^q$). 72
4.34 (Reminder) Inverter circuit schematic. 72
4.35 Input-output relation for the Inverter circuit (Figure 4.34), from semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$, and from simulation (SPICE). 72
4.36 Inverter circuit with a resistor at the input interface. 77
4.37 Two modeling approaches of NMOS characteristics for a set value of $\mathcal{V}_{\text{GD}}(X)$, for $X \in \mathbf{D}_{\texttt{NMOS}}$ and $V_t = 0.2\,\text{V}$. 79

5.1 (Reminder) Use of a level-shifter to connect two Inverters to build a safe buffer circuit. . 82
5.2 A PMOS device at the interface of two power domains, in a steady state corresponding to an MLS error. 83
5.3 Comparing speedups among semantics $\mathcal{S}^q$ vs. $\mathcal{S}^t$, and $\mathcal{S}^s$ vs. $\mathcal{S}^t$. 84
5.4 Time spent in the solving phase for each analyzed case. 85
5.5 Two-by-two comparisons of time spent in the solving phase for each analyzed case. . . . 85
5.6 N-bit multi-supply full-adder. 86
5.7 Full-adder benchmarks analysis results. 87
5.8 Average number of EOS devices per solver run. 88
5.9 Total EOS detection time per circuit, with and without using incremental solving, for both choices of input constraints. 89
5.10 Termination of EOS analyses for a 500 s time-limit, with different approaches. 91
5.11 SPICE total simulation time, per circuit, for EOS detection. The '+' marks indicate circuits for which analysis did not terminate after the 500 s time-limit is reached. 92
5.12 Total solving time for EOS analyses, with $\mathcal{R}_{\mathbf{I}}$ strict, compared among different circuit semantics. The '+' marks indicate circuits for which analysis did not terminate after the 500 s time-limit is reached. 92
5.13 Two-by-two semantics' comparison of solving time (s), with $\mathcal{R}_{\mathbf{I}}$ strict. 93

5.14 Comparison of semantics' solving time (s) with respect to the choice of $\mathcal{R}_\mathbf{I}$ ($\mathcal{R}_\mathbf{I}$ strict vs. $\mathcal{R}_\mathbf{I}$ relaxed). . . . 93

6.1 (Reminder) OMT based worst-case reliability analysis flow. . . . 96
6.2 Two-input Inverter, with different PMOS and NMOS device shape parameters. . . . 99
6.3 Reliability of the two-input Inverter (Figure 6.2) with respect to the circuit state. . . . 99
6.4 Piecewise affine approximation of $\omega = \ln(\mathscr{R}_{\text{MOS}}(\delta_{\mathcal{V}}(M), t_{\text{ref}}))$. . . . 102
6.5 Polyhedron representing a 3-dimensional SMT problem, with an excluded face representing the optimum of some objective function. . . . 104
6.6 Comparison of Z3 solving time based on the use of the solver's incremental feature, with $\mathcal{S}^s$. . . . 106
6.7 Termination of Z3 solving with respect to $|\mathbf{D}_{\mathtt{MOS}}|$. . . . 107
6.8 Comparison of reliability lower-bound at age $t_{\text{ref}} = 5\,\text{yr}$, obtained based on circuit semantics used. . . . 107
6.9 Comparison of the solving time for minimizing reliability based on the circuit semantics used. The + marks mean that the 120 s time limit is reached with at least one of the two semantics and the solver did not terminate. . . . 108
6.10 Comparing speedups among semantics $\mathcal{S}^q$ vs. $\mathcal{S}^t$, and $\mathcal{S}^s$ vs. $\mathcal{S}^t$. . . . 108
6.11 Comparison of reliability lower-bound at $t_{\text{ref}} = 5\,\text{yr}$, obtained with $\mathcal{S}^q$ against $\mathcal{S}^s$. . . . 109
6.12 Comparison of the solving time using $\mathcal{S}^q$ against $\mathcal{S}^s$. . . . 109
6.13 Speedup with $\mathcal{S}^q$ relative to $\mathcal{S}^s$. . . . 109
6.14 SPICE-based approach for identifying a lower-bound on circuit reliability and the corresponding circuit state. . . . 110
6.15 Comparison of reliability lower bound at age $t_{\text{ref}} = 5\,\text{yr}$, obtained from SPICE simulations against circuit semantics, excluding timeouts. . . . 111
6.16 Comparison of the simulation time for finding the lower bound for circuit reliability (at age $t_{\text{ref}} = 5\,\text{yr}$) against the solving time with circuit semantics. The + marks mean that the 120 s time limit is exceeded by the total simulation, or achieved with circuit semantics and the solver did not terminate. . . . 112
6.17 Two-by-two comparison of Z3 solving time, using $\mathcal{S}^s$. . . . 114
6.18 Two-by-two comparison of reliability, using $\mathcal{S}^s$. . . . 114
6.19 Gain obtained using semantics-based analysis over the baseline. . . . 115
6.20 Solver performance for $\mathcal{S}^s$, with respect to circuits' size. . . . 115
6.21 Gain using $\mathcal{S}^s$ over the baseline approach, with respect to the solving time. . . . 116
6.22 Average solving time for binary search with semantics $\mathcal{S}^s$. . . . 116

B.1 Abstraction of voltage levels from actual supplies voltage values. . . . 128
B.2 (Reminder) CMOS-based NAND gate. . . . 132
B.3 NAND gate voltage abstraction. . . . 132

C.1 Inverter's device states and output valuation with $\mathcal{S}^q_1$. We set $\mathcal{V}(\mathtt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\mathtt{VDD}) = 1\,\text{V}$. . . . 134
C.2 (Reminder) CMOS-based NAND gate. . . . 135
C.3 Truncated pull-down network of the NAND gate (Figure C.2). . . . 136
C.4 Illustration of relations between current and voltage for devices M3 and M4 of Figure C.3, where resistance values $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ are set to pre-defined values. No solution is possible. 136
C.5 Illustration of relations between current and voltage for devices M3 and M4 of Figure C.3, where only resistance value $r_{\mathtt{ON}}$ is set to pre-defined value, while $r_{\mathtt{OFF}}$ is variable. An interval of solutions, represented with the thick blue segment, is possible. . . . 137
C.6 Illustration of relations between current and voltage for devices M3 and M4 of Figure C.3, where both resistance values $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ are variable. The magenta-colored area represents the possible solutions. . . . 138
C.7 Equivalent resistance $\mathfrak{R}_{eq}(M)$ of a device M with respect to its applied voltage ($\mathcal{V}_{\text{GS}}(M)$ and $\mathcal{V}_{\text{GD}}(M)$), in $\mathcal{S}^q_4$. . . . 139

List of Tables

3.1 Elements of CDL syntax. 20
3.2 Summary of the approaches to transistor level electrical rule checking. 39

4.1 NMOS regions of operation for a device $X \in \mathbf{D}_{\texttt{NMOS}}$. 67
4.2 Summary of formal semantics comparison. 72

5.1 Statistics related to the case-study. 84
5.2 MLS analysis results. 'E' denotes an MLS Error, and 'N' denotes absence of MLS (No error). 84
5.3 Benchmark composition. 90
5.4 EOS analysis results (timeouts excluded). 'E' denotes an Error, and 'N' denotes absence of EOS (No error). 90

6.1 Illustration of how linearisation intervals for reliability are chosen according to the adaptive approach, for a circuit having supply values 0 V, 1.2 V, and 3.3 V. 102
6.2 Hypotheses regarding circuit semantics use for worst case reliability analysis. 105
6.3 Benchmark composition. 105
6.4 Summary conclusions on the hypotheses initially stated in Table 6.2. 117

B.1 Enumeration of NAND gate solutions (each column represents a solution), numbered s_0–s_8. 132

C.1 NAND gate obtained models with $\mathcal{S}_2^q$. 135

List of Listings

2.1 Checking the satisfiability of $a \wedge (\neg b \vee c)$ in SMT-LIB. 12
2.2 A model obtained for the problem in Listing 2.1. 12
2.3 Checking the satisfiability of E_1 in SMT-LIB. 14
2.4 A model obtained for the problem in Listing 2.3. 14
2.5 Maximizing $2x + y$ while satisfiying the constraints of E_1 in SMT-LIB. 15
2.6 A model obtained for the problem in Listing 2.5. 15
3.1 Inverter circuit netlist. 20
3.2 Buffer circuit netlist. 25
3.3 Fixed buffer circuit netlist. The `level_shifter` module is described in Listing 3.4. . . 26
3.4 Level-shifter circuit netlist. 27
4.1 Two-stage buffer (Figure 4.3) circuit netlist. 45

List of Algorithms

1 Search for devices in electrical overstress . . . 88
2 Search for safe lower-bound circuit reliability . . . 103

1

Introduction

INTEGRATED CIRCUITS (ICs) constitute an essential (sometimes, *the* essential) part of modern machinery. This is especially the case for computer(-like) systems. Recently built ICs tend to get larger in terms of number of transistors they are made of, as they become more complex, and as Moore's law—the observation that the density of transistors in an IC doubles about every two years [Moore, 1965]—continues to be relevant. It is, for example, common for microprocessor chips to accommodate several dozen billion transistors [Alam and Alam, 2020]. Of course, with that comes the burden of verification. Circuit verification has long been a non-trivial task to carry. Since at least the late 90s, the trend has been to spend 40–50 % of the design cost on validation and testing [Burger and Goodman, 1997]. In 2024, for most IC design projects, 50–60 % of the entire project time is spent in verification [Foster, 2024]. Despite multiple efforts in the area of circuit verification, new application-specific integrated circuit (ASIC) designs are still subject to logical and functional flaws, among which at least 60 % were caused by design errors in the period 2016–2024 [Foster, 2024]. Some of these errors are functional and can be discovered at register-transfer level (RTL) [Gupta, 1992; Ashar et al., 1998; Irfan et al., 2016]. Others are physical. They are detectable at the layout level, where information about geometry and placement is available [Gibson et al., 2010; Kollu et al., 2012; Lescot et al., 2015]. Between RTL and layout level, there are different other sorts of errors and properties to reason about. It is in that intermediate spot that the so-called *electrical* errors are to be identified—which is the focus of this thesis. Electrical properties require knowledge about the circuit schematic and the power scenarios associated with it. Such information is accessible at transistor level and lower levels of abstraction, like the layout level, or even the physical level.

Design rules have been established by categorizing existing bugs to avoid design errors. These rules aim at protecting the circuit against various types of damage and malfunction, in which case violations may lead to functional errors as well. We consider that there are two kinds of verification associated: (1) design rule checking (DRC), which aims at verifying constraints related to the physical layout of the circuit, and (2) electrical rule checking (ERC), which verifies the electrical behavior of the circuit. It is the latter that this thesis focuses on.

Existing ERC approaches are mostly simulation-based. Some of them combine simulation with other ad-hoc analyses. Conducting simulations can be time-consuming, especially when the goal is to verify the absence of a specific error over a large number of circuit states, let alone the whole circuit's state space. Sometimes, the state space can be even infinite if we consider continuous values of voltages, which is particularly the case in multi-supply designs. Classical analysis techniques include voltage propagation algorithms and circuit topology analysis. Although such techniques can be fast, they are infamous for resulting in plenty of false alarms. In this regard, formal methods have the potential of performing better in terms of false alarm reduction. Yet, the use of formal approaches in the context of ERC was rarely studied. By 2022—the year this thesis began—some of the most remarkable works included the use of satisfiability (SAT) solving for the detection

of short-circuit conditions in logic circuits [Afonso and Monteiro, 2017a], and the verification of analog circuit parameters using SAT solving [Miller and Brewer, 2013]. While such works showed promising results for using formal methods in the context of ERC, no additional research directions were explored ever since. At the beginning of this thesis, we lacked verification techniques which go beyond SAT solving and which are *general-purpose* (in the sense that they are useable to verify different sorts of errors, and not limited to a single use case they were initially designed for). We did, as well, lack approaches where circuit analysis is topology-agnostic and where analyzability does not depend on the circuit's power configuration. Several modeling approaches were explored in this thesis with all of these aspects in mind. Their usability was studied on multiple use cases, supported both with formal explanations and empirical evaluation.

1.1 Motivation and Objectives

In this work, we target not only digital circuits but multi-supply designs at large, either purely digital (i.e., circuits which consist in building logic functions) or with some analog parts in them. To do so, we define circuit semantics at transistor level, which we use to generate circuit formulas in the form of satisfiability modulo theories (SMT) problems. The corresponding verification flow is depicted in Figure 1.1. The analysis approaches we introduce are applicable to all circuit descriptions (notably, multi-supply designs with analog parts), and are highly customizable. This means that user assumptions can be considered, in addition to the possibility to formulate custom electrical properties and verify them.

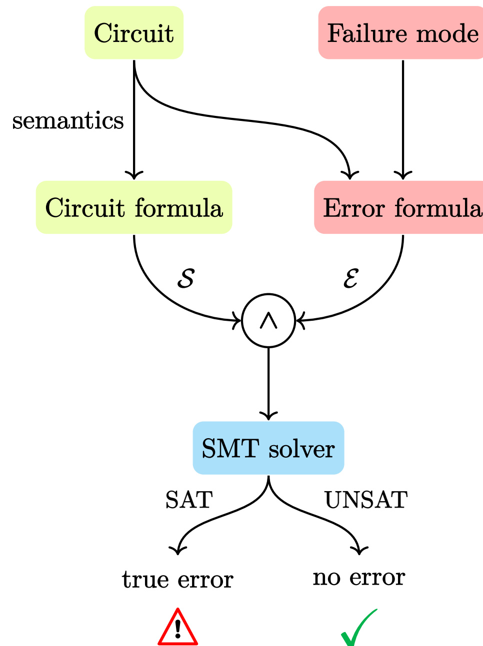


Figure 1.1: SMT based error detection analysis flow.

The number of false alarms produced by a verification tool reflects the degree of coarseness of the modeling abstractions made. Some abstractions do not take into account the behavior of the circuit, as is the case for standard approaches such as static voltage propagation based ones [Lescot et al., 2012; Zwerger and Graeb, 2012, 2014]. In such case, there will be some circuit states which are reported as erroneous although not being actually feasible, due to omitting circuit semantics. Instead, we look for the level of detail adequate enough for the task at hand, and propose circuit semantics, as SMT formulas, on that basis. Using those circuit semantics in our verification framework allows us to reduce the number of false alarms, without missing true electrical errors.

We apply the circuit semantics developed in this thesis on two distinct problems. The first one is the problem of checking the possibility of an error configuration on a circuit. The electrical

errors we study in this regard are electrical overstress (EOS) and missing level-shifter (MLS). The second usage of circuit semantics is the analysis of the reliability of circuits with respect to the time-dependent dielectric breakdown (TDDB) failure mechanism. More precisely, we formulate and solve the problem of identifying the worst-case circuit state in terms of aging. For the second use case, we make use of optimization capabilities of modern SMT solvers, like the νZ extension [Bjørner and Phan, 2014] of the Z3 solver [De Moura and Bjørner, 2008], in order to formulate objective functions and solve them while satisfying the hard constraints of the SMT problem.

1.2 Industrial Context of this Thesis

This thesis was conducted as part of the Convention Industrielle de Formation par la Recherche (CIFRE) program, in collaboration with the ANIAH company [Aniah, 2025]. This collaboration helped us, through exchanges with engineers in the semiconductor industry, to better understand design issues caused by electrical errors and the challenges related to ERC. It also allowed us to continuously assess, empirically, the approaches developed against industrial case studies from real-life circuits. As a matter of fact, virtually all experimental evaluations presented later in this thesis were run on circuit design databases coming from ANIAH.

1.3 Structure of this Thesis

This thesis is organised in chapters. In addition to this introduction, there are six more chapters.

Chapter 2—Formal Methods presents an overview of formal verification methods, and focuses on the techniques used in this thesis.

Chapter 3—Transistor Level Electric Circuit Verification presents the domain background for circuit design, with an emphasis on transistor level and associated terminology. It discusses electrical errors and states the problem at hand. It presents and discusses existing ERC approaches, and prospects for improvements. The contents of this chapter are mostly based on our survey article [Ferres et al., 2025], which provides an overview of existing ERC techniques, proposes a new classification taxonomy, and highlights research challenges in this domain.

Chapter 4—Circuit Semantics presents circuit semantics developed during this thesis. The overall approach of our verification flow was initially presented in our publication [Ferres et al., 2023], and switch-based circuit semantics were detailed later in [Oulkaid et al., 2024]. Extensions to these semantics, as well as introduction of new quantitative ones, is part of our journal article [Oulkaid et al., 2025]. The chapter thoroughly compares them all formally. Further comparisons, with an emphasis on experimental evaluation, are performed, later, in Chapters 5 and 6.

Chapter 5—Leveraging Circuit Semantics for Error Detection shows how circuit semantics of Chapter 4 can be applied to address circuit verification problems. It is supplemented with industrial use cases, for which experimental evaluations are conducted. Applications include the detection of MLS, as done in our paper [Oulkaid et al., 2024], as well as EOS analysis, as we present in our journal article [Oulkaid et al., 2025].

Chapter 6—Time-Dependent Dielectric Breakdown Worst Case Reliability Analysis addresses a sub-problem of circuit reliability analysis for the TDDB failure mode: that of identifying the worst-case circuit state. It formalizes such problem and demonstrates the use of circuit semantics (introduced in Chapter 4) to solve it, using an optimizing SMT solver. Experimental evaluation is conducted on industrial benchmarks. Contributions of this chapter are also part of a journal article submission [Oulkaid et al., nd].

Chapter 7—Conclusion and Future Directions concludes this work with a summary of contributions, lessons learned, various possible extensions, and future research directions envisaged.

Note. A comprehensive glossary of notations used in this thesis is provided in page 153.

2

Formal Methods

In safety-critical systems—i.e., systems whose failure may cause environmental harm, or injury or death to people in that environment [Sommerville, 2011]—mathematical rigor is highly demanded in specification, design, and verification. Formal methods provide a means to achieve such rigor. They are used to verify safety properties of software and hardware systems, and to establish proofs for correctness (having the systems behave as expected). The underlying techniques rely, mostly, on symbolic or enumerative exploration of the system's state space [Butler, 2001], in a systematic rather in an ad-hoc manner [Wing, 1989]. In Section 2.1, we present an overview of the techniques used for system verification and their domains of application. It is more of a brief panorama showing high-level concepts of most known approaches than a thorough exhaustive study. We then focus more particularly on the methods specifically used in this thesis, namely on decision procedures, in Section 2.2. Finally, in Section 2.3, we present the methods used in this thesis, while briefly positioning them with respect to the state of the art. A much more thorough positioning is kept for the next chapter.

Note. If the reader is already familiar with formal methods, they can skip Section 2.1, which presents formal methods in general. If in addition they are familiar with decision procedures of Section 2.2, which are particularly the area of focus of this thesis, they may jump right away to Section 2.3 where the methods used in this thesis are presented.

2.1 A Tour of Formal Methods

Verifying the conformity of a system against its specification, i.e., the absence of implementation *bugs*, is not an easy task. In order to obtain full assurance that the system verifies some property, one ought to check whether the property holds on each of the system's states. Of course, exhaustive exploration of the whole state space is explosive (especially if such exploration is naively done), and is thus only reasonable to apply on relatively small systems. Otherwise, sophisticated means (as opposed to naive exploration) must be leveraged to cope with this problem.

2.1.1 Model Checking

Model checking (MC) is a well established technique which provides the means of determining whether an abstract system representation satisfies its specification, and, if the property does not hold, exhibits a counterexample showing the source of the problem [Clarke et al., 2009]. It is targeted at systems with discrete evolution, provides *exact* analyses with respect to reality, and has the advantage of exploring system states fully automatically [Clarke et al., 2011]. To surmount the combinatorial blow up of the state space, model checking primarily relies on abstraction—dropping

thus unnecessary details to the properties being checked [Graf and Loiseaux, 1993; Clarke et al., 1994; Hsieh and Levitan, 1998].

Figure 2.1 illustrates an upper approximation ($\mathcal{A}$) of a finite system's states, which necessarily includes all *concrete* system states ($\mathcal{C}$). Having $\mathcal{A}$ as an upper approximation of $\mathcal{C}$ is what ensures that no concrete state is missed by the approximation. This also means that the approximation possibly includes non-sensible states with respect to the concrete system. The bad states ($\mathcal{B}$) correspond to implementation bugs. The intersection $\mathcal{C} \cap \mathcal{B}$ are the states where the implementation does not match the specification. A bug-fix consists of making sure the concrete system includes no bad states, i.e., $\mathcal{C} \cap \mathcal{B} = \emptyset$. Some bad states are completely out of the concrete space, or even out of the abstract one. As a result of $\mathcal{A}$ being an upper approximation of $\mathcal{C}$, if $\mathcal{A}$ includes no bad states (i.e., $\mathcal{A} \cap \mathcal{C} = \emptyset$), then also $\mathcal{C}$ doesn't. However, if it turns out the abstraction contains buggy states, the concrete space may or may not be concerned by such buggy states. The choice of the abstraction is key. A too loose abstraction may barely be useful, if not completely useless, e.g., an abstraction in which the system is always potentially erroneous is not really helpful (it teaches us virtually nothing). A too precise abstraction becomes not so abstract, and is therefore faced with the same scaling issues in exploring the concrete space. A trade-off is to be found.

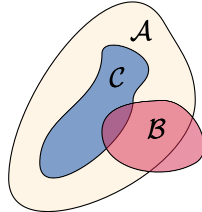


Figure 2.1: Concrete system states ($\mathcal{C}$), bad states ($\mathcal{B}$), and *an* upper approximation ($\mathcal{A}$).

A model checker usually reasons about the *reachable* states of a system. In some contexts, like that of electronic circuit analysis, it may also be used to reason about the *possible* states. The considered states here are the so-called 'steady-states', which are equilibrium states the circuit remains at when led to. However, they may or may not be physically stable.

Reachability analysis is to find out whether there exists a sequence of successive steps starting from an initial state and leading to a specific state, while checking whether such state is possible simply means that it is coherent with the given system semantics. Put simply, a concrete steady-state is not necessarily a concrete reachable state, but all concrete reachable states are part of the set of concrete steady-states. When a property is violated, the counterexample the model checker exhibits is the set of transitions (from an initial state) leading to an error state, in the case of reachability analysis. In the case of checking the possibility of an error state, the counterexample consists of such state itself, regardless of the transitions (if any) that lead to it.

Figure 2.2 depicts an example of a transition diagram, with only states q_0, q_1, and q_4 being reachable from the 'start'—the only possible sequence of transitions being $(t_{0,1}t^*_{1,1}t_{1,4}t_{4,0})^*$. Here, $t_{i,j}$ denotes the transition from a state q_i to a state q_j, and $t^*_{i,j}$ corresponds to taking such transition an arbitrary number of times. There is no sequence which leads to states q_2, q_3, and q_5 although they are legitimate states with respect to the semantics of the system. Reachability analysis is addressed at (but not limited to) problems which essentially appear in the context of concurrent systems—like communicating systems.

Temporal logic [Pnueli, 1977; Lamport, 1994] is a common formalism used in model checking, to reason about properties involving *time*, in a whole or part of an execution trace. The notion of 'time' in temporal logic does not refer to physical time; it simply refers to the fact that events are ordered. There exist several frameworks for temporal logic. A common language used to describe system properties considering linear time (a single timeline of progress) is linear temporal logic (LTL). LTL enriches propositional logic with time-specific operators. It has two fundamental operators, from which more operators can be derived. The two fundamental operators are **Until** and **Next**, denoted, respectively, $\mathcal{U}$ and $\bigcirc$. An example is shown in Figure 2.3, where (1) a property φ_1 holds in a program trace until property φ_2 holds, after which we are indifferent, and (2) a next property (next to a state q_i) that holds is φ, which may also hold or not for forthcoming steps.

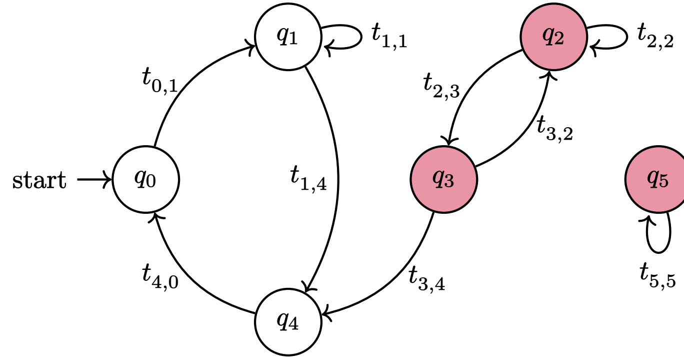


Figure 2.2: A transition diagram of a system where some states (q_2, q_3, and q_5) are not reachable.

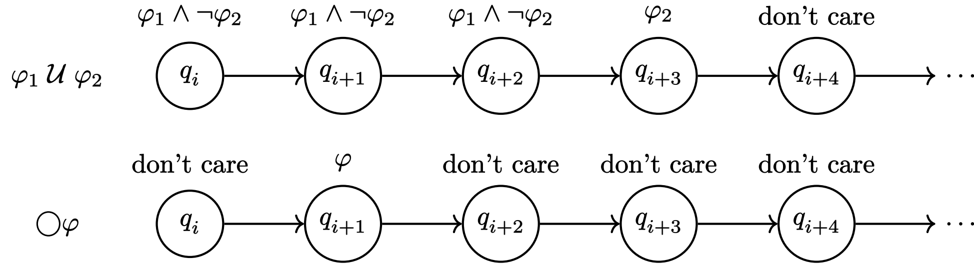


Figure 2.3: Use of LTL fundamental operators in program traces.

Among the derived operators there are **Eventually** (denoted $\Diamond$) and **Always** (denoted $\Box$) operators. They are both constructed from basic operators.

'Eventually' is constructed as

$$\Diamond\varphi \triangleq \text{true}\,\mathcal{U}\,\varphi,$$

which means that at some time in the future φ holds.

'Always' is constructed as

$$\Box\varphi \triangleq \neg(\Diamond\neg\varphi),$$

which means there never is going to be a point in the future where φ doesn't hold.

Both operators are illustrated with the examples of Figure 2.4. For example, in communicating systems, $\Diamond\varphi$ may be 'for every request sent, a response is eventually received', while $\Box\varphi$ can be 'non encrypted communications never occur'.

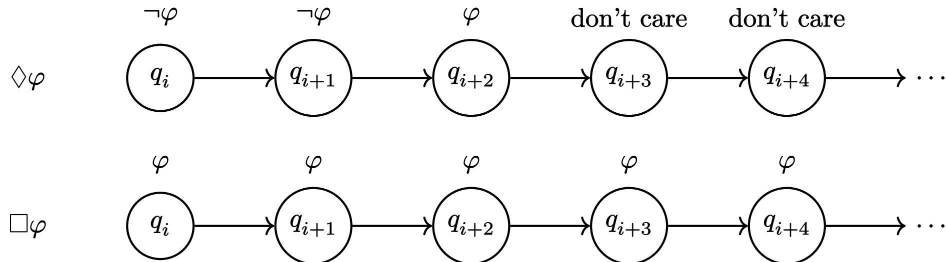


Figure 2.4: Use of LTL 'Eventually' and 'Always' operators in program traces.

When it comes to proving the correctness of a system, famous properties temporal logic can be used to reason about are *safety* and *liveness* properties. Usually, safety properties are described with $\Box$, and liveness properties are described with $\Diamond$. Lamport [1977] defines a safety property as the one which states that something *bad* will not happen during execution, while a liveness property states that something *good* must (eventually) happen. It is proved that every property is the conjunction of a safety property and a liveness property [Alpern and Schneider, 1985]. Distinguishing between

the two properties is especially helpful when deciding how to prove them [Schneider, 1987]. This is more relevant for non-fully-automated formal verification techniques—which we discuss later in this section. For now, we resume our discussion on model checking, and take a look at some of the strategies model checkers implement.

Abstracting the system model is a sine qua non for model checking deployment, yet finding relevant abstractions is not trivial, and doing so manually often requires considerable creativity [Clarke et al., 2000]. Counterexample-guided abstraction refinement (CEGAR) is a technique introduced by Clarke et al. [2000] as a means to automatic abstraction generation. CEGAR starts from an upper approximation of the system model (which can be extremely coarse), and iteratively refines it through several steps, as shown in Figure 2.5. Knowledge about the properties to verify is essential. It is what actually guides abstraction choices, typically in a CEGAR framework [Alwi and Encrenaz, 2014]. The process can conclude that a property holds on the original model if it does on

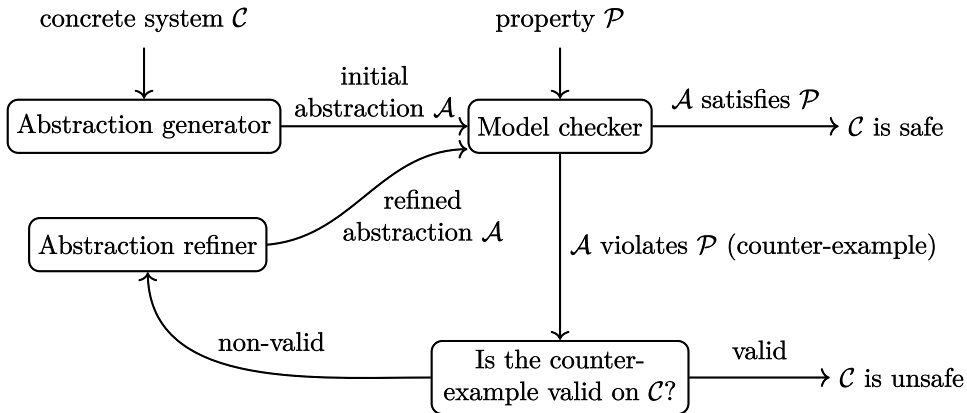


Figure 2.5: CEGAR framework.

the abstract one—since it is an upper approximation. However, finding a counterexample on the abstract model does not necessarily mean that the concrete design is erroneous, in which case the abstraction must be refined based on information obtained from the counterexample. The abstract model can be formalized with the use of Boolean functions, which can be symbolically represented with binary decision diagrams (BDDs) [Lee, 1959; Akers, 1978; Bryant, 1986], which are considered to be compressed representations. To illustrate this, consider function $f(a, b, c) = (a \vee b) \wedge c$, given Boolean variables ordering $a < b < c$. The corresponding binary decision tree can be represented in an expanded way as shown in Figure 2.6. However, it can be reduced to a more compact representation—shown in Figure 2.7—by (1) compressing the nodes for which the resulting value is indifferent to the value of the corresponding Boolean variable (e.g., $f(a = 0, b = 0, c)$ always evaluates to 0 regardless of the value of c), and (2) reducing identical sub-trees into a single shared copy (e.g., $f(a = 1, b, c) = f(a = 0, b = 1, c) = c$).

Still, BDDs are not spared from exponential growth. Obtaining small BDDs largely depends on variable ordering, and choosing optimal ordering is not an easy task. Algorithms for bounded model checking (BMC) were developed to cope with this problem, and have been successfully used in industry [Biere, 2021]. BMC uses propositional formulas as the basis of encoding of system states, and runs multiple iterations to check whether a property holds only on a limited number of (successive) states [Clarke et al., 2001]. The verification itself is delegated to some decision procedure, e.g., a satisfiability (SAT) solver. We discuss decision procedures more in detail in Section 2.2, and SAT solvers in Section 2.2.1.

BMC is primarily useful to answer whether it is possible for some desired system state (or for some bad state, in the context of verification) to be reached in k_{ref} steps starting from the initial state. It searches for counterexamples witnessing violation of some property, in a full automatic manner. It cannot, however, prove the absence of errors in executions beyond the initially set completeness threshold k_{ref}. Figure 2.8 shows the corresponding verification flow, which takes at most k_{ref} steps to conclude. An obvious challenge relies in finding sufficiently small values for such

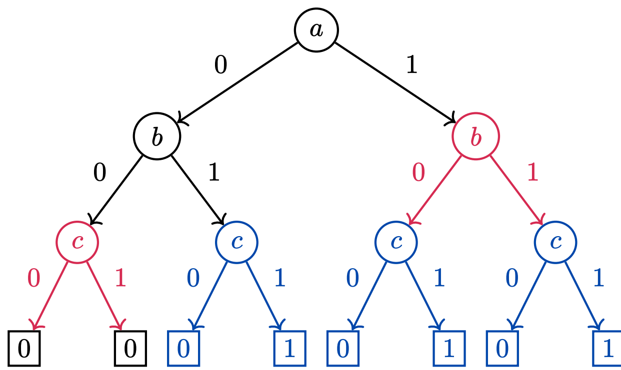


Figure 2.6: Expanded BDD for $(a \vee b) \wedge c$ using ordering $a < b < c$.

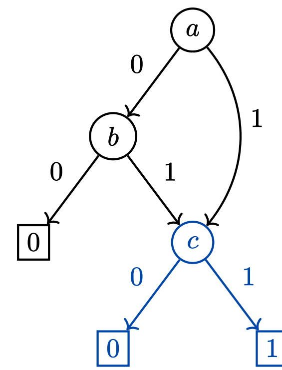


Figure 2.7: Reduced BDD for $(a \vee b) \wedge c$ using ordering $a < b < c$.

completeness threshold [Kroening et al., 2011]. BMC has shown to solve problems that BDD-based techniques cannot solve, yet there are still problems that are better solved with BDDs.

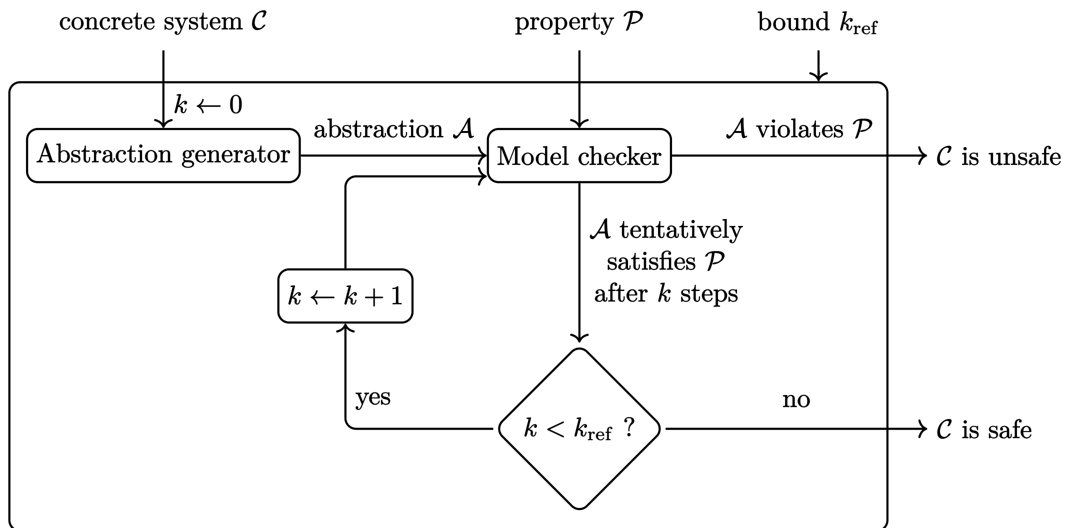


Figure 2.8: Bounded model checking (BMC) framework.

Model checking based verification techniques are inherently inefficient in terms of scalability [Visser et al., 2003]. This is due to its exhaustive exploration nature, which leads to the state explosion problem.

2.1.2 Abstract Interpretation

Abstract interpretation (AI) provides an interesting alternative which scales better (at the cost of precision). It is primarily used for static program analysis, where abstract semantics (as opposed to concrete semantics) are attached to programs in order to reason about them [Cousot and Cousot, 1977]. Information is abstracted from the concrete domain ($\mathcal{C}$) into an abstract domain ($\mathcal{A}$), and the relation between the two domains is described by a *Galois connection* as shown in Figure 2.9. Galois connections are the mathematical backbone for abstract interpretation. They are used to relate between concrete program semantics and their abstractions, in a way that facilitates reasoning. A Galois connection is a correspondence between two partially ordered sets. In this case, these are $\langle \mathcal{C}, \sqsubseteq \rangle$ and $\langle \mathcal{A}, \preccurlyeq \rangle$. $\sqsubseteq$ is the partial order on $\mathcal{C}$, and $\preccurlyeq$ is the partial order on $\mathcal{A}$. Both the abstraction function $\alpha \colon \mathcal{C} \longrightarrow \mathcal{A}$ and concretization function $\gamma \colon \mathcal{A} \longrightarrow \mathcal{C}$ preserve the order, i.e., α and γ are

monotone. The order between each two elements of $\mathcal{C}$ must be the same for the corresponding elements in $\mathcal{A}$, and vice versa. Formally, this is written:

$$\forall c \in \mathcal{C}, \forall a \in \mathcal{A}, \alpha(c) \preccurlyeq a \Leftrightarrow c \sqsubseteq \gamma(a)$$

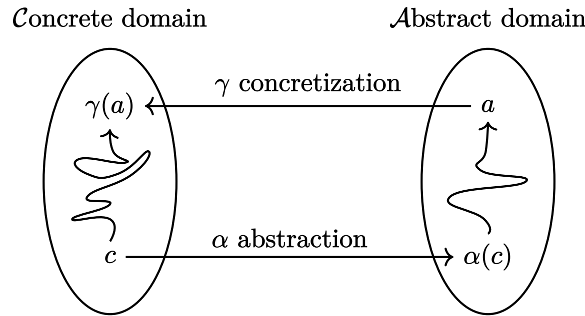


Figure 2.9: Galois connection $\langle \mathcal{C}, \sqsubseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{A}, \preccurlyeq \rangle$.

In abstract interpretation, actual program numerical variable values are, for example, abstracted into their sign ($+$, $-$, or 0). Figure 2.10 shows the corresponding Hasse diagram (mathematical representation of partially ordered sets). In a Hasse diagram for a partially ordered set $\langle S, \leq \rangle$, an element p appears lower than an element q if $p \leq q$. For example, in Figure 2.10, $\{\}$ is lower than $\{0\}$, and $\mathbb{Z}_{<0}$ is lower than $\mathbb{Z}_{\leq 0}$. The highest item in the diagram is $\mathbb{Z}$ (the set of integer numbers). It is referred to as 'Top', and includes all the items below it (i.e., the entirety of items in the diagram). Top gives the lesser knowledge about a variable. It says that the variable is an element of $\mathbb{Z}$, but nothing more than that. The lowest item in the diagram is $\{\}$, which corresponds to no value. It is the 'Bottom'. Between Top and Bottom lie all other possibilities. Variables can be projected from the left diagram to the right diagram based on the interval of values they belong to.

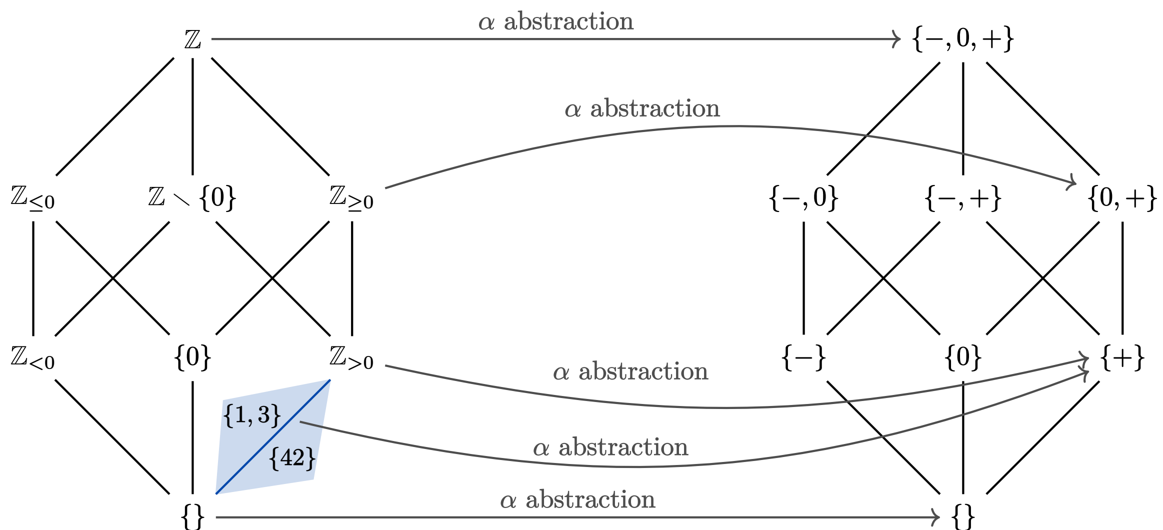


Figure 2.10: Abstract interpretation of integers by signs [Burghardt, 2013].

The loss of precision induced by abstractions may indeed lead to false alarms, but it is extremely useful when it comes to prove some (decidable) properties. Examples of properties abstract interpretation is good at proving are runtime errors like division-by-zero, square root of negative numbers, out-of-bound array indexing, etc. [Blanchet, 2002]. In general, such properties involve reasoning

about numerical variables, where abstraction-less exploration of system states is explosive. Typically, N-bit integer encoding would involve exploration of 2^N combinations if no abstraction is made. Unlike model checking, abstract interpretation is not automatic, for choosing proper abstractions relies on expert knowledge.

Many verification tools use abstract interpretation in their reasoning. ASTRÉE [Cousot et al., 2001] is a static analyzer for C and C++ programs. It is based on abstract interpretation, and aims at proving the absence of runtime errors. FRAMA-C is another C program static analyzer, whose *value analysis* plug-in uses abstract interpretation as its theoretical framework to compute the set of possible values for variables in a program [Cuoq and Prevosto, 2009]. In the hardware realm, a property checking tool named QUESTA PROPCHECK relies on engineering knowledge provided by users to define proper abstractions used in its abstract interpretation framework, while it can automate some specific parts of the design [Siemens, 2025].

2.1.3 Theorem Proving

Verification can also be done via deduction—in what is referred to as theorem proving. Theorem proving is precise, but requires user interaction; there can be no automatic algorithms to theorem proving, for some problems are undecidable. The system must be formalized in a rigorous framework, possibly in mathematical terms. Formal proofs are then generated using interactive theorem provers, like LEAN, ISABELLE, and ROCQ, providing thus strong guarantees on correctness.

Theorem proving is, in practice, used to analyze pre-selected systems in advance. Its user-centered approach—where formalization is mostly manual and highly tailored for the system at hand—makes it hard to generalize proofs even among at-a-first-glance-similar systems. Trying to 'adapt' existing formal proofs of a system to another similar system does not really work in practice. So, it is not reasonable to prove *any* program in the absolute, plus, it is not commercially viable to do so. What would be reasonable is not to prove every system, but rather to prove the system generator (if applicable). That is, in the case of programs, to prove correctness of the compiler instead of that of programs generated by such compiler. Doing so, one will obtain correct-by-construction programs. This is precisely what the COMPCERT optimizing compiler does [Leroy et al., 2016]. It is a popular example of a formally verified compiler, which produces executable code that is proven to behave exactly as specified by the semantics of programs written in the C programming language.

2.2 Decision Procedures

The *Entscheidungsproblem*, first introduced by Hilbert and Ackermann [1928], is the problem of determining the *satisfiability* of first-order logic formulas [Börger et al., 2001]. An algorithm used to solve a decision problem is called a decision procedure. A decision procedure takes the problem as input and terminates with a yes/no answer [Kroening and Strichman, 2008]. The following subsections focus on satisfiability-related decision problems, including when they are combined with optimization problems.

2.2.1 Satisfiability

The Boolean satisfiability problem (SAT for short) refers to determining whether there exists a *valuation* of the variables of a propositional logic formula such that the formula evaluates to 'true' (or $\top$, as opposed to 'false' or $\bot$). If the answer is 'yes', the formula is said to be *satisfiable* (SAT). Otherwise it is *unsatisfiable* (UNSAT). Boolean formulas are formed by connecting Boolean variables (or sub-formulas involving those variables) with logic operators for negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$). A tool implementing decision procedures to solve the SAT problem is named a 'SAT solver'. In addition to the satisfiable/unsatisfiable answer, the SAT solver provides a *model* representing a valid valuation of the variables in the formula (when satisfiable).

SMT-LIB [Barrett et al., 2010] is an international initiative providing a standard interface to satisfiability solvers. It provides a language for expressing problems, and interacting with the actual solvers (e.g., check the satisfiability of a formula, retrieve a satisfying model, etc.). In fact, SMT-LIB covers other theories and problems beyond mere SAT solving. It also deals with *satisfiability*

modulo theories (Section 2.2.2) and *optimization modulo theories* problems (Section 2.2.3). We present both of these problems just after we are done with Boolean satisfiability. We use the SMT-LIB syntax to write the different examples we present hereafter, which we solve using the Z3 solver [De Moura and Bjørner, 2008].

For example, the formula $a \wedge (\neg b \vee c)$ is satisfiable because we can find at least a valuation $(a = \top,\ b = \bot,\ c = \bot)$ for which the formula evaluates to $\top$. In the SMT-LIB syntax, we can state and solve this example, as shown in Listing 2.1. The logic is set to `QF_UF`, which means that we restrict the solver to only use quantifier-free (QF) uninterpreted functions (UF), which only unlocks the SAT solving capabilities of the solver. We unlock further theories in upcoming examples, in the next sections, as needed. The obtained model is shown in Listing 2.2. A model of a formula $\mathcal{F}$ is denoted $\mathcal{M}$. We say that $\mathcal{M}$ satisfies $\mathcal{F}$, and this is denoted $\mathcal{M} \models \mathcal{F}$.

```
1  (set-logic QF_UF)
2
3  (declare-const a Bool)
4  (declare-const b Bool)
5  (declare-const c Bool)
6
7  (assert (and a (or (not b) c)))
8
9  (check-sat)
10 (get-model)
```

Listing 2.1: Checking the satisfiability of $a \wedge (\neg b \vee c)$ in SMT-LIB.

```
1 sat
2 (
3   (define-fun b () Bool
4     false)
5   (define-fun c () Bool
6     false)
7   (define-fun a () Bool
8     true)
9 )
```

Listing 2.2: A model obtained for the problem in Listing 2.1.

Most SAT solvers take as input a formula in conjunctive normal form (CNF). A CNF-shaped formula is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is either a Boolean variable or its negation. This is due to the fact that CNF is a well structured form that can be obtained with a linear cost from any Boolean formula. There are many algorithms for solving SAT problems. They mostly derive from the well known Davis-Putnam-Logemann-Loveland algorithm (DPLL) [Davis and Putnam, 1960; Davis et al., 1962]. DPLL uses chronological backtracking algorithms. The formula solving process starts by assigning initial valuations to the formula's variables, based on some heuristics. Since the goal is to find a satisfying assignment, the solver tries new assignments whenever a choice leads to UNSAT. It only reaches a conclusion either when a satisfying assignment is found, or when the formula remains UNSAT no matter what variables valuations are chosen. The strategies the solvers implement are advanced enough the search process ought not to start from scratch every time a conflict is reached. Backtracking consists of choosing the number of steps to go backwards before resuming the assignment of truth values to the formula's literals. Many techniques have been proposed to improve the efficiency of SAT solving algorithms. Conflict-driven clause learning (CDCL) is one of the most remarkable works [Marques-Silva and Sakallah, 2002]. An even more radical evolution is the arrival of SMT solvers, which we present in the next section.

2.2.2 Satisfiability Modulo Theories

Like SAT, satisfiability modulo theories (SMT) is the problem of determining the satisfiability of a formula. However, SMT deals with more complex kind of formulas—involving other types of variables beyond Booleans. Variables of an SMT formula can be of any type. This includes, for example, integers and rational numbers. They can be used with different data structures, like Arrays and bit-vectors. Some SMT solvers also support quantifiers ($\forall$, $\exists$) in their logic, but adding quantifiers is generally undecidable.

A classical method implemented by general-purpose SMT solvers is *bit-blasting* [Jia et al., 2023]. Pure bit-blasting consists of rewriting the SMT formula to transform it into a SAT formula. For example, a 32-bit integer (a) is represented with 32 Booleans (a_{31}, a_{30}, ..., a_0). An assertion that $a = 3$ is then encoded as $(a_{31} \Leftrightarrow \bot) \wedge (a_{30} \Leftrightarrow \bot) \wedge \cdots \wedge (a_2 \Leftrightarrow \bot) \wedge (a_1 \Leftrightarrow \top) \wedge (a_0 \Leftrightarrow \top)$, as $(3)_{10}$ stands for '0...011' in Base 2. Bitwise operators, like equalities or inequalities are straightforwardly

translated. For arithmetic operations (e.g., addition and multiplication), translation consists of implementing the circuit corresponding to each operation (e.g., full adder, etc.).

This *eager* approach (Figure 2.11) may be difficult to run on large problems, as the bit-blasted version of the formula must be solved at once, while not benefiting from extra-Boolean theories [Hadarean et al., 2014].

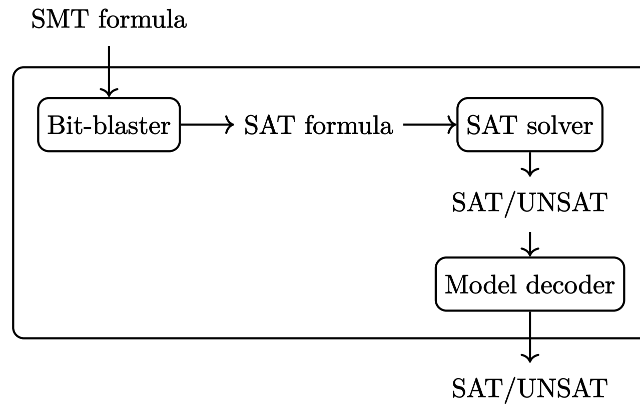


Figure 2.11: The eager SMT solving approach.

A *lazy* approach, instead, invokes a dedicated theory solver. The underlying mechanism (depicted in Figure 2.12) still uses pure SAT solving for instances translated from SMT. For instance, theory solvers handle non-Boolean parts which may mix Booleans and numerical constraints, while checking formulas resulted from different fragments of the problem are taken care of by SAT—which treats the theory atoms as Boolean variables [Junttila, 2020]. These approaches are usually called DPLL(T) or CDCL(T)—with T standing for *Theory*. The solving of the instances composing the problem goes back and forth between the SAT solver and the theory solver, until a valuation satisfying both the propositional logic instance and the underlying theory is found, or when a conclusion the problem is unsatisfiable is achieved.

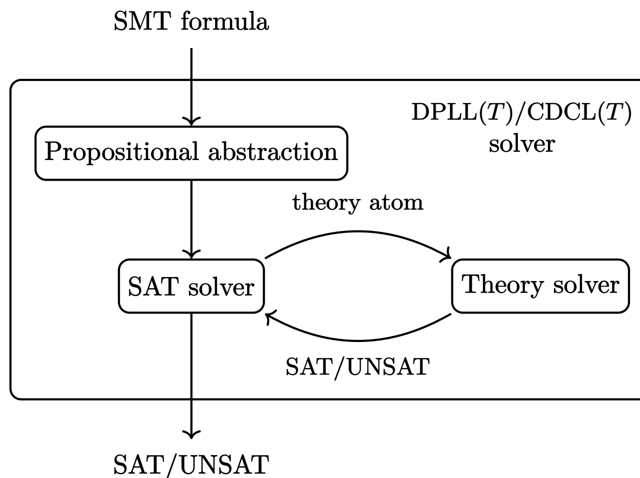


Figure 2.12: The lazy SMT solving approach.

Modern SMT solvers embed portfolios of theories (e.g., linear and non-linear arithmetic, integer numbers, real numbers, etc.), and are able to choose the more efficient ones depending on the input formula. An example of such tool is the Z3 SMT solver by De Moura and Bjørner [2008].

In SMT, we can express problems constraining variables with linear and non-linear arithmetic. We consider an example SMT formula, described in E_1, which constrains two numerical variables

x and y.

$$(y \leq 4 - x) \land (y \leq 1) \land \left(y \geq 2 - \frac{1}{2}x\right) \land \left(y \geq \frac{1}{12}x^2\right) \tag{E_1}$$

A graphical representation of the constraints as well as the solution space are shown in Figure 2.13. We encode the problem in SMT-LIB, with enabling quantifier-free non-linear arithmetic theory (`QF_NRA`), as shown in Listing 2.3. Enabling such theory allows us to express logic constraints reasoning about non-linear expressions in terms of the variables x and y, while excluding theories related to quantifiers (which we don't use anyway). A solution yielded by the solver is shown in Listing 2.4. In case the intersection of all constraints is empty, the solver would have answered that the formula is unsatisfiable—yielding no model.

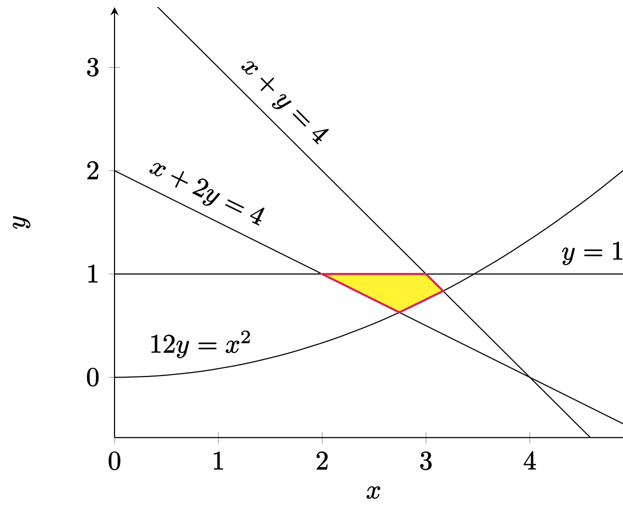


Figure 2.13: Graphical representation of the solution space (colored area) to the SMT problem of E_1.

```
1  (set-logic QF_NRA)
2
3  (declare-const x Real)
4  (declare-const y Real)
5
6  (assert (<= y (- 4 x)))
7  (assert (<= y 1))
8  (assert (>= y (- 2 (/ x 2))))
9  (assert (>= y (/ (* x x) 12)))
10
11 (check-sat)
12 (get-model)
```

Listing 2.3: Checking the satisfiability of E_1 in SMT-LIB.

```
1  sat
2  (
3    (define-fun y () Real
4      (/ 7.0 8.0))
5    (define-fun x () Real
6      3.0)
7  )
```

Listing 2.4: A model obtained for the problem in Listing 2.3.

2.2.3 Optimization Modulo Theories

A model yielded by an SMT solver is arbitrary. Yet, many SMT solvers' users have applications where the choice of assignments matters. They, for example, tend to build algorithms on top of the SMT solver in order to get optimal assignments with respect to their goals. Some SMT solvers, like νZ [Bjørner et al., 2015], provide features for expressing and solving optimization problems alongside checking the satisfiability of the hard constraints of an SMT problem.

The νZ optimizing SMT solver offers the ability to instruct the solver to maximize or minimize a term (either integrer, real, or bit-vector), as well as to assert *soft* constraints with associating

some weights to them. As part of this thesis, we only consider the objective function optimization feature of νZ.

To illustrate its use, we add an optimization objective to the same SMT problem of E_1. The objective consists of maximizing $2x + y$. The corresponding `maximize` command is added to the SMT-LIB syntax as shown in Listing 2.5. Listing 2.6 shows the obtained maximal values of $2x + y$ as well as the corresponding model in terms of variables x and y.

```
1  (set-logic QF_NRA)
2
3  (declare-const x Real)
4  (declare-const y Real)
5
6  (assert (<= y (- 4 x)))
7  (assert (<= y 1))
8  (assert (>= y (- 2 (/ x 2))))
9  (assert (>= y (/ (* x x) 12)))
10
11 (maximize (+ (* 2 x) y))
12
13 (check-sat)
14 (get-objectives)
15 (get-model)
```

Listing 2.5: Maximizing $2x + y$ while satisfiying the constraints of E_1 in SMT-LIB.

```
1  sat
2  (objectives
3   ((+ (* 2.0 x) y) (/ 695.0 96.0))
4  )
5  (
6    (define-fun y () Real
7      (/ 107.0 128.0))
8    (define-fun x () Real
9      (/ 101.0 32.0))
10 )
```

Listing 2.6: A model obtained for the problem in Listing 2.5.

2.2.4 System Modeling and Verification

To model and verify a system using satisfiability checking, one first needs to identify the variables that are necessary for modeling. These are essentially the ones needed in order to reason about the properties of interest. They depend on the system itself, and on the level of abstraction needed. Modeling can use Boolean variables, in which case safety properties (e.g., absence of error) can be expressed. A standard SAT solver (e.g., CryptoMiniSat [Soos et al., 2009]) can then be used to efficiently find a condition on the circuit inputs that satisfies this formula, somehow propagating the property violation condition to check if it is satisfiable.

More generally, system encoding can use a combination of Boolean and numeric variables, in which case an SMT solver is used for satisfiability checking. The encoding specifies relations between these variables in a way that describes an abstract behavior of the system. If $\mathcal{F}$ is the formula encoding the system, and $\mathcal{P}$ is a property that must hold on $\mathcal{F}$, then verifying the property boils down to verifying that there is no model of $\mathcal{F}$ for which $\mathcal{P}$ is violated. In terms of SAT/SMT solving, this is equivalent to checking whether the conjunction $\mathcal{F} \wedge \neg P$ is unsatisfiable. If so, $\mathcal{F}$ verifies $\mathcal{P}$, otherwise (if satisfiable), $\mathcal{F}$ violates $\mathcal{P}$ and the solver yields a counterexample (i.e., a model of the formula $\mathcal{F}$ which violates property $\mathcal{P}$).

In the context of circuit verification in this thesis, the systems at hand are transistor level circuit descriptions, and the properties to verify are the absence of electric rules violations (electrical errors). The formula encoding the circuit is denoted $\mathcal{S}$, for 'semantics'. So, to check whether a circuit is error-free (considering some error $\mathcal{E}$), we use a SAT or SMT solver to check whether the error $\mathcal{E}$ is satisfied by circuit semantics $\mathcal{S}$, i.e., the satisfiability of $\mathcal{S} \wedge \mathcal{E}$ (see Figure 1.1, page 2).

2.2.5 Discussion on Soundness and Completeness

When it comes to verification, strong (i.e., formal) guarantees on the result provided by the verification tool play a major role in determining its *trustworthiness.* In the formal methods community, this is often defined using two complementary concepts: soundness and completeness [Smith, 2010]. In a nutshell, a verification approach that is sound should identify all possible errors (i.e., no error is missed but false alarms may be raised), while a complete approach should identify only errors

that appear in feasible scenarios (i.e., no false alarm can be raised, but some errors may be missed). In other words, sound approaches can only prove properties that are actually true, while complete approaches can prove all true properties. An ideal verification approach should hence be both complete and sound, meaning that the approach should identify real bugs (electrical violations in the context of this thesis), and only those. However, for most interesting analyses, finding exactly the errors is an undecidable problem, and each verification approach can only provide guarantees of either soundness or completeness, while limiting the number of false alarms or missed errors (ideally, no true error must be missed) for the approach to be usable. This, in essence, is what determines the precision of a verification approach. A (sound) verification method is more precise than another (also sound one) if it results in less false alarms.

2.3 Methods Used in this Thesis

The main circuit verification framework adopted in this thesis is the one depicted in Figure 1.1 (page 2). Based on the nature of the formula encoding the circuit (semantics $\mathcal{S}$) in terms of the variables types involved and the theories used, it can invoke a SAT or an SMT solver for satisfiability checking.

SAT solvers have been used in the past for electrical error identification at transistor level, notably by Afonso and Monteiro [2017b]. Their approach consists in building a logical formula from a given circuit netlist, and global conditions for the existence of short-circuits in such netlist. In fact, their approach is limited to mono-supply circuits (which are often logic circuits encoding Boolean functions), and their error conditions encoding is intertwined with the encoding of the circuit behavior. We detail further the approach and its limitations in the next chapter. Generally speaking, a pure SAT solving based verification approach would require more workarounds. This is especially true when the stakes are to handle not only mono-supply circuits, but multi-supply circuits at large. A SAT solver can be deployed in such context, necessarily by bit-blasting actual voltage values. As we previously discussed in Section 2.2.2, bit-blasting is not ideal at all performance-wise.

It is in fact way more simple to use the actual voltage values instead of symbolic ones, in this context. So, we use SMT instead of SAT, to avoid manual bit-blasting of variable values representing electrical quantities (like voltage and current), and, more importantly, to be able to reason about continuous (different size) intervals of values. SMT allows us to directly use the theory of Rationals to encode such values. Also, a model yielded by a solver is hence usable 'as is' without any *decoding* phase required. As for the SMT theories used, we restrict ourselves to the non-linear arithmetic (LRA) theory, and only use quantifier-free (QF) formulas. In fine, our theory decision procedure is the satisfiability of a group of linear constraints—represented with polyhedra. The solution space of one of our SMT formulas is delimited by the corresponding polyhedron. If the polyhedron is empty, then the formula is unsatisfiable.

In relation to previous discussions on reachability analysis versus possibility checking (Section 2.1.1), it is important to note that this work is about the latter. We leverage SMT solving (precisely, the Z3 SMT solver by De Moura and Bjørner [2008]) to check specific properties (e.g., absence of error), looking for the possibility of erroneous steady-states and not their reachability. In a second use-case, we use OMT solving (via the νZ feature of Z3) in the context of reliability analysis, where the goal is to identify the worst-case circuit state in terms of aging.

In the following chapter, we thoroughly study the state of the art in transistor level electric circuit verification, after presenting some background about circuit design, circuit behavior, and electrical errors. A clear panorama of research works should start to appear as we walk the reader through the plethora of existing ERC verification techniques.

3

Transistor Level Electric Circuit Verification

CIRCUIT DESIGN, be it analog or digital, is carried according to a design flow which starts from high-level descriptions and goes all the way down to the physical implementation. It consists of several stages, combining automatic and manual processes, depending on the chips being produced. At each of these stages, bugs of all sorts may be introduced. The mission of verification engineers is to chase hidden errors in the design, and prevent their appearance in the manufactured chip. In many design projects, still, there are errors which ultimately make their way to the final product without being noticed. This chapter provides the basis of understanding of design errors at transistor level, their origins, and their implications. It presents existing verification approaches, weighting, for each, the limitations against the advantages. Most of the contents in this chapter come from our survey paper [Ferres et al., 2025]. The approaches developed within this thesis are then put in perspective with respect to the literature. This should enable the reader to apprehend the larger picture, before going in detail into the contributions of this thesis one chapter away.

3.1 Transistor Level and Circuit Modeling

Errors in a system on a chip (SoC) can be traced down to the physical level. The laws of physics can be used to explain how exactly an error originated and how the underlying circuit state came to exist. Clearly, that is overkill, for what actually matters is to find and exhibit errors in a way that designers can understand and fix.

Very-large-scale integration (VLSI) is the integrated circuit (IC) design process where the density of integration is so high a single chip can support up to billions of transistors [Mead and Conway, 1980]. VLSI designers deal with circuits from much higher levels of abstraction. In fact, depending on their area of specialization, different designers deal with different aspects of the same design. Some of them focus on the functionalities to build, while others dedicate their efforts to extra-functional concerns, e.g., optimizing power distribution or avoiding signal interference.

When it comes to the verification of electric properties, a very low-level of abstraction (like the physical level, or even the layout level) is unnecessarily complex, yet a high-level, like the register-transfer level (RTL), is generally too abstract to reason about electrical properties.

In this work, we aim at verifying electrical properties at transistor level, which is the most suited abstraction level for our needs: it gives us information about power, and tells us how the physical circuit components are connected to each other. In the following section, we see where electrical errors come from, by looking at the whole design flow. Afterwards, we present the terminology associated with transistor level, presenting both syntactical elements and behavioral aspects.

3.1.1 Circuit Design Flow

A typical circuit design process involves multiple people performing complementary tasks. Figure 3.1 depicts an example of a design flow, where the circuit is assembled from a combination of digital and analog circuit cells. First, the specification is partitioned into analog and digital blocks. The RTL description (usually written in VHDL, Verilog, or SystemVerilog), representing the digital part, is automatically synthesized into constituent standard cells (e.g., logic functions, storage functions like D flip-flops, etc.) and converted into a digital layout via a 'place and route' operation. The analog layout on the other hand is directly generated from analog circuit blocks, with a combination of manual and automatic operations.

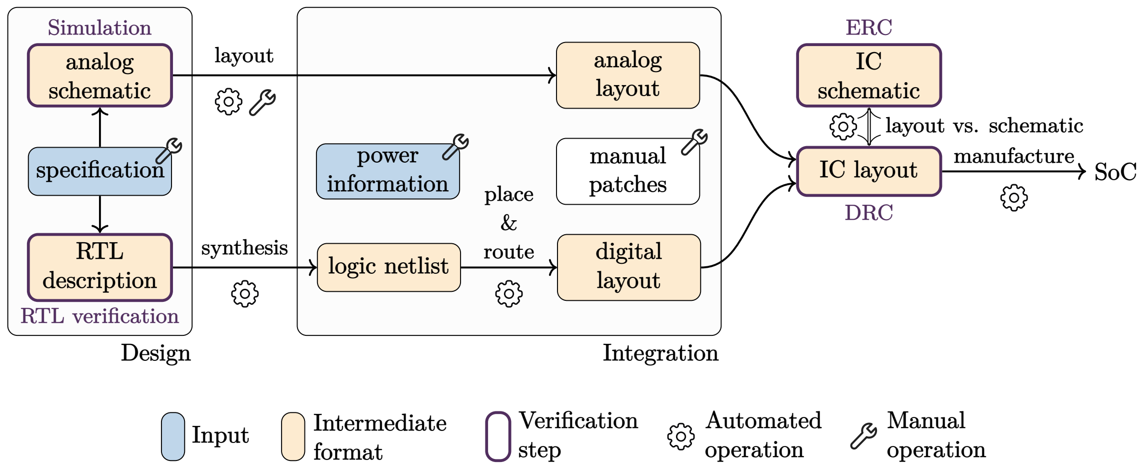


Figure 3.1: A typical design and integration flow.

Modern circuits generally adopt strategies for power management so that energy consumption is reduced. For example, they operate on different modes—among which they can switch. Typically, a 'high-performance mode' leverages the maximum capacity of the circuit, maximizing thus power consumption, while a 'low-power mode' is more frugal in terms of energy consumption and puts less emphasis on performance. Many industrial applications tend to achieve good trade-offs by combining both low-power and high-performance features [Ko et al., 1995; De and Borkar, 1999; Abu-Khater et al., 2002]. Another common technique for reducing power consumption is dynamic voltage and frequency scaling (DVFS). This technique consists in decreasing the frequency of the clocks cadencing digital systems (like processors), by partitioning the system into several voltage/frequency islands. DVFS-based techniques can be useful at times. However, they have shown to be less effective in modern processor designs, leading to minimal gains [Herbert and Marculescu, 2007], or even being counterproductive sometimes [Le Sueur and Heiser, 2010]. In processor design contexts, energy efficiency can also be attained with spacial breakdown of the circuit into several clock domains, in what is called multiple clock domain (MCD) [Semeraro et al., 2002]. MCD leads to a minimization of inter-domain synchronization costs, since synchronization is performed locally instead of relying on a single global clock. In the absence of such division, singly-clocked designs are faced with clock distribution challenges due to wire delays and signal attenuation. Another low-power design choice relies in simply 'turning off' parts of the circuit which are not in use at a given moment. This mechanism is referred to as the 'power-down mode'. For example, the SoCs of our smartphones are equipped with power management units, which ensure that power is not wasted on some subsystem that is not being used [Jian and Xubang, 2008]. One can imagine how complicated it is to keep track of the different possible circuit operating modes. These operating modes can be derived from combinations of all of the power management techniques mentioned. Add to that the complexity in terms of power scenarios, which increases as the number of supplies in a circuit gets higher. Consider a multi-supply circuit composed of a dozen of power domains, assuming each power domain only has a single power-down toggle (which is a rather optimistic assumption). There would be thousands of possible power scenarios to be considered, by only taking account of enabling/disabling the power-down mode on each power domain. Modern designs may even consist of some sort of adaptive power

domains, where supply values are not just set once and for all, but are rather configurable (e.g., on the basis of operation modes). Each of such supplies is allowed multiple values. This, of course, leads to an even more explosive number of possible power scenarios the circuit may operate on.

In a circuit design flow, power information and associated power management constraints are given as input. They are essential for the integration part, as they are taken into account when constructing the analog layout, as well as for optimizing the outcome of placement and routing steps. At the integration level, additional changes may be ported to the design, manually. These changes include, for example, the insertion of protection blocks against electrostatic discharges (ESD), or the modification of the design to either satisfy various constraints or optimize specific aspects. The resulting layout of the whole IC is first checked against design rules. This process is called design rule checking (DRC). It is then checked against the original schematic, in a process named layout-versus-schematic (LVS), before it is sent for manufacturing. The schematic itself is checked against electrical errors, in what is referred to as electrical rule checking (ERC).

Albeit verification occurs at different stages of the flow, the manufactured SoC may still contain errors of different sorts. Electrical errors may originate from manual patches, where the designs are modified or combined to build even larger ones—which tend to be much more difficult to verify. Besides, errors can also occur earlier in the design flow. For example, RTL descriptions may be handwritten, and the analog schematics can be manually created with the help of computer-aided design (CAD) tools.

High abstraction level circuit verification is a well studied topic. It concerns, for example, RTL designs [Augustin et al., 1988; Abrahams and Barkley, 1998; Yadav et al., 2020], or even higher-level descriptions like in transaction level modeling (TLM) [Oliveira et al., 2012; Le et al., 2016]. Correctness aspects have been well tackled with the use of formal methods in arithmetic circuits [Drechsler et al., 2022b,a], sequential circuits [Dominik and Drechsler, 2024], adders [Drechsler, 2021], multipliers [Keim et al., 2003], and more. In some industrial contexts, graph-based analyses have been used to eliminate safe RTL circuit cells from further investigation. This have been applied to both combinatorial [Mongelli et al., 2024] and sequential designs [Mongelli et al., 2025].

These types of verification primarily reason about functional aspects of the design, to check whether the design meets system specification. However, some properties cannot be verified at such high levels, e.g., supply-related properties. A typical example is to check whether a connection between a supply and a ground (i.e., short-circuit) is possible. Power information (i.e., power supplies, voltage values, etc.) is not available at RTL and higher levels. It is only integrated after synthesis, as shown in Figure 3.1. The information required to reason about short-circuits and other similar (electrical) properties is only available at transistor level and lower levels.

Formal verification have been incorporated in transistor level circuit analysis, notably for non-linear analog circuit behavioral analysis, primarily using automata-based models [Tarraf and Hedrich, 2019; Abu-Haeyeh and Hedrich, 2024]. These analyses are concerned with reachability analysis, i.e., verifying whether there exists a sequence of transitions from an initial state leading to an erroneous state. This differs from our intents—that are about satisfiability analysis, i.e., the possibility of a circuit steady-state regardless of whether it can be achieved via a successive number of states.

This thesis targets circuit verification at transistor level, for the purpose of performing ERC. Before going into the details of what such verification means, the kind of errors it addresses, and the methods used to detect them, let us first go through an introduction of the terminology we use, and take a look—via minimal examples—at the way circuits operate.

3.1.2 Transistor Level Terminology and Circuit Behavior

A transistor level circuit schematic is a graphical representation of the circuit netlist. A netlist declares the devices composing the circuit and tells how they are connected, which is pure syntactical information. Figure 3.2 depicts the schematic of an Inverter circuit, whose netlist is shown, in the circuit description language (CDL), in Listing 3.1. Elements of the CDL language are defined in Table 3.1.

We consider transistors of the MOSFET technology. MOSFET stands for metal-oxide semiconductor field-effect transistor. The Inverter of Listing 3.1 is made of two transistors of complementary

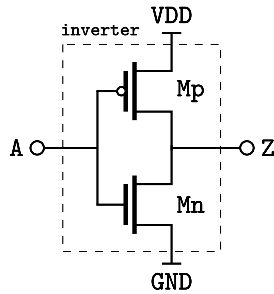


Figure 3.2: Inverter circuit schematic.

```
1  * Inverter
2
3  .SUBCKT inverter A Z VDD GND
4  Mp Z A VDD VDD pmos
5  Mn Z A GND GND nmos
6  .ENDS
```

Listing 3.1: Inverter circuit netlist.

Syntax	Semantics
.SUBCKT <name> [<pin1> [<pin2> ...]]	Header of circuit description with circuit name followed by the list of it pins
M<name> <Nd> <Ng> <Ns> <Nb> <model> [m=<val>]	MOSFET transistor with 'm' parallel devices
R<name> <n1> <n2> <value>	Resistor with resistance 'value'
D<name> <anode> <cathode> <model> [m=<val>] + [n=<val>]	Diode instance with 'm' parallel devices and 'n' series devices
X<name> [<pin1> [<pin2> ...]] <circuit>	Hierarchical instantiation of 'circuit'
.ENDS	Directive which ends a circuit description

Table 3.1: Elements of CDL syntax.

types: Mp of type PMOS (P-channel MOSFET), and Mn of type NMOS (N-channel MOSFET). They have, each, three primary terminal pins that are the *gate*, *source*, and *drain.* A fourth pin, which goes by the *substrate*, *bulk*, or *body*, defines the biasing of the transistor junctions (not shown in the schematic). In practice, the bulk is always connected to a common source of voltage, depending on the transistor type; the bulk of a PMOS is connected to a supply, and that of an NMOS to a ground. We adopt this hypotheses in the entirety of this thesis. Any other cases where the bulk is used differently are not supported. An NMOS device type is turned on when a high voltage is applied on its gate, resulting thus in connecting its source and drain together. A lower voltage closes the connection, and intermediate values result in intermediate degrees of connectivity. The PMOS type is complementary; a high voltage on the gate turns the device off, while a lower voltage turns it on.

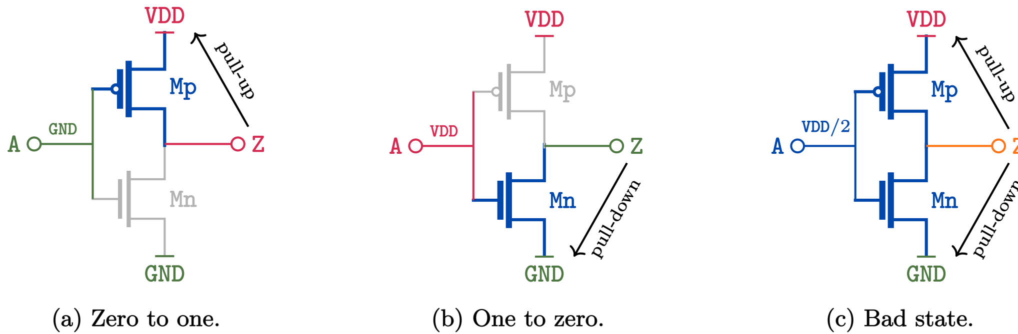
Wires of the circuit are called *nets.* A net is either internal or external. The Inverter circuit example contains no internal nets—all four nets (A, Z, VDD, and GND) being external. Inputs, outputs, supplies, and grounds are all external nets—they are accessible from the environment of the circuit.

Table 3.1 presents a subset of the syntax to describe circuits in CDL. It is limited to the subset of devices considered in this thesis. Angle brackets '<>' are used to indicate mandatory elements, while square brackets '[]' are used to denote optional fields. The technological parameters (e.g., width-to-length ratio, threshold voltage, temperature, resistance value for resistors) are optional in the CDL syntax. They are usually obtained from dedicated files given alongside circuit descriptions. We omit those from the syntax description in Table 3.1. We only keep optional parameters related to specific device instances. Note that the '+' symbol can be used at the beginning of a line as indicator of the continuation of the previous line. Also note that, for hierarchical instantiations, the name of the circuit being instantiated is added as the module name for the 'X' device.

The netlist (or the schematic) alone is not sufficient to determine the behavior of the circuit. To reason about the functioning of the Inverter, we need to consider concrete voltage values for some of the external nets. Let $\mathcal{V}$ be the function which associates to each net in the circuit a voltage value. We set the circuit's supply (VDD) voltage to 1 V and the ground's (GND) to 0 V. One may think of 1 V as the logical '1' and of 0 V as the logical '0'. Figure 3.3 enumerates the states of the

circuit based on the value of $\mathcal{V}(\texttt{A})$ chosen among $\mathcal{V}(\texttt{VDD})$ and $\mathcal{V}(\texttt{GND})$, or anything-in-between. The assumption that an input is binary is based on a digital abstraction of the behavior of the circuit. Functionally speaking, the Inverter only makes sense in two states: a 0 V on the input `A` turns `Mn` off and turns `Mp` on, pulling thus the output `Z` up to 1 V (the voltage of `VDD`)—see Figure 3.3a. Setting `A` to 1 V results in a complementary state (b). When a continuous range of input voltages are taken into account, we obtain a third state: some voltage values can turn both `Mp` and `Mn` on, resulting thus in the output `Z` being both pulled-up to the supply and pulled-down to the ground, which, functionally speaking, is a *bad* state (c). The functional problem of state (c) is in fact caused by the short-circuit created on the output (`Z`), which, in this context, is an electrical error, caused by the input (`A`).

Note. In Figure 3.3 (and all other circuit states depiction figures in this document), a **thick blue** line is used to draw switched-on devices, and a light gray line is used to draw switched-off devices. Wires are colored based on their voltage. The name of the supply a wire is connected to is either indicated with small letters on the wire, or indicated with a pull-up/pull-down labeled arrow. In general, green is used for the ground, blue and red are used for supplies, orange is used for short-circuits, and light gray is used for floating nets. When the figure simply depicts the circuit's schematic, regardless of the circuit's state, the whole drawing is done in black.



(a) Zero to one. (b) One to zero. (c) Bad state.

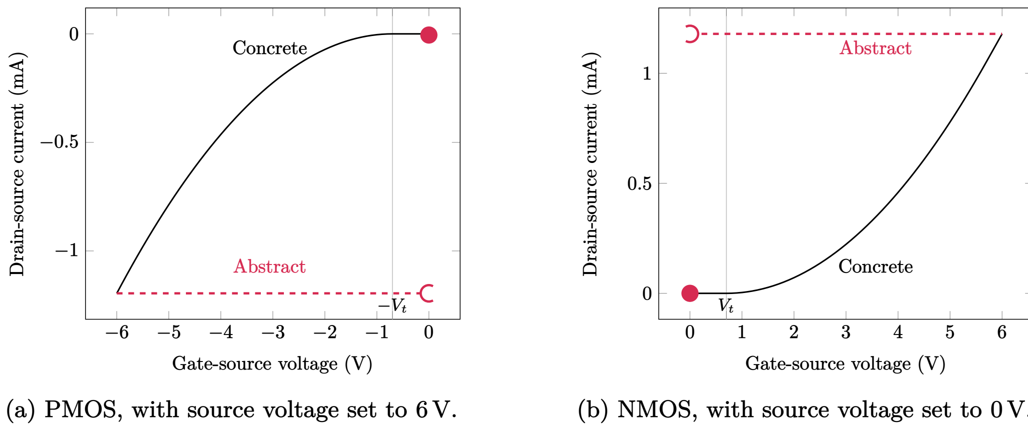
Figure 3.3: Three states of the Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$.

Considering transistors as ideal switches is referred to as the switch-based abstraction. It is widely adopted in the modeling of digital circuits [Randal E Bryant, 1984; Zwerger and Graeb, 2012], but it is neither suited for analog circuit modeling, nor for any circuit where the context of voltages is beyond binary.

Furthermore, the concrete transistor behavior is continuous and depends on a threshold voltage, denoted V_t, for its activation. Only circuit simulator engines provide an approximation of the concrete behavior based on transistor's characteristics equations. Figure 3.4 shows the difference between a simulation-approximated concrete current-voltage relationship and the switch-based abstraction, for PMOS and NMOS devices.

On the basis of the switch-based abstraction, if we consider the Inverter circuit, all values $\mathcal{V}(\texttt{A}) \leq \mathcal{V}(\texttt{GND}) + V_t$ lead to state (a) of Figure 3.3, all values $\mathcal{V}(\texttt{A}) \geq \mathcal{V}(\texttt{VDD}) - V_t$ lead to state (b), while state (c) corresponds to all values $\mathcal{V}(\texttt{A}) \in]\mathcal{V}(\texttt{GND}) + V_t, \mathcal{V}(\texttt{VDD}) - V_t[$.

The switch-based abstraction is particularly useful when reasoning about circuits built using Complementary MOSFET (CMOS). CMOS is a low-power design method widely used to implement logic functions (e.g., the Inverter circuit of Figure 3.2) [Semiconductor, 1983; Lancaster and Berlin, 1988]. In theory, CMOS designs use no power when the inputs are unchanged. Power may thus dissipate only during transitions, i.e., when the inputs change. In CMOS, a circuit consists of (1) a pull-up network through which the output pin can reach the primary circuit supply (i.e., logical '1'), and (2) a pull-down network through which it can reach the ground (i.e., logical '0'). The pull-up network is made of PMOS devices. It builds the circuitry which tells whether the Boolean function being implemented evaluates to *true*. On the other hand, NMOS devices make the pull-down network, encoding thus the negation of the function, i.e., the circuitry which produces *false*



(a) PMOS, with source voltage set to 6 V.

(b) NMOS, with source voltage set to 0 V.

Figure 3.4: Transistor characteristics curves obtained from simulation, and their switch-based abstraction. With $V_t = 0.7\,\mathrm{V}$.

as output (see the Inverter circuit in Figure 3.2). Certainly, not all designs are CMOS-based, yet it is important to be able to properly model their behavior for the purpose of verifying them.

The ultimate way to describe the behavior of a circuit is achieved with the laws of physics. Still, what the physics tells us is way beyond the level of detail needed for the analysis of most electrical properties. Nonetheless, the underlying physical principles of transient circuit behavior can be modeled using systems of non-linear partial differential equations [Bank et al., 1985; Nichols et al., 1994; Lang, 2010]. Such systems of equations can be solved via numerical methods implemented in simulation engines like Simulation Program with Integrated Circuit Emphasis (SPICE) [Nagel and Pederson, 1973]. We discuss more about the use of simulation for circuit verification in Section 3.3. For now, we take a closer look at the so-called electrical errors, and the circuit aspects one needs to model in order to reason about them.

3.1.3 Transistor Level Circuit Modeling

Electrical properties differ from other sorts of properties (e.g., functional properties) in that they reason about the way circuit components are used and about their connectivity in each circuit steady-state (as opposed to transient responses which characterize the transition from one circuit's state to another). A circuit steady-state may either be stable (i.e., an equilibrium point the circuit stabilizes at) or not. Such reasoning—from a transistor level standpoint—requires knowledge about three main elements: (1) circuit environment, (2) devices' states, and (3) nets' states.

3.1.3.1 Circuit Environment

Part of the assumptions considered when modeling circuits at transistor-level are related to the voltage values of supplies and inputs. In fact, even identifying who's who among the circuit pins may be tricky, mainly because the definition of inputs and outputs largely depends on the intent of use. Sometimes, a pin can be used both as input and output.

In software engineering, and particularly in functional programming, functions can be mathematically *pure*—in the sense that function arguments (i.e., inputs) are only used by the function to produce some output without changing them. A pure function, written—for example—in OCaml or Haskell functional programming languages, have its arguments well-defined as either input or output; it has no such thing as input-output argument. A program is then constructed by composing functions—just as we do in math. If a function happens to *change* one of its arguments, that is a side effect—which functional programming aims at avoiding. There are, of course, programming paradigms where it is completely fine (or even desired) to have side-effects. We can even say that the overwhelming majority of real-world software do not follow the functional paradigm. This is valid for rapid prototyping and scripting languages, like BASH and Python, but also for low-level high-performance languages, like C and C++, all belonging to the family of imperative languages.

A hardware design can also implement functions taking inputs and producing outputs. The main difference, however, is that the hardware often creates relations between inputs, outputs, and supplies. Information flow does not necessarily happen from input to output, and even the distinction between input and output is not always possible.

The notion of *typing* we can find in programming consists in checking the consistency of the data types the program manipulates. It is very helpful for limiting the number of bugs the program may have, since the programmer is aware of any typing errors at compile time. Unlike in software, there is no well established form of typing in the hardware realm, namely for transistor level designs. Initiatives which promote correct-by-construction methods for the design of hardware systems are relatively rare, and they remain mostly targeted at higher level (digital) hardware design. Early developments in this area led to the μFP language for VLSI design [Sheeran, 1984, 2005]. It was greatly inspired by the works of Backus [1978, 1981] introducing the FP functional language. μFP enables the programmer to reason about circuits in terms of functions, while allowing both manipulation of circuit behavioral specification and layout description. CHISEL is a library of the SCALA programming language, used to construct hardware systems. It relies on both object-oriented and functional paradigms. Unlike traditional hardware description languages (like VHDL and VERILOG), CHISEL provides powerful abstraction facilities which make the task of components reuse easier, via, e.g., parametrization of hardware blocks [Bachrach et al., 2012]. More recently, a design framework named HARDCAML is introduced [Ray et al., 2023]. It leverages the richness of the type system of the OCAML programming language, along with custom circuit checks, in order to lower the chances of user-introduced bugs in RTL hardware designs.

In Chapter 4, we walk the reader through the challenges related to defining transistor level circuit contexts of use. We provide more details on our formalization of circuit environment in Section 4.2.2.

3.1.3.2 Devices' States

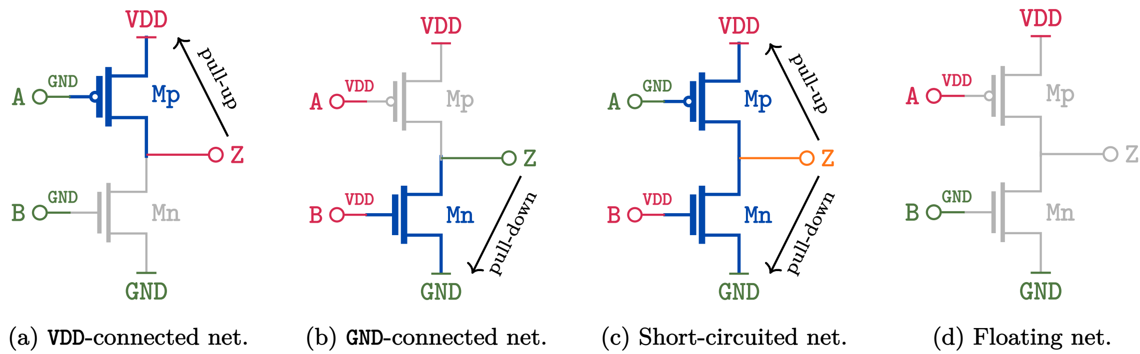
A device state is defined by its surrounding voltages of gate, source, and drain nets. We use the switch-based abstraction to illustrate the behavior of circuits. This assumes a device is either on or off. It may seem too abstract with respect to the concrete model approximated with simulation (Figure 3.4). Yet, switch-based abstraction has shown to be useful in digital circuit modeling [Randal E Bryant, 1984; Zwerger and Graeb, 2012]. Some variants of our abstractions are based on it, while for some others we use more refined modeling based on the actual device characteristics curves. The abstractions in this thesis are developed with keeping in mind scalability matters, the goal being to find a middle ground of scalability versus precision. All device modeling variants are defined as part of circuit semantics, in Chapter 4.

3.1.3.3 Nets' States

In switch-based abstraction, a net can be seen as either *consistently* connected to a single voltage or not. If not, it is either in short-circuit (i.e., connected to at least two supplies) or floating (also referred to as high-impedance or high-Z state, i.e., connected to no supply). Figure 3.5 illustrates the different possible states for the output `Z` of a two-input Inverter circuit. Two-input Inverters may typically be used in the design of some Latches [Dai et al., 2022]. Beyond that, they seldom serve any functional purpose, especially when considered as standalone circuits. Our use of such circuit, and similar simple examples to come, is primarily done for illustration reasons; some ideas are better explained on minimal unrealistic examples rather than realistic large ones.

As for our two-input Inverter, when the output (net `Z`) has a connection to a single supply, its voltage value is the same as the supply it is connected to (see states (a) and (b) of Figure 3.5). When it is short-circuited, its actual voltage value is comprised between the range defined by the minimum and maximum voltages it is connected to (c). In case of a floating net, the voltage is non-deterministic, but a range of possible voltages can be defined from the supplies the net can statically reach—which are `GND` and `VDD` in our example (d).

A property of a net can, for example, be that it is never in short-circuit. Such property is relevant, for example, for the inputs and outputs of CMOS-based logic circuits. However, there are cases where a short-circuit is a valid behavior of the net. This is particularly true in analog



(a) VDD-connected net. (b) GND-connected net. (c) Short-circuited net. (d) Floating net.

Figure 3.5: Circuit states of the two-input Inverter for different states of net Z. With $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{VDD})$.

circuits which manipulate current, like current mirrors—which use current replication mechanisms to provide bias current or model ideal current generators. Figure 3.6 depicts an example current mirror topology. It uses a diode-connected NMOS device (M1) in *saturation* mode. The resistance value of resistor R1 can be tuned to determine the value of the input current I_{REF}, which is then either copied to the mirrored branch as I_{OUT}, when the device geometry of M2 is identical to that of M1, or multiplied by some gain otherwise [Sansen, 2007].

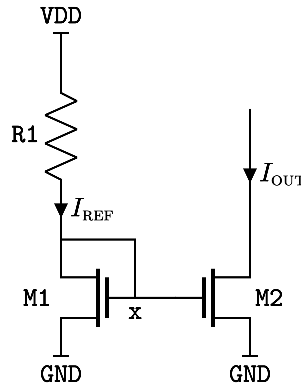


Figure 3.6: Current-mirror schematic.

We go into the details of modeling these three aspects (circuit environment, devices' states, and nets' states) in Chapter 4. It is now time for a brief tour of electrical errors. The following section provides a grasp of electrical properties verification and related challenges, through motivating examples.

3.2 Electrical Errors

There are several stages of the circuit design flow where electrical errors might be induced. We discussed some of these reasons in Section 3.1.1. In this section, we describe different sorts of electrical errors. Afterwards, we thoroughly present the literature review of related verification techniques.

3.2.1 Floating Gates

As shown in the example of Figure 3.5d, a floating net may lead to a functionally erroneous behavior, and would create undesired current leakage when it is used as input of some digital circuit. Sometimes, however, floating gates are desired, e.g., for building long-term nonvolatile information

storage devices [Hasler et al., 1999; Hasler and Lande, 2001]. It is, therefore, the circuit use cases which determine whether we are dealing with a design error or not.

3.2.2 Electrical Overstress

A device is said to be in electrical overstress (EOS) when it is applied an electrical stress (either gate-to-source or gate-to-drain voltage) which exceeds its specified absolute maximum ratings, causing thus the device to fail [Kaschani, KT, 2015]. Overall, over-stressing devices contributes to accelerate their aging [Saha et al., 2011].

Note. A transistor's source and drain terminals are physically symmetrical, and are, thus, interchangeable. This is taken into account in the condition for EOS.

3.2.3 Missing Level-Shifter

Consider a potentially buggy version of the buffer circuit depicted in Figure 3.7, with its netlist shown in Listing 3.2. The schematic is given *flat*, but can also be described in a hierarchical manner, where the Inverter is defined once, then instantiated twice within the buffer. It could as well be implemented using any chain of an even number of Inverters. Buffers are used, in practice, to ensure signal integrity and avoid its attenuation (e.g., when the signal needs to travel far away in the circuit), or as part of synchronizing circuitry.

Our buffer example consists of two Inverters connected in series. Nets `GND`, `VDDL`, and `VDDH` are, respectively, the ground, the first Inverter's supply, and the second Inverter's supply. We assume, for this example, that $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{VDDL}) < \mathcal{V}(\texttt{VDDH})$. Typically, $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$. `M1` and `M3` are PMOS devices, and `M2` and `M4` are NMOS devices. The first Inverter (made of `M1` and `M2`) either (1) takes its input `A` as `GND` which pulls-up `x` to `VDDL` through `M1` or (2) takes `A` as either `VDDL` or `VDDH` and results in `x` being pulled-down to the ground through `M2`. The second Inverter (made of `M3` and `M4`) in turn *tries* to invert the value of the internal net `x` and assign it to the output `Z`. However, only $\mathcal{V}(\texttt{x}) = \mathcal{V}(\texttt{GND})$ is properly inverted into $\mathcal{V}(\texttt{Z}) = \mathcal{V}(\texttt{VDDH})$, because with $\mathcal{V}(\texttt{x}) = \mathcal{V}(\texttt{VDDL})$, both `M3` and `M4` are switched-on (according to the switch-based abstraction), causing a short-circuit on the output `Z`. When net `Z` is short-circuited, its exact value depends on the exact values of $\mathcal{V}(\texttt{VDDL})$ and $\mathcal{V}(\texttt{VDDH})$. Chances are that `Z` is pulled-down to `GND` more than it is pulled-up to `VDDH`, as `VDDL` gets closer to `VDDH` than it is to `GND`. Figure 3.8 summarizes the corresponding possible states of the buffer circuit.

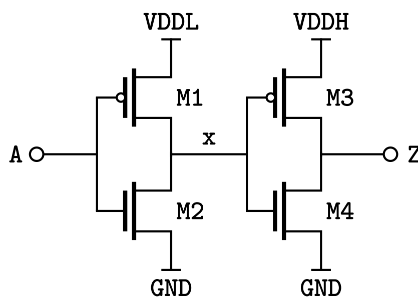
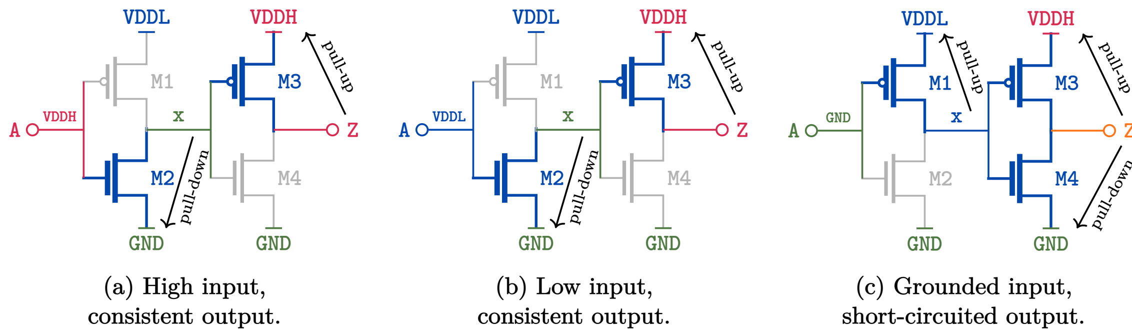


Figure 3.7: One-stage buffer circuit schematic.

```
1  * Buffer
2
3  .SUBCKT buffer A Z VDDL VDDH GND
4  M1 x A VDDL VDDL pmos
5  M2 x A GND GND nmos
6  M3 Z x VDDH VDDH pmos
7  M4 Z x GND GND nmos
8  .ENDS
```

Listing 3.2: Buffer circuit netlist.

Circuit state (c)—Figure 3.8 is erroneous for it generates a short-circuited output, which, when used to drive some other circuitry (which is what the output of a buffer is usually used for), may lead to functional and extra-functional errors. For example, a short-circuit happening on net `Z` means that current leaks through devices `M3` and `M4`, which contributes to their degradation. This also violates CMOS design rules. In fact, there misses a piece of circuitry in this example: a *level-shifter* to properly connect the two parts of the buffer. The configuration of supplies makes it that devices `M3` and `M4` are subject to conflicting power domains, as they can statically reach both `VDDH` and `VDDL` (through device `M1`, which belongs to the `VDDL`-domain, and whose drain *drives* `M3` and `M4` gates).



(a) High input, consistent output.

(b) Low input, consistent output.

(c) Grounded input, short-circuited output.

Figure 3.8: Buffer states for different input (`A`) values. With $\mathcal{V}$(`GND`) < $\mathcal{V}$(`VDDL`) < $\mathcal{V}$(`VDDH`).

A missing level-shifter describes a circuit state where part of a circuit receives a signal beyond its power domain. This is exactly the case when the internal net `x` gets connected to `VDDL`, causing thus both `M3` and `M4` to be switched-on, resulting in the output `Z` to be short-circuited. To fix this, one must include a level-shifter cell between the two inverters composing the buffer, as illustrated in Figure 3.9. Its netlist is described hierarchically—instantiating a level-shifter—in Listing 3.3. An example level-shifter is shown in Figure 3.10, with its netlist shown in Listing 3.4. For ease of reading, pins of the instantiated circuit are named, on purpose, after the net they are connected to in the instantiating circuit.

Reminder. In CDL syntax, a line prefixed by the '`+`' symbol is a continuation of the previous line.

The level-shifter produces the *right* voltage for `x2` based on $\mathcal{V}$(`x1`), in the context of a signal originating from the `VDDL`-domain and entering the `VDDH`-domain. Figure 3.11 illustrates the different steady-states of the level-shifter circuit: a low voltage is converted into a high voltage—see state (c), whereas the ground or a high voltage are preserved at the output—see states (a) and (b). This prevents net `x` of the buffer circuit from having a low voltage (that of `VDDL`), hence prevents any current leakage on the output `Z` (Figure 3.8c).

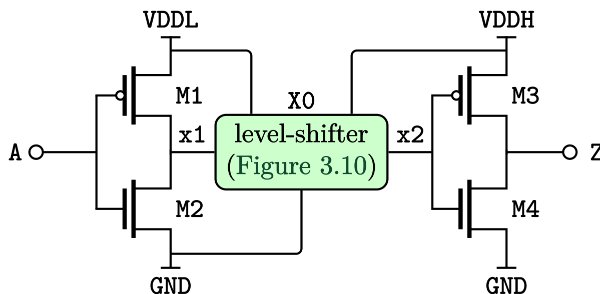


Figure 3.9: A fix of the buffer circuit of Figure 3.7 using a level-shifter.

```
1  * Buffer
2
3  .SUBCKT buffer A Z
4  + VDDL VDDH GND
5
6  M1 x1 A VDDL VDDL pmos
7  M2 x1 A GND GND nmos
8  M3 Z x2 VDDH VDDH pmos
9  M4 Z x2 GND GND nmos
10
11 X0 x1 x2 VDDL VDDH GND
12 + level_shifter
13 .ENDS
```

Listing 3.3: Fixed buffer circuit netlist. The `level_shifter` module is described in Listing 3.4.

Intersection with Clock Domain Crossing. In large synchronous SoCs involving multiple processors, data can be passed across different clock domains. Such systems are said to be multi-clocked, and the transfer of data between clock domains in referred to as clock domain crossing (CDC). CDC configurations in multi-clocked designs can be subject to *glitches*. They may occur

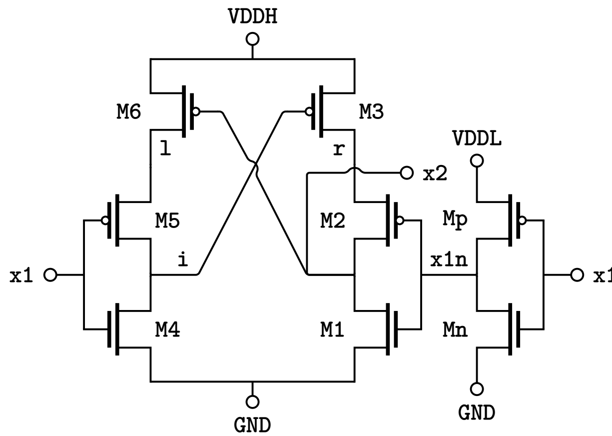
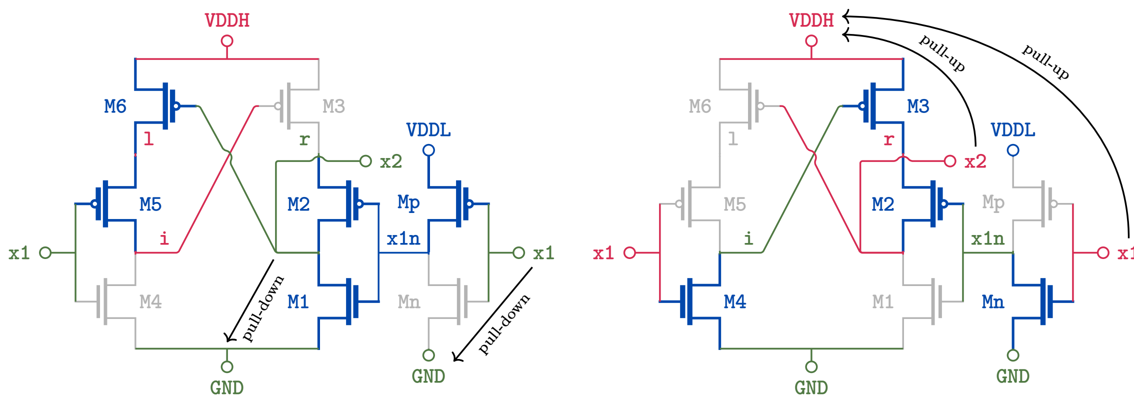


Figure 3.10: An example of a level-shifter to be used in Figure 3.9.

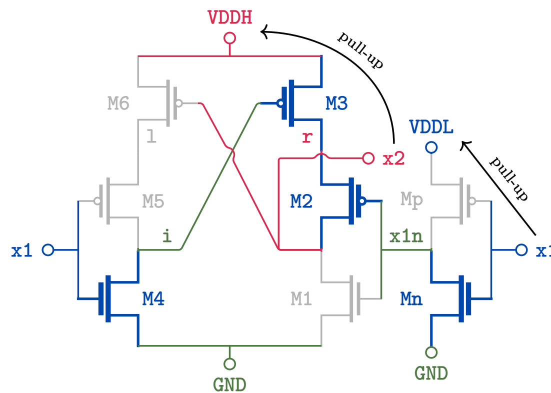
```
1   * Level-shifter
2
3   .SUBCKT level_shifter x1 x2
4   + VDDL VDDH GND
5
6   M1 x2 xin GND GND nmos
7   M2 x2 xin r VDDH pmos
8   M3 r i VDDH VDDH pmos
9   M4 i x1 GND GND nmos
10  M5 i x2 l VDDH pmos
11  M6 l r VDDH VDDH pmos
12  Mp xin x1 VDDL VDDL pmos
13  Mn xin x1 GND GND nmos
14  .ENDS
```

Listing 3.4: Level-shifter circuit netlist.



(a) Preservation of the ground voltage: $\mathcal{V}(\texttt{x1}) = \mathcal{V}(\texttt{x2}) = \mathcal{V}(\texttt{GND})$.

(b) Preservation of a high voltage: $\mathcal{V}(\texttt{x1}) = \mathcal{V}(\texttt{x2}) = \mathcal{V}(\texttt{VDDH})$.



(c) Conversion of a low voltage $\mathcal{V}(\texttt{x1}) = \mathcal{V}(\texttt{VDDL})$ to a high voltage $\mathcal{V}(\texttt{x2}) = \mathcal{V}(\texttt{VDDH})$.

Figure 3.11: Level-shifter states.

when a sub-system cadenced at some clock samples some input coming from another sub-system that is cadenced at a different clock. The glitch precisely happens when the sampling (i.e., rising-edge of the clock) happens as the wire being sampled changes its value.

On top of the different clock domains operating at different frequencies, any additional (asynchronous) circuitry contributes to delaying data transfer. In fact, a level-shifter connecting two power domains, which also happen to correspond to two clock domains, is an example of a circuit

which induces such latency. Its presence must thus be taken into account in synchronizing signals. Semi-formal and structural verification techniques have been used to tackle design errors due to CDC [Kalel, 2024]. Approaches using a combination of counterexample-guided abstraction refinement (CEGAR) and user-aided abstraction refinement (UsAAR) are shown to be effective for CDC verification [Plassan et al., 2016, 2017]. However, timing analysis considerations are out of scope of this thesis, and are thus not considered. Still, synchronizing elements between clock domains can be seen as somewhat analogous to level-shifters between power domains.

Multi-clock designs are often concerned with multi-supply constraints as well. In this thesis, however, we do not study clocking related problems, including CDC verification. The only properties we address are those which reason about circuit steady-states.

3.2.4 Current Leakage

Current leakage happens when there is a continuous electrical path connecting two (or more) different supplies. Nets on such path are all in a short-circuit state. This may be caused by another error, like a missing level-shifter between two power domains, illustrated on Figure 3.8c. However, in some contexts current leakage is legitimate, e.g., to propagate information using current instead of voltage. Although circuits which use current as the basis for information propagation consume more power compared to using voltages, they are more precise in terms of information transfer, especially in applications which demand high precision [Guggenbuhl et al., 1994; Fish et al., 2005]. So, simply checking the complete absence of current leakage in a circuit is not always relevant; properties of interest are generally more specific than that. For example, one could prove that no current leakage is possible at the interface between power-domains.

3.2.5 Electrostatic Discharge

An electrostatic discharge (ESD) is an external event to the circuit, where undesired current flows in the devices. This can happen when an excessive voltage is applied on the pads of the circuits (i.e., top-level circuit pins), and may cause damages if not properly handled. An ESD event is typically caused by a human touching a device, or can be field-induced—when a device builds up internal voltage and releases it in the circuit. The susceptibility of a device to get damage from EDS is commonly modeled either with the *human-body model* (HBM) [ANSI/ESDA/JEDEC, 2012] or with the *charged-device model* (CDM) [ANSI/ESDA/JEDEC, 2022].

In addition, ESD events are becoming even more critical with the development of new manufacturing technologies, as reducing the size of transistors also reduces the breakdown voltage for ESD [Liu et al., 2008]. Specific protections must be introduced in the circuit to prevent induced damages, should an ESD event occur. It is thus necessary to check that critical parts of the circuits are protected against ESD, to ensure their robustness [Lu and Bell, 2010; Viale and Allard, 2020]. ESD checks may be performed at various stages of an IC design flow—from cell level to full-chip level [Khazhinsky et al., 2012]: some of these checks require considering very low-level details of the circuit (e.g., physical device characteristics), while others might only consider the topology of the circuits under verification.

3.3 State of the Art in Electrical Rule Checking at Transistor Level

Verifying steady-state circuit properties at transistor level is, in essence, to model circuit behavior to a degree of detail which allows reasoning about the properties of interest. In this section, we explore various verification techniques and the underlying circuit models. We then discuss the pros and cons of each approach. Finally, we present an overview of our contributions, and position them with respect to the state of the art. Further details on contributions are presented in upcoming chapters. The classification of the verification methods presented hereafter is based on the taxonomy summarized in Figure 3.12, which is based on the modeling techniques used for ERC.

Simulation engines, like SPICE, are industry-standard tools for analysing how a circuit behaves against a set of input vectors. It can be used for ERC either in a *naive* way, by simulating every single input vector (which is obviously explosive in terms of execution time), or as a complement

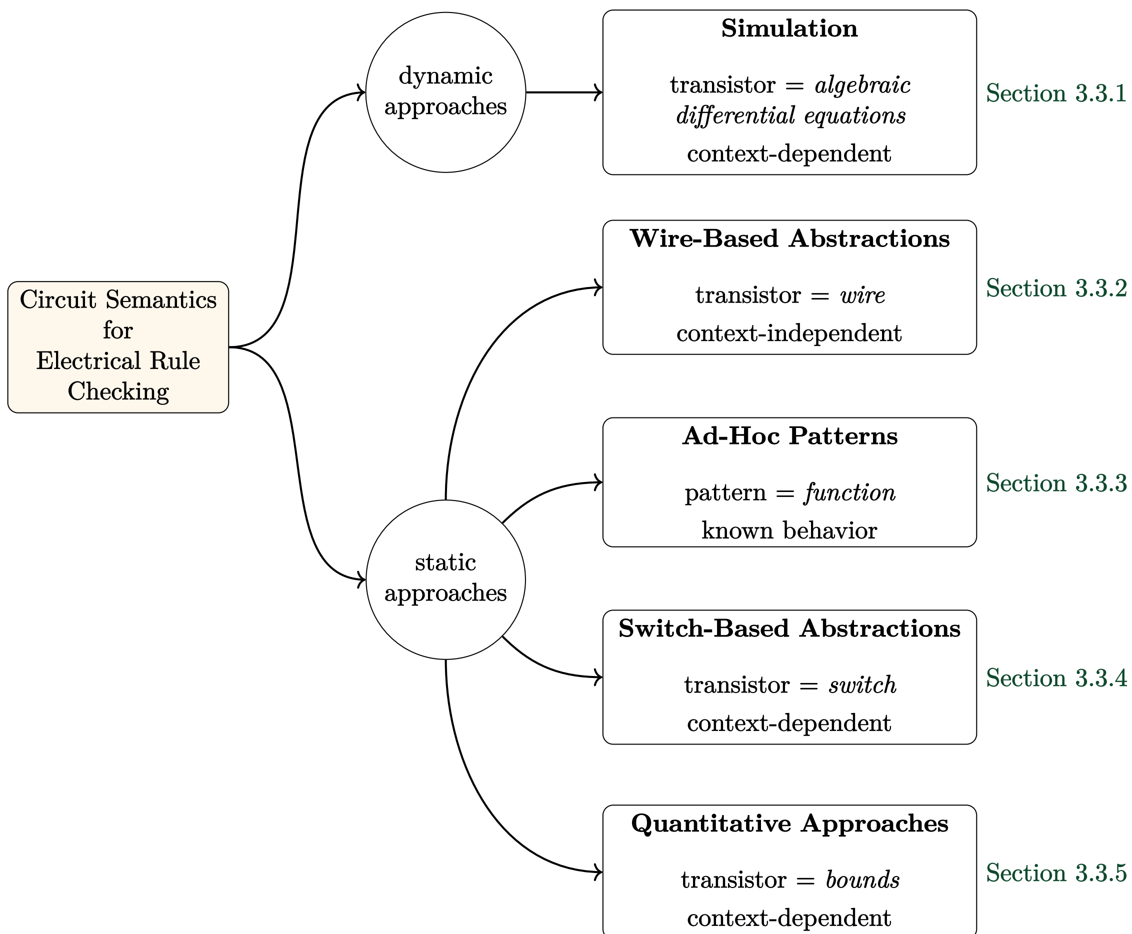


Figure 3.12: Taxonomy for transistor level electrical rule checking approaches.

to other (static) verification approaches—e.g., to double-check whether some presumable erroneous state is actually feasible.

Wire-Based abstractions model transistors as wires: the analysis considers that current may flow between drain and source independently of the voltage applied on the gate terminal. This is a coarse approximation of the circuits concrete behavior, yet its simplicity has benefits for ERC in terms of scalability.

Ad-hoc patterns based approaches aim to identify, in a given circuit, some structures with a specific behavior (e.g., *correct* interfaces between power domains). Instead of modeling the behavior of each component in the circuit, these approaches only focus on some identified structures and thus only model parts of the circuit. They can be used for specific behaviors which require details on the low-level implementation and can not be modeled precisely by combining the individual behaviors of the constituting components.

Switch-based abstractions model the switching behavior of transistors, making it possible to model more precisely (compared with wire-based abstractions and ad-hoc patterns) the voltages at each net of a circuit. In these approaches, a transistor is modeled either as a wire (when it is switched-on) or as an open circuit (when switched-off) depending on the voltage applied on its gate.

Quantitative approaches use approximations of device characteristics equations as a means to over-approximate their operation bounds in terms of current and voltage relations. Such relations are known as Intensity-Voltage (I-V) characteristics. A verification method modeling device behavior using such approximation is able to determine the circuit states where some device's specification, in terms of I-V relations, is violated (and may yield false alarms as well).

3.3.1 Simulation

The next best way to precisely learn how a circuit behaves in real-life, other than actually observing its behavior in real-life, is to run it in simulation. The more accurate the models used in the simulation tool with respect to the physics are, the better. SPICE simulators have different device models' levels, based on the physical details they capture [Synopsys, 2013], from Level 1 (less detailed) to Level 3 (most detailed). The use of simulation in transistor level verification mostly consists in exploring combinations of the input vector, and checking for possible violations of some electrical rules.

Because it is very precise, transistor level simulation requires large computational resources, which makes it difficult to handle large designs [Lang, 2010]. Not only does SPICE use heavyweight mathematical models including differential algebraic equations [Nichols et al., 1994], but each simulation is done for a given combination of input voltages. An exhaustive simulation requires to consider a number of input vectors that grows exponentially with the input size. Moreover, if one wants to fully explore combinations of input values that varies in continuous intervals, the number of combinations becomes intractable. That is practically impossible to simulate. Large circuits cannot be verified exhaustively in reasonable time, hence simulation provides no guarantee that all errors are detected. In addition, simulation cannot easily explore the state space of memory-dependent circuit states. Indeed, the internal state of a circuit with memory does not only depend on the input vector, but also depends on the previous state points, hence a straightforward exhaustive enumeration of inputs does not explore all internal states.

3.3.2 Wire-Based Approaches

Wire-based approaches are verification strategies which consist in searching for potential errors. They use some of the so-called (static) voltage propagation algorithms, which is the abstraction of all transistors being potentially switched-on regardless of the voltage applied on their gate, i.e., a transistor is modeled as a wire between the source and drain terminals (the gate remains disconnected from source and drain). It captures all reachable scenarios, plus, possibly, nonsensical scenarios as well. Note that there exist other approaches which use voltage propagation algorithms with different semantics, taking thus into account the on/off states of transistors. We discuss them more, later, in Section 3.3.4. Despite being naive, wire-based abstractions provide sound results, but have the flaw of resulting in a lot of false alarms; some unfeasible circuit states may arise given that the considered electrical states of a circuit are a coarse over-approximation of the actually reachable states. For example, the wire-based abstraction of an Inverter circuit considers that it may connect the supply `VDD` to the ground `GND` (Figure 3.13). The actual circuit, however, contains two transistors in series that cannot be switched-on at the same time under the hypothesis that $\mathcal{V}(\mathtt{A}) \in \{\mathcal{V}(\mathtt{GND}), \mathcal{V}(\mathtt{VDD})\}$.

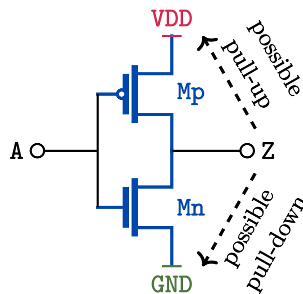


Figure 3.13: Wire-based abstraction of an Inverter.

Performing an analysis with wire-based abstractions is often called voltage propagation (or supply propagation), and mainly relies on traversing a graph representation of the circuit to annotate every net with the possible supplies that can reach it [Lescot et al., 2012; Zwerger and Graeb, 2014]. If a given supply can reach the source (respectively, the drain) of a transistor, then voltage propagation considers that it can also reach its drain (respectively, the source). After a static

voltage propagation step is performed, basic ERC analysis can be performed by traversing the annotated graph and use voltage propagation information to check specific properties—mostly the ones which speak about the connectivity of nets, e.g., short-circuits. Note that as the approach over-approximates the reachable supplies, it is sound (while coarse) on short-circuits, but may result in false negatives for floating nets (i.e., concluding a net is not floating while it is). In practice, wire-based abstractions are used as a fast and reliable first filter to eliminate parts of the circuit where no error can occur. In other words, if such a coarse abstraction finds no possible connection between two given nets, then we know for sure that no such connection will happen in real-life whatsoever.

Examples of use of wire-based abstractions are numerous. Hogan et al. [2013] use it in one of their analysis modes, namely the *vectorless* mode—where no input vector is provided to the tool. They demonstrate the usability of a wire-based abstraction for EOS identification.

Srinivasan et al. [2014] demonstrate how a property can be propagated throughout all resistors in a hierarchical SoC, without flattening its structure thanks to a symbolic algorithm. The kind of properties they study are path-dependent, such as point-to-point resistance in a circuit. They model resistors as wires, which makes their approach properly fit in the category of wire-based ERC approaches, and which they demonstrate for the identification of the paths possibly subject to ESD.

Several techniques rely on custom type systems to propagate more information on nets' connectivity, such as stating if the net is a drain, source and/or gate [Lu and Bell, 2010; Viale and Allard, 2020]. In particular, it can be used to identify power domain interfaces, where specific design rules must be used to protect the circuits (notably against ESD events) or ensure a correct behavior (e.g., with level-shifters). Such interfaces can be identified using a type system, with the following rule: if a transistor's source and drain have some voltage v_1, and its gate has some voltage v_2, then this transistor is part of a power domain crossing interface. For example, Viale and Allard [2020] use a specific net type propagation algorithm as a prerequisite for further analysis of ESD paths. They use static voltage propagation to initialize the nets in their representation of the circuit with specific types and voltage labels, which can then be used for further processing. Custom type systems can be used for other checks, as the nature of a net (i.e., source, gate, etc.) has to be considered in error scenarios. For example, if a net is not a gate, it can not be subject to a floating gate error, by definition.

Another example of wire-based abstractions approach can be found within the graph-based approach for ESD vulnerability identification from Liu et al. [2008]. In this approach, hierarchical circuits are modeled as weighted graphs, where weights are used to define current paths between Input/Output (I/O) pads, which may lead to ESD vulnerabilities. The proposed algorithm can hence be used to identify the pad-to-pad paths to protect against ESD events, considering that only a small fraction of pad-to-pad paths are vulnerable to such events (i.e., one only needs to protect paths where there are few transistors, as the other paths are not vulnerable to ESD events due to their higher resistance).

Lescot et al. [2012] propose an integrated solution to identify various electrical errors, namely floating transistor gates, over-voltage on thin-oxide transistors and missing level-shifters. Their approach is based on supply detection—using heuristics on bulk-connections to identify the supplies at the different levels of hierarchy—and static voltage propagation, demonstrating that static voltage propagation can be useful in many use cases.

Performing static voltage propagation on industrial-scale circuits—which may include up to billions of transistors—is a computationally expansive task. Integrating such approach in standard verification toolchains requires an optimized implementation favoring scalability. Calibre PERC [Siemens EDA, 2023] is an example industrial tool which does that, by combining supply propagation [Gibson et al., 2010; Lescot et al., 2012; Kollu et al., 2012] with pattern recognition features (which we introduce next in Section 3.3.3). Calibre PERC is designed to handle supply propagation in a hierarchical way, which helps boost its scaling. The abstraction considered in wire-based abstractions approaches is rarely used as a standalone method, for it is naive—hence false alarm prone. Instead, it is generally used as a first step of advanced analysis processes. We will see in Section 3.3.3 and Section 3.3.4 how static voltage propagation can be used, as an effective prerequisite, in more advanced ERC techniques.

3.3.3 Ad-Hoc Patterns

Voltage propagation analyses are usually combined with some refinement approaches as a means to reduce the number of false alarms raised by such coarse abstraction. One of the classical complements to wire-based abstractions is circuit pattern recognition—which is the operation of identifying specific patterns within a circuit so that conclusions may be drawn regarding properties they verify. Ad-hoc patterns based approaches are most of the time targeted at extra-functional considerations. A typical application involves using it as a refinement technique, by analyzing circuit structures on which warnings were raised, to check whether they embed topologies known to act as protection blocks against the error of interest. For example, identifying topologies of existing circuitry for ESD protection on a path that was identified as vulnerable by a previous wire-based analysis [Lu and Bell, 2010; Lescot et al., 2012, 2015; Trivedi and Alvarez, 2015; Gevinti et al., 2015]. They can also be used to identify level-shifter topologies, typically to avoid false missing level-shifter alarms raised by static voltage propagation techniques (when the warning is raised within a level-shifter component itself) at the interface of two power domains [Lescot et al., 2012]. In other applications, pattern recognition is used to assert symmetry properties, such as differential pairs and current mirrors, in analog structures [Zwerger and Graeb, 2015; Neuner and Graeb, 2020b]. Similarly, Kunz et al. [2010] use pattern matching to identify structures that are known to be vulnerable against ESD events. Pattern-based analysis can also provide a qualitative analysis of such structures, e.g., to characterize the quality of ESD protections in a design [Viale et al., 2016; Viale and Allard, 2020], making it possible to quickly identify the most vulnerable parts of it. Patterns can be provided by both the designers of the circuit and the engineers in charge of the verification task, and can be used to specify the behavior of specific blocks with respect to the models of errors under analysis. These approaches hence rely on pre-defined pattern libraries, which provide all variations of the blocks to identify in order to ensure a given property.

From an implementation standpoint, the different approaches mainly use *graph isomorphism* algorithms to detect pre-selected patterns in a given circuit [Ohlrich et al., 1993; Pelz and Roettcher, 1994]. Viale and Allard [2020] propose an innovative process to characterize and visualize the quality of ESD protection patterns. In their framework, the quality of each ESD protection is quantified, and can be visualized through a visualization matrix, where each cell represents the vulnerability of the path between two I/O pads. Users can hence use this visualization feature to quickly apprehend which parts of the circuit are the most vulnerable, and where the design efforts should be put to improve the circuit resilience. This is particularly useful during the *revision* process, i.e., the task of modifying a circuit when the verification process highlighted some design rule violations. By using such approach, designers can hence reduce the effort required for revision by focusing on the actual vulnerabilities of the circuit, as highlighted by the analysis tool. CALIBRE PERC [Siemens EDA, 2023] is an industrial tool which implements this feature. It is commonly used in ERC verification approaches, notably for pattern recognition [Lescot et al., 2012; Viale et al., 2016; Viale and Allard, 2020; Hany and Hogan, 2022].

To scale up to billion-scale SoCs, pattern recognition techniques need to handle hierarchy. Neuner and Graeb [2020b] demonstrate how a hierarchical approach can use pattern matching techniques, in particular by matching module patterns for symmetry assertions. Other optimizations have been proposed to improve the performance of pattern recognition on realistic use cases, for example by using pattern reduction [Viale and Allard, 2020] to reduce the complexity of the approach.

Albeit their usefulness, pattern-based approaches do present some limitations. Their major limitation lies in their reliance on libraries of patterns, which may lead to erroneous conclusions during the analysis. On one hand, these libraries cannot be exhaustive. When pattern recognition is used as a refinement step after an analysis yielding false alarms, some warnings which should have been filtered out will not be filtered out. Conversely, if pattern recognition is used to identify vulnerable patterns, some might not be found by the tool, as they do not correspond precisely to the ones in the library. On the other hand, libraries are provided by the engineers—who leverage their expertise to identify relevant patterns—along with the properties of those patterns. An engineer may, by error, introduce an erroneous pattern in the library, which would make the analysis unsound; real errors from a prior analysis may be mistakenly filtered out. Additionally, some

functional blocks with interesting properties might replace the usual patterns and not be detected by a standard pattern-recognition technique. For example, functional parts might be considered as secondary protections against ESD events [Viale and Allard, 2020], but cannot be identified by pattern recognition in the general case. Moreover, while pattern recognition techniques can be used to ensure properties that cannot be detected by a simpler approach such as the ones described in Section 3.3.2, it cannot be used to accurately model the dynamic behavior of the designs. It is hence limited to identifying structures where the interesting behavior (e.g., the ability to prevent damages during an ESD event) does not depend on the electrical state of the circuit. In this way, the pattern recognition algorithms search for structures with a specific 'static behavior', but cannot consider the 'dynamic behavior' of those structures. For example, Viale et al. [2016] and Viale and Allard [2020] use pattern recognition to identify both static and dynamic ESD intellectual properties (IPs), e.g., respectively structures that are always a protection against ESD events, and structures that must be triggered to efficiently protect the design. However, as patterns do not model the electrical state of the circuit—hence cannot determine the value of the trigger signal—they specify that for dynamic ESD IPs, they can only assume that the pattern is properly controlled, i.e., that the trigger signal is correctly set. This results in potential flaws in the analysis, as their assumption cannot be enforced by the verification process. In this example, dynamic ESD IPs may never be enabled, while the verification process considers they are. In the following section, we will see how circuit states can be taken into account, and how they can be a good alternative to methods combining wire abstractions and pattern matching.

3.3.4 Switch-Based Abstractions

Switch-based abstractions consider circuit device states to be binary (on/off). This section focuses specifically on the approaches which make use of such abstraction as their main modeling hypothesis.

3.3.4.1 Iterative Voltage Propagation

Iterative voltage propagation is an improvement of the static voltage propagation technique introduced in Section 3.3.2, where the switching behavior of transistors is taken into account during the voltage propagation phase. The technique mainly relies on marking the voltage at each net and the state of each transistor of a circuit as initially unknown. Then, an input vector is used to start an iterative traversal of the nets. For each gate for which a voltage can be defined, the transistor is either switched on or off, and this information can be used for further propagation at the next iteration. At the end of this propagation phase, every net has been annotated with either a voltage, or an unknown value if the input vector was not sufficient to propagate information to this net. By design, this approach can estimate the voltages of the different nets of a circuit in a more accurate manner than does static voltage propagation.

A first approach for iterative voltage propagation was introduced by Blieck and Janssens [1996], which used a simple algorithm and type system to iteratively propagate the voltage values of the circuit nets as well as their electrical status (i.e., short-circuit, floating, etc.) to check for undesired currents and floating nets. This approach has been generalized by Zwerger and Graeb [2012, 2015], making it possible to efficiently propagate supplies in analog circuits with power-down mode. This allows, for example, to identify asymmetries of voltage conditions in power-down mode, which may lead to faster *aging* of the devices involved. They provide various models for basic components (namely, transistors, resistors and capacitors), and use them to build circuit graph representations, with vertices being the nets and edges being connections through the devices between nets. Doing so, they can define an ad-hoc propagation algorithm that iteratively scans this graph to propagate the supply values, as well as a given power-down signal value, throughout the circuit. Edges may be considered as on/off switches, depending on the voltages already propagated in the circuit. The complexity of this iteration-based supply propagation algorithm is $O(n^2)$, with n being the number of connections in the circuit. Hogan et al. [2013] propose a similar technique with a two-mode analysis: (1) the *vectorless* mode which corresponds to static voltage propagation—presented in Section 3.3.2, and (2) the *vectored* mode. In the vectored mode, users must provide an input vector to define the value of each input signal of the circuit (making this approach quite similar to simulation-based approaches). The voltage propagation algorithm can then use this input vector

to model transistors either as on or off switches (in contrast to the vectorless mode which models every transistor as a potentially switched-on transistor), iteratively propagating the supplies from source to drain in the design. This can hence be used to define the global electrical state of the design—i.e., the voltage at every net—with respect to an input vector. The approach has been demonstrated for EOS identification. Also, the identification of ESD vulnerabilities in circuits have been demonstrated by Neuner and Graeb [2019, 2020b] who, in addition, propose a generalization of the voltage propagation algorithm from Zwerger and Graeb [2012] to make it possible to handle hierarchical circuits. Such contribution is particularly helpful in making the approach scalable to industrial-scale designs.

While iterative supply propagation approaches seem to be a promising improvement of static supply propagation techniques, they currently suffer from some limitations. The first limitation is related to I/O value management. In most approaches, input values are not modeled [Blieck and Janssens, 1996; Zwerger and Graeb, 2012], and the iterative voltage propagation mechanism actually only propagates the different supplies in the circuit. This means that if an input net (with an unknown voltage) is applied to the gate of a transistor, the current status of this transistor is undefined. It is hence considered as being switched-on, for correctness reasons. On the other hand, in the vectored approach from Hogan et al. [2013], the user must provide an input vector to specify the voltage of each input net, meaning that each propagation is valid only for this vector. In the first case, the voltage propagation over-approximates the reachable electrical states. In the second case, supply propagation must be run on every input vector of interest [Zwerger and Graeb, 2015], which can be an exhaustive combination of all possible vectors, or a subset of it if defined by the user (either based on previous simulations, or user's proper knowledge), but this is in general too inefficient. Additionally, if only some scenarios are explored, this under-evaluates the set of reachable states. Moreover, in the current state of the art, iterative supply propagation is also limited due to its computational complexity [Zwerger and Graeb, 2012]. As it is an emerging approach, there is no standard, optimized tool to perform this propagation, and algorithmic choices as well as implementation optimizations are still to be proposed to improve its efficiency.

3.3.4.2 Power-Up Transformation

Power-down mode is a mode where power is cut from a circuit. It consists of some switching circuitry that is triggered by some power-down input signal which either isolates the functionally relevant part of the circuit from a supply or ensures that it is well connected to such supply. The switching circuitry can be as simple as a single MOSFET transistor whose gate is the power-down signal. The transistor, in this case, acts as intermediary connection between the supply and the circuit.

Power-up transformation is an operation proposed by Zwerger and Graeb [2015] as a prerequisite for ERC verification on analog circuits with a built-in power-down mode. Power-up transformation uses an identified 'power-down signal' as input, which is the trigger signal for the power-down mode. For example, the `PWD` signal in Figure 3.14 is used to trigger the power-down mode for the Inverter circuit. This analysis can hence be seen as a particular case of iterative voltage propagation, where the input vector consists of the power-down signal `PWD`. In addition, a library of patterns is also used to identify digital blocks which may be involved in the power-down circuitry.

Power-up transformation is often a necessary first step, because standard verification tools are not built to work with the power-down circuitry, and may fail to do so. For example, analyses may use pattern recognition to identify analog patterns that should be symmetrical, but such patterns may be different from the ones in the pattern recognition libraries, due to the power-down circuitry. Indeed, once the power-down circuitry has been identified, it can be removed; the corresponding transistors will be replaced either by a wire, if the power-up mode requires them to be switched-on, or will be removed otherwise. The resulting circuit can then be checked using standard techniques, such as pattern recognition. If we consider the Inverter circuit from Figure 3.14, we can use power-up transformation to remove the power-down circuitry (the PMOS transistor controlled by the `PWD` signal), in order to retrieve the original Inverter circuit (Figure 3.2). This makes it possible to model the circuit behavior during its *normal* operating mode.

In [Zwerger and Graeb, 2015], power-up transformation is followed by another supply propagation algorithm that uses an input vector to propagate voltages to the nets of the circuit. The

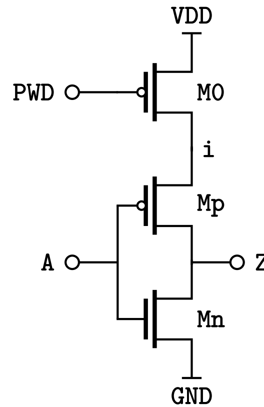


Figure 3.14: Inverter circuit with a power-down mode.

use case considered is the one for symmetry assertions, using two complementary pattern libraries: a first library to identify and remove power-down circuitry, and a second one to identify analog patterns in which symmetry rules must be asserted. For each of those patterns, and for each input vector, they identify symmetry violations and provide an error report that can be used to verify that no mismatch in the analog design may lead to premature aging of the circuit. Based on this approach, Neuner and Graeb [2019, 2020b] proposed a generalized approach to identify not only symmetry rules violations, but also undesired short-circuits and floating nets. Neuner and Graeb [2020b] also address a more practical concern by providing guidelines to revise designs after analysis. The technique has shown to be general enough to be integrated in more complex approaches for verification and synthesis. Based on the same techniques, initiatives have also been proposed for synthesis of power-down circuitry [Zwerger et al., 2017], in order to ensure the electrical properties of the circuits while adding a power-down mode.

Power-up transformation techniques are meant to be a necessary first-step of the analysis. As such, they must be efficient and scalable. By nature, the analysis is not much precise and thus relatively scalable. Hierarchical optimization techniques have been proposed in order to improve the performance of the analysis on large scale circuits [Neuner and Graeb, 2020b]. Still, power-up transformation approaches suffer from the limitations due to the techniques used. First of all, they rely on pattern recognition techniques, which lack exhaustiveness and correctness, as detailed in Section 3.3.3. Moreover, the voltage propagation phase relies on input vectors to identify power-down signals and are thus sensitive to the fact that these signals must be exhaustively identified.

3.3.4.3 SAT Solving Based Approaches

Formal methods are able to prove properties by reasoning about the state space of a system without *executing* it. They are usually sound (i.e., they provide a guarantee that all errors are detected), which makes them well-suited for the verification of safety-critical systems as well as RTL circuit verification—where proofs for correctness are highly demanded [Grimm et al., 2020].

Afonso and Monteiro [2017a] use a SAT solver to identify sufficient conditions for a short-circuit. To build the formula, the algorithm iteratively scans the netlist, and, for each device, adds the condition for a short-circuit to happen on this device to the existing short-circuit formula. The existing formula is simplified if an existing term in the formula is in contradiction with the new term to add. In this approach, the dynamic behavior of the circuit is modeled by encoding the switching conditions of the transistors directly in the short-circuit formula. Resistors and inductors are modeled as wires, and capacitors are modeled as open circuit, as the approach only considers the steady states of the circuits. Consequently, if no input combination is found, it also means that no short-circuit is possible in a given circuit. As an example, their method can be used to identify the short-circuit conditions in the circuit from Figure 3.15, where the input combination

$$\mathcal{V}(\mathtt{X}) = \mathcal{V}(\mathtt{GND}) \land \mathcal{V}(\mathtt{Y}) = \mathcal{V}(\mathtt{Z}) = \mathcal{V}(\mathtt{VDD})$$

leads to a direct path from the supply to the ground.

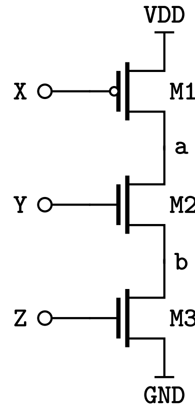


Figure 3.15: A three-transistor based disposition constituting an electrical path from supply (VDD) to ground (GND).

In their work, Afonso and Monteiro [2017a] propose an approach where the different terms of the formula are built independently from different parts of the circuit. This phase can be parallelized before merging the different terms for solving [Santos et al., 2020], leading to a reasonable scalability (circuits with 10 000 internal nodes are analyzed in less than a minute, in average, with 16 threads).

In [Newcomb, 2012a], a method to encode the semantics of a circuit is presented. The encoding considers not only 0/1 logical values for each net, but also high-impedance—i.e., floating nets. It uses a one-hot encoding of net states, assigning one Boolean variable for 'net has value 0', 'net has value 1' and 'net is floating'. The encoding is applied to floating net detection [Newcomb, 2012b]. The method relies on a prior search for suspect floating gates. A logical tree representation is then built for every suspect gate net, taking into account the immediate sub-circuit containing the gate net. An iterative process is run to check the satisfiability of the error on predecessor circuits, which repeats until either no predecessor circuit that can cause the gate to float is found, or until a circuit causing the gate to float is found. The analysis is performed on a subset of gate nets representing floating gate candidates, it then follows a counterexample-guided abstraction refinement (CEGAR) over circuit portions on which a candidate floating gate net is found. The net is only identified to be actually floating when a root cause is encountered. The authors also discuss the possibility of reducing the number of reported floating gates—e.g., by dismissing floating gates when they correspond to powered-off devices. The authors claim that parallelism can be used to independently analyze several floating gates and discover whether each of them is a real error or disappears when enlarging the boundaries of the analyzed circuit.

SAT solving have only been used for short-circuit and floating gates identification, but in principle it could be generalized to identify sufficient conditions for any path to be active in a given circuit. This could identify other errors and could be composed with other approaches, for example to perform voltage propagation by modeling the conditions for each net to be connected to the various supplies of a design. The restriction to a single supply in the SAT-based approach from Afonso and Monteiro [2017a] and Santos et al. [2020] is, in practice, an important limitation for an integration in standard design flows. For example, short-circuits are usually avoided—by construction—on CMOS designs synthesized from e.g., RTL, but they can appear by mistake on circuits with multiple supplies, for example as a consequence of a missing level-shifter. Not many results exist in the literature analyzing the scalability of SAT-based approaches. Moreover, while both of the presented approaches consider exploiting parallelism to improve the performances, hierarchy considerations are not discussed, hence limiting the potential of scalability. Another limitation of the approach in [Afonso and Monteiro, 2017a] is that the problem (that of detecting short-circuit conditions) is hard-coded in the modeling itself. This does not allow for modularity of the analysis so that more kinds of errors could be easily checked. Overall, this field of research highlights a promising usage of SAT solvers, long known to software verification engineers for symbolic model-checking, in the context of ERC.

3.3.5 Quantitative Approaches

There exist some more precise SAT-based approaches that aim at reasoning about analog circuit parameters in steady-states. Miller and Brewer [2013] introduce a quantitative approach to circuit modeling, considering the operation bounds of each device in terms of I-V relations. Their approach is based on a SAT encoding of the circuit, with a bounded behavior model of transistors. I-V relations are over-approximated with an envelope defined with a lower-bound and an upper-bound of the actual device characteristics. It considers actual numerical values of current and voltage, but uses bit-blasting for representing them. This makes it possible to capture different sorts of device variations, and can find circuit states where some device is operated beyond its specified bounds.

Such approach can be seen as complementary to ERC since it proposes an alternative to simulation to explore the state space exhaustively. It has the advantage of being applicable in analog circuits analysis. However, it does not easily scale beyond a few transistors.

3.3.6 Discussion of Related Work

Table 3.2 summarizes the different approaches presented, plus the contributions of this thesis—which we will present in next chapters. The table exhibits errors and properties considered by the various approaches, highlighting the fact that several properties may be checked with similar approaches. Note that we do not include synthesis approaches, such as the ones presented in [Zwerger et al., 2017; Neuner and Graeb, 2020a], in the table, as we focus on verification techniques rather than correct-by-construction methods that are safer but not adapted to some of the industrial practices.

One can first observe that many approaches are very similar and analyze similar errors and properties. For example, similar approaches, based on iterative supply propagation, have been proposed to diagnose undesired short-circuits in various functioning conditions [Blieck and Janssens, 1996; Zwerger and Graeb, 2012; Neuner and Graeb, 2019]. Another example lies in identifying vulnerabilities to ESD events, with similar methods based on both supply propagation and structure recognition [Viale and Allard, 2020; Lescot et al., 2015; Viale et al., 2016], highlighting how the various techniques can be combined for efficient analysis.

Concerning precision, switch-based abstractions is an interesting and under-exploited direction for improving the performances and reliability of ERC approaches. Switch-based abstractions model the fact that transistors can dynamically switch from on to off states, given the electrical state of a circuit, while operating at transistor level (i.e., without considering complex details such as differential equations to model the concrete behavior of the transistor). Formal methods [Afonso and Monteiro, 2017a; Santos et al., 2020] can effectively model the switch-based abstractions of circuits. In particular CEGAR and SMT techniques seem particularly adapted to reason about electrical circuit properties. As a matter of fact, recent initiatives based on formal methods have tackled ERC. For example, Plassan et al. [2017] use CEGAR to assert that clock domain crossing parts of circuits are correctly protected. While this check is performed at the RTL abstraction level, the problem itself is very similar to asserting that level-shifter structures are correctly inserted in a circuit at transistor level; ERC could hence benefit from such initiatives.

3.4 Overview of Contributions and Positioning

The use of formal methods for ERC remains very rare. The work of Afonso and Monteiro [2017b] constitutes an original contribution, demonstrating the use of SAT solving for short-circuit analysis at transistor-level. The fact that their approach is only targeted at single-supply circuits is clearly a limitation. Besides, having the error conditions hard-coded in the formula obstructs the way to modularity. This makes it hard to reuse the same framework to check other (different) properties without having to apply significant changes. Sometimes, the lack of modularity leads to the whole approach being unusable beyond the purpose it was initially built for. Hence, the top two criteria for the approaches this thesis aims at developing are multi-supply design coverage and its applicability to a wide range of electrical errors—with a big emphasis on modularity.

The SAT-based modeling technique of Newcomb [2012a] uses three symbolic values (low, high, and high-Z), but can, in fact, be extended further by defining more symbolic values for additional

supplies in a multi-supply circuit. However, it would be tricky to deal with voltage values situated between two supplies (which may result from short-circuits, for example). While SAT can—in theory—be adapted for such use, it appears to make things unnecessarily cumbersome. It requires us to translate the SMT formula into a SAT one, which is achievable with bit-blasting. Miller and Brewer [2013] have done so as part of their technique for the verification of device operation bounds in analog circuits.

It is even simpler to use SMT and not bother about bit-blasting altogether. SMT solvers are really powerful tools at handling problems involving numerical variables (as long as the theories invoked are decidable). They allow us to not only directly encode concrete voltage values as numeric variables, but also to model continuous intervals of voltages. The good news is that we do not have to sacrifice performance of the analysis with using SMT over SAT. So, that is the core idea we explore in this thesis—i.e., encoding multi-supply designs as SMT formulas and verifying them against electrical violations.

Our use of SMT solving in this very context is new. It is striking how no prior work applying SMT encodings to ERC exists to date. The *raison d'être* of this thesis is to fill such a gap. This research work *had* to be done sooner or later (but no more late). We hope this thesis contributes to catalyzing interest in this topic, and in the use of formal methods at large in addressing ERC.

Our goal is to introduce more modular circuit encoding approaches, which can deal with different kinds of errors on multi-supply designs. Ultimately, we combine the precision of [Miller and Brewer, 2013] with the benefits of SMT solving [Newcomb, 2012a,b; Afonso and Monteiro, 2017a]. We aspire to build a modular framework where any property to check can be plugged in without worrying about additional tweaks of the circuit modeling, for the circuit formula simply remains the same. In a way, we avoid having a verification approach tailored to only a specific use case and not usable elsewhere. Along the same lines, we seek eliminating—as much as possible—hypotheses which limit the applicability of the approaches developed to some kind of 'well-defined circuits'. To sum it up, we aim for approaches which handle all digital and analog circuits, singly- and multi-supplied ones, all by being topology-agnostic, and by making these approaches modular.

This thesis explores several circuit modeling variations, which we introduce later in Chapter 4. In addition to ERC applications, which we study in Chapter 5, we extend the application of our circuit semantics to reliability analysis—Chapter 6. Our reliability analysis framework is centered around finding most critical circuit states with respect to the failure mechanism considered. It, thus, involves formalizing and solving optimization problems—a task for which the νZ optimization modulo theories (OMT) solver is efficient.

Table 3.2: Summary of the approaches to transistor level electrical rule checking.

	References	**Analysis Technique**	**Circuit Semantics**				**Errors Considered**[1]
			Wire-based Section 3.3.2	*Ad-hoc patterns* Section 3.3.3	*Switch-based* Section 3.3.4	*Quantitative* Section 3.3.5	
	Blieck and Janssens [1996]	Type based & dynamic supply propagation	-	-	Supply propagation	-	SC
	Hogan et al. [2013]	Custom rule checking	Supply propagation in 'vectorless mode'	-	Supply propagation in 'vectored mode'	-	EOS
	Lescot et al. [2012]	Custom rule checking and error filtering	supply detection & supply propagation	Pattern identification for error filtering	-	-	FN, MLS, ESD, EOS
	Lescot et al. [2015]	ESD verification using cross domain checks and pattern matching	Supply detection & supply propagation	ESD structure recognition	-	-	ESD
	Viale et al. [2016]; Viale and Allard [2020]	ESD pattern recognition	Supply detection & supply propagation	ESD network identification	-	-	ESD
	Lu and Bell [2010]	Cross-domain interface verification with custom rule checking	Topology aware net type for supply propagation	Ad-hoc type-based ESD protections (anti-parallel diodes and GGNMOS)	-	-	ESD
	Liu et al. [2008]	ESD path identification with graph decomposition	ESD path identification	-	-	-	ESD
	Kunz et al. [2010]	ESD sensitive topology identification	-	Identification of topology patterns that are vulnerable to ESD events	-	-	ESD
	Zwerger and Graeb [2012]	Graph-based dynamic supply propagation	-	-	Supply propagation	-	SC, FN
	Zwerger and Graeb [2015]	Asymmetry voltage conditions in analog power-down mode	-	Power-down mode structures & symmetrical pattern recognition	Supply propagation & power-up transformation	-	Sym
	Neuner and Graeb [2019, 2020b]	Power-up transformation, dynamic supply propagation, and symmetry assertions	-	Power-down mode structures & symmetrical pattern recognition	Supply propagation & power-up transformation	-	SC, FN, Sym
	Afonso and Monteiro [2017a]; Santos et al. [2020]	Identifying short-circuit conditions using a SAT solver	Single supply	-	Encode short-circuit conditions as logical formula	-	SC
	Newcomb [2012a,b]	Checking the Satisfiability of floating gate candidates, by iterative propagation of error conditions	-	-	Refinement based analysis for identifying floating gates	-	FN
	Miller and Brewer [2013]	Checking operation bounds in analog circuits	-	-	-	Encode operation bounds as logic formula	SC, EOS
This Thesis	Section 4.5; Section 5.1	Identifying missing level shifters using an SMT solver	-	-	Encode both circuit semantics and erroră conditions as logical formula	-	MLS
	Section 4.6; Section 5.2	Detection of electrical overstress using an SMT solver	-	-	Encode both circuit semantics and error conditions as logical formula either with switch-based or quantitative semantics		EOS
	Section 4.5; Chapter 6	Worst-case analysis of circuit reliability using an OMT solver	-	-	Encode both circuit semantics and error as logical formula	Optimize objective function	TDDB failure mode

[1]SC: Short-Circuits; ESD: ElectroStatic Discharges (in particular, *Human-Body Model*); EOS: Electrical OverStress; FN: Floating Nets; MLS: Missing Level Shifters; Sym: Symmetry properties; TDDB: Time-Dependent Dielectric Breakdown.

4

Circuit Semantics

Any circuit schematic, or its equivalent netlist—usually written in the circuit description language (CDL) or automatically extracted—barely provides structural information about the circuit. It lists the devices the circuit is made of, and describes connections between them [Quarles et al., 1994]. To infer how the circuit behaves, one needs to refer to its semantics. The ideal circuit behavior is given by the concrete semantics (i.e., systems of differential equations describing the behavior of the circuit in detail over time), which may be useful in some contexts, like for the physical design of highly tailored electronic components, or to validate analog circuits functioning. Yet, the use of concrete semantics is seldom needful in the context of verification of electrical properties at transistor level, like verifying that the voltage applied on some device or the current flowing through it do not exceed its specified operating bounds. Using highly detailed semantics for such type of verification would clearly be superfluous, mostly due to scaling related issues, as we have thoroughly explained in Chapters 2 and 3. What we propose, instead, are abstract semantics, that are specifically targeted at reasoning about electrical properties—the end goal being to identify electrical errors in circuits. This chapter presents different semantics variants, demonstrates how they work in practice, and compares them to each other, both formally and via experimental evaluation. These semantics serve, firstly, as the basis for identifying electrical errors. By identifying electrical errors we mean identifying circuit states which correspond to these errors. The use cases considered and industrial benchmarks are presented in Chapter 5. Secondly, a method for circuit reliability analysis is presented in Chapter 6, for a specific case study regarding the aging of integrated circuits.

Note. As transient circuit states are out of scope of this thesis, whenever the term 'circuit state' is employed, it must be understood as 'circuit steady-state'.

4.1 Overview of the Analysis Framework

Electrical rule checking (ERC) requires intervening at transistor level, for transistor level provides necessary power-related information (as explained in Chapter 3). There, circuits are described in terms of netlists, i.e., sets of devices and connections between them. The core analysis approach of this thesis consists in taking circuit netlists and accompanying power information as input, and encoding them into logic formulas on which we can reason. Our reasoning makes use of satisfiability modulo theories (SMT) solving to tell whether an electrical configuration of a circuit can lead to an error. This is achieved by (1) encoding the circuit into a logical formula according to some semantics, (2) encoding the property to verify (e.g., absence of some error) as a Boolean predicate over semantics variables, and (3) checking the satisfiability of the conjunction of the circuit formula

and error formula. We delegate the formula solving task to an external solver, e.g., Z3, which gives an answer on whether the property is satisfiable. If so, the solver yields an error, i.e., a steady-state that is a counter-example for the property. Otherwise, we know that the error cannot occur. Figure 4.1 depicts the verification flow described. The examples of electrical errors we study in this thesis are missing level-shifters (MLS) and electrical overstress (EOS). We study both use cases in Chapter 5.

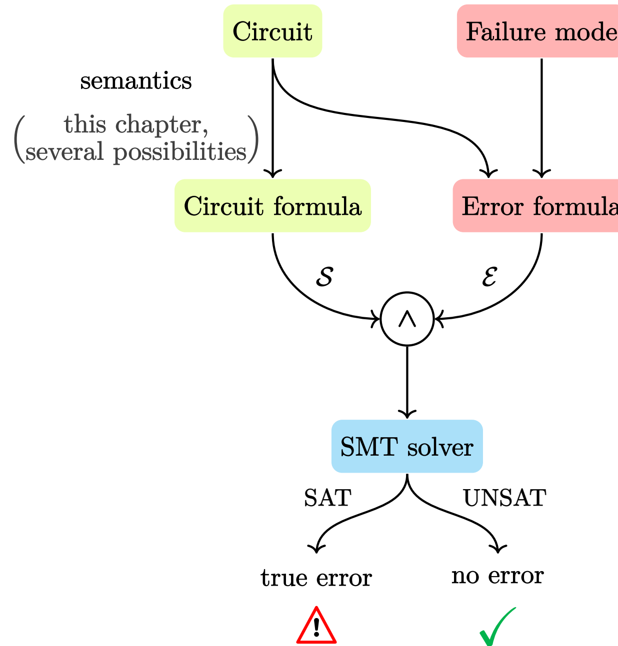


Figure 4.1: (Reminder) SMT based error detection flow.

This thesis also extends the use of circuit semantics to reliability analysis. In particular, we consider the problem of identifying the worst-case circuit state with respect to a specific circuit degradation mechanism. We consider the time-dependent dielectric breakdown (TDDB) type of failures, where the constant voltage stress values applied on each device in a steady-state directly impacts the circuit's degradation [Khorasani et al., 2013]. Different steady-states may lead to different paces of circuit degradation. We opt for approximate encodings of device reliability models that ensure the generated SMT formulas remain in the theory of linear real arithmetic (LRA)—which is decidable [Fourier, 1827]. An expression representing the circuit reliability is then computed, and an optimization modulo theories (OMT) solver is asked to minimize it. The solver must also preserve the circuit semantics (defined by the SMT part) while doing so. The output obtained is hence a legitimate circuit state—one which makes the circuit deteriorate the fastest. This use case is studied in Chapter 6. A high-level overview of this analysis flow is illustrated with Figure 4.2.

This analysis framework is implemented in a software prototype, written in OCaml ($\approx$ 12 000 lines of code). The applications presented in this manuscript and the various experimental benchmarks are generated using this prototype. Due to the industrial context of this thesis, this software is not available to the public.

Before we dive into presenting circuit semantics, let us first introduce the notation system and define a few concepts related to transistor level circuit modeling. Formalizing such concepts upfront will facilitate the task of defining the semantics later, and will hopefully help the reader have a better experience reading through this chapter.

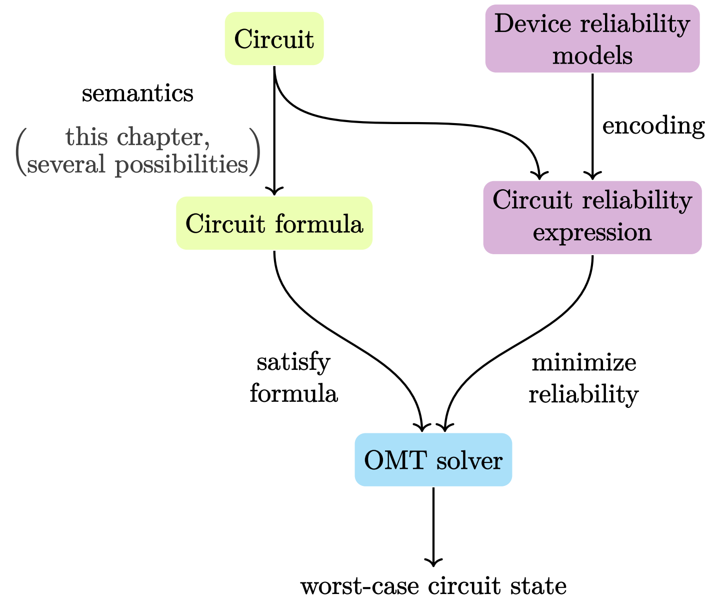


Figure 4.2: OMT based worst-case reliability analysis flow.

4.2 Common Formalism

This section serves to formalize aspects that are common to all semantics variants we introduce. For ease of reading, we use different typestyles for different sorts of information handled. The usage of each of these notations remains consistent, in terms of what they signify, throughout this manuscript.

Bold font is used to denote different sets of elements of a circuit (nets, supplies, devices, etc.).

SMALLCAPS font is used to denote static functions. These are functions which operate on the netlist of the circuit (usually on its graph representation), with no consideration of any circuit semantics whatsoever. Their return values are pure structural information (e.g., a transistor's gate, source, or drain). Static functions are invoked when building the circuit formula (see Figures 4.1 and 4.2). In our implementation, they correspond to OCAML functions returning a value during translation from a CDL netlist to an SMT formula.

Calligraphic letters are reserved for dynamic functions, which are the variables constrained by circuit semantics, i.e., variables that appear in the SMT formula encoding the circuit. Typically, these are device on/off states, net voltages, currents, etc. In practice, they are SMT-LIB functions generated by the OCAML compiler.

Sans-serif font is occasionally used to denote elements of some finite sets.

All of these notations can possibly be decorated with subscripts and/or superscripts, as a way to differentiate variations of a notation.

4.2.1 Notation System

We consider transistor level circuit descriptions. A circuit is composed of devices connected to each other in a specific way. Connections between devices are possible via the parts they expose, called terminals (i.e., gate, source, and drain for transistors, anode and cathode for diodes, etc.). The exposed parts of a circuit are its pins, e.g., inputs, outputs, supplies. Circuits may be described hierarchically, by instantiating other circuits, in which case the instantiated circuits are called devices, while their pins correspond to device terminals in the instantiating circuit. The highest level of the circuit hierarchy is often called 'top-level'. It typically corresponds to the description of some

intellectual property (IP)—i.e., circuit blocks which are intended to fulfill a precise functionality when used in a standalone mode, or may correspond to the description of a whole system on chip (SoC) made of different IPs. The top-level circuit pins are referred to as 'pads' instead of pins. In this thesis, hierarchical analysis of full circuits is out of scope. It is orthogonal work that can be conducted as a complement to this thesis. Instead, we target the analysis of the flat subcircuits (i.e., the leaves) constituting full circuits. Some of these subcircuits may correspond to a full IP, but that is rarely the case.

4.2.1.1 Circuit Structural Information

This section presents various notations and functions related to reasoning about structural information of the circuit.

4.2.1.1.1 Nets

Circuit nets, also called wires, connect the terminals of the different devices composing the circuit. The set of a circuit's nets is denoted $\mathbf{N}$. Connections between devices are identified with the shared nets among their terminals. Nets are said to be 'internal' when they are only accessible within the circuit—but not from the outside. Syntactically, internal nets are not part of the list of pins in a circuit netlist. A net may as well be used to expose the circuit to its external environment, typically to establish connections with other circuits. This is the case for input and output (I/O) pins, supplies, and grounds. One cannot say a net is internal or external in the absolute, because such classification is only meaningful in the context of some specific circuit or subcircuit.

To illustrate this, consider the hierarchically built two-stage buffer circuit of Figure 4.3. Its CDL netlist is shown in Listing 4.1. The top-level circuit consists of two buffers connected in series. A 'chip enable' (`CE`) signal either activates or deactivates the power-down mode (i.e., a mode where power supplies are disabled, often to reduce power consumption), depending on its voltage. The connection between the two buffers corresponds to the top-level's internal net `x`, which, from the perspective of the two buffer instances, is a pin. The same applies on wires `x/X1`, `x/X2`, and `y`.

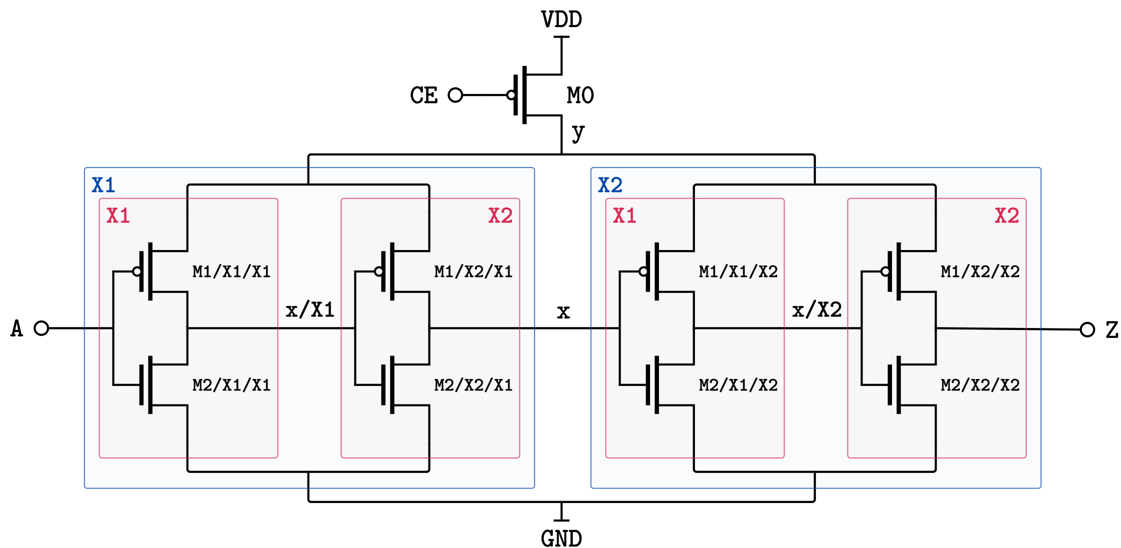


Figure 4.3: Two-stage buffer circuit schematic.

Note. For a detailed presentation of the CDL syntax, the reader may refer back to Section 3.1.2.

```
1  * Top-level circuit
2
3  .SUBCKT two_stage_buffer CE A Z VDD GND
4  M0 y CE VDD VDD pmos
5  X1 A x y GND buffer
6  X2 x Z y GND buffer
7  .ENDS
8
9  * Buffer subcircuit
10
11 .SUBCKT buffer A Z VDD GND
12 X1 A x VDD GND inverter
13 X2 x Z VDD GND inverter
14 .ENDS
15
16 * Inverter subcircuit
17
18 .SUBCKT inverter A Z VDD GND
19 M1 Z A VDD VDD pmos
20 M2 Z A GND GND nmos
21 .ENDS
```

Listing 4.1: Two-stage buffer (Figure 4.3) circuit netlist.

4.2.1.1.2 Pins

Pins are external nets of the circuit via which connections with the external environment is possible. We denote $\mathbf{S}$ the set of supplies, $\mathbf{I}$ the set of inputs, and $\mathbf{O}$ the set of output pins. Note that some pins may be used as both input ($\mathbf{I}$) and output ($\mathbf{O}$), in which case $\mathbf{I} \cap \mathbf{O}$ is not empty. The intersection corresponds to the so-called 'input-output' (or 'in-out') pins. The set of pins is denoted $\mathbf{P} \triangleq \mathbf{S} \cup \mathbf{I} \cup \mathbf{O}$. We define each of these sets more formally, later, in Section 4.2.2. All pins are nets: $\mathbf{P} \subseteq \mathbf{N}$.

4.2.1.1.3 Devices

The set of devices is denoted $\mathbf{D}$. Sets $\mathbf{D}_{\texttt{PMOS}}$ and $\mathbf{D}_{\texttt{NMOS}}$ respectively denote the set of a circuit's PMOS and NMOS devices, with $\mathbf{D}_{\texttt{MOS}} \triangleq \mathbf{D}_{\texttt{PMOS}} \cup \mathbf{D}_{\texttt{NMOS}}$, and $\mathbf{D}_{\texttt{PMOS}} \cap \mathbf{D}_{\texttt{NMOS}} = \emptyset$. $\mathbf{D}_{\texttt{RES}}$ denotes the set of resistors, and $\mathbf{D}_{\texttt{DIO}}$ denotes the set of diodes. By definition, $\mathbf{D} \triangleq \mathbf{D}_{\texttt{MOS}} \cup \mathbf{D}_{\texttt{RES}} \cup \mathbf{D}_{\texttt{DIO}}$.

Note that devices whose behavior is intrinsically transient, like capacitors and coils, are not handled in this thesis, since our focus is on steady-state analysis. In a steady-state analysis, capacitor terminals could be modeled as always-disconnected, and that of coils as short-circuited.

4.2.1.1.4 Device Terminals

Transistors. NMOS and PMOS gate, source, and drain terminals (Figure 4.4), are given, respectively, by functions

$$\textsc{Gate}\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N},$$
$$\textsc{Src}\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N},$$
$$\textsc{Drn}\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N}.$$

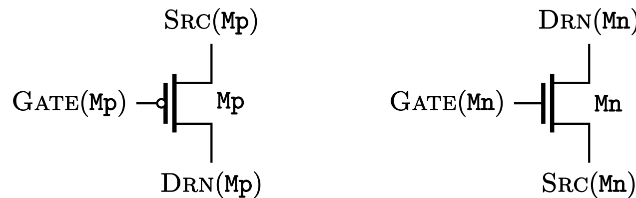


Figure 4.4: Transistor terminals.

Resistors. Resistor terminals (Figure 4.5) are retrieved, in lexicographical order, with functions

$$\mathrm{A}\colon \mathbf{D}_{\texttt{RES}} \longrightarrow \mathbf{N},$$

$$\mathrm{B}\colon \mathbf{D}_{\texttt{RES}} \longrightarrow \mathbf{N}.$$

This is based on preference, since it is an implementation choice.

Figure 4.5: Resistor terminals.

Diodes. Diode's anode and cathode terminals (Figure 4.6) are given, respectively, by functions

$$\textsc{Anode}\colon \mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbf{N},$$

$$\textsc{Cathode}\colon \mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbf{N}.$$

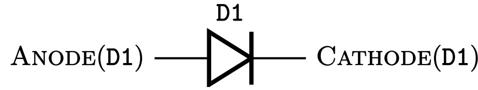


Figure 4.6: Diode terminals.

4.2.1.1.5 Net Neighborhood

Neighbors of a net $\mathit{self} \in \mathbf{N}$, denoted $\textsc{Neighbors}(\mathit{self})$, is the set of triples of the form (self, X, n), such that some device $X \in \mathbf{D}$ is statically connected to both *self* and another net $n \in \mathbf{N}$, i.e., nets *self* and n are both terminals of the same device X, with the condition that direct connection between them (through X) is semantically possible. The possible connections are between source and drain for transistors, between anode and cathode for diodes, and between the two resistor terminals for resistors.

Neighborhood information is static information that may, for example, be extracted from the circuit netlist, without taking into account information about its semantics. For a given circuit state (with respect to some circuit semantics), two neighboring nets are said to be *dynamically* connected if and only if a device allows such connection. Later, we present what exactly that means for different semantics.

Transistors. For transistors, either *self* is $\textsc{Src}(X)$ and n is $\textsc{Drn}(X)$, or vice versa. There is no connection between gate and source, nor between gate and drain, except when such connection is enforced (e.g., using a common net connecting gate and source or gate and drain). The gate is hence not directly reachable from source or drain through X. Function $\textsc{Neighbors}_{\texttt{MOS}}$, defined in $\textsc{Neighbors}_{\texttt{MOS}\mathrm{def}}$, gives, for a net, transistor neighbors only (Notation: $\mathscr{P}(\mathbb{E})$ is the powerset of a set $\mathbb{E}$).

$$\begin{aligned}\textsc{Neighbors}_{\texttt{MOS}}\colon \mathbf{N} &\longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D} \times \mathbf{N}) \\ \mathit{self} &\longmapsto \bigcup_{\substack{X \in \mathbf{D}_{\texttt{MOS}},\\ \mathit{self} \in \{\textsc{Src}(X),\ \textsc{Drn}(X)\},\\ \mathit{other} \in \{\textsc{Src}(X),\ \textsc{Drn}(X)\},\\ \mathit{self} \neq \mathit{other}}} \Big\{ (\mathit{self},\ X,\ \mathit{other}) \Big\}\end{aligned} \qquad (\textsc{Neighbors}_{\texttt{MOS}\mathrm{def}})$$

Resistors. For resistors, *self* and n are, in any order, the A-pole ($\mathrm{A}(X)$) or B-pole ($\mathrm{B}(X)$) of the resistor. Function $\textsc{Neighbors}_{\texttt{RES}}$, defined in $\textsc{Neighbors}_{\texttt{RES}\,\mathrm{def}}$, gives, for a net, resistor neighbors only.

$$\begin{aligned}\textsc{Neighbors}_{\texttt{RES}} : \mathbf{N} &\longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D} \times \mathbf{N}) \\ \mathit{self} &\longmapsto \bigcup_{\substack{X \in \mathbf{D}_{\texttt{RES}}, \\ \mathit{self} \in \{\mathrm{A}(X),\ \mathrm{B}(X)\}, \\ \mathit{other} \in \{\mathrm{A}(X),\ \mathrm{B}(X)\}, \\ \mathit{self} \neq \mathit{other}}} \left\{ (\mathit{self},\ X,\ \mathit{other}) \right\}\end{aligned} \qquad (\textsc{Neighbors}_{\texttt{RES}\,\mathrm{def}})$$

Diodes. Similarly—for diodes, *self* and n are, in any order, $\textsc{Anode}(X)$ or $\textsc{Cathode}(X)$. Function $\textsc{Neighbors}_{\texttt{DIO}}$, defined in $\textsc{Neighbors}_{\texttt{DIO}\,\mathrm{def}}$, gives, for a net, diode neighbors only.

$$\begin{aligned}\textsc{Neighbors}_{\texttt{DIO}} : \mathbf{N} &\longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D} \times \mathbf{N}) \\ \mathit{self} &\longmapsto \bigcup_{\substack{X \in \mathbf{D}_{\texttt{DIO}}, \\ \mathit{self} \in \{\textsc{Anode}(X),\ \textsc{Cathode}(X)\}, \\ \mathit{other} \in \{\textsc{Anode}(X),\ \textsc{Cathode}(X)\}, \\ \mathit{self} \neq \mathit{other}}} \left\{ (\mathit{self},\ X,\ \mathit{other}) \right\}\end{aligned} \qquad (\textsc{Neighbors}_{\texttt{DIO}\,\mathrm{def}})$$

All Devices. The set of neighbors of a net—considering all transistors, resistors, and diodes—is hence constructed from the union of all device type neighbors. This is defined in $\textsc{Neighbors}_{\mathrm{def}}$.

$$\begin{aligned}\textsc{Neighbors} : \mathbf{N} &\longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D} \times \mathbf{N}) \\ \mathit{self} &\longmapsto \textsc{Neighbors}_{\texttt{MOS}}(\mathit{self}) \cup \textsc{Neighbors}_{\texttt{RES}}(\mathit{self}) \cup \textsc{Neighbors}_{\texttt{DIO}}(\mathit{self})\end{aligned} \qquad (\textsc{Neighbors}_{\mathrm{def}})$$

Example. To illustrate the net neighborhood notion, consider the circuit schematic in Figure 4.7, where pins `S` and `G` are, respectively, the primary supply and ground. The neighbors, for each of the circuit nets, are the following:

$$\textsc{Neighbors}(\texttt{A}) = \{\ (\texttt{A, R1, x})\ \}$$

$$\textsc{Neighbors}(\texttt{B}) = \emptyset$$

$$\textsc{Neighbors}(\texttt{S}) = \{\ (\texttt{S, M1, y})\ \}$$

$$\textsc{Neighbors}(\texttt{G}) = \{\ (\texttt{G, M2, x}),\ (\texttt{G, D1, y})\ \}$$

$$\textsc{Neighbors}(\texttt{x}) = \{\ (\texttt{x, M2, G}),\ (\texttt{x, R1, A})\ \}$$

$$\textsc{Neighbors}(\texttt{y}) = \{\ (\texttt{y, M1, S}),\ (\texttt{y, D1, G})\ \}$$

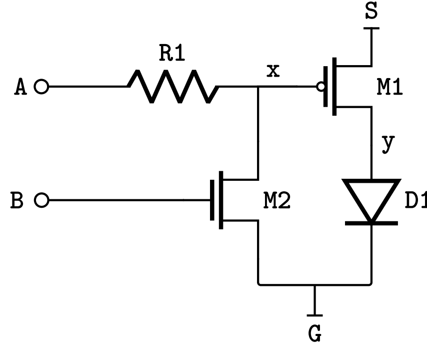


Figure 4.7: Example circuit schematic containing different device types.

4.2.1.1.6 Reachable Supplies

Function $\textsc{ReachSuppl}\colon \mathbf{N} \longrightarrow \mathscr{P}(\mathbf{S})$, gives the set of reachable supplies (including grounds) from a given net. The implementation of $\textsc{ReachSuppl}$ is achieved with some static voltage propagation algorithm.

The set of reachable supplies can be recursively constructed, starting from the net of interest and then exploring the neighboring nets and their respective neighbors, and so forth, until a supply is reached or until all nets of a path were visited. We introduce (helper) function $\textsc{ReachSuppl}^{\natural}\colon \mathbf{N} \times \mathscr{P}(\mathbf{N}) \longrightarrow \mathscr{P}(\mathbf{S})$ (defined in $\textsc{ReachSuppl}^{\natural}{}_{\text{def}}$) which, given an initial set of 'visited' nets—denoted V (which is initially the empty set), gives the reachable supplies for a net.

$$\textsc{ReachSuppl}^{\natural}\colon \mathbf{N} \times \mathscr{P}(\mathbf{N}) \longrightarrow \mathscr{P}(\mathbf{S})$$
$$(n, V) \longmapsto \begin{cases} \{n\} & \text{if } n \in \mathbf{S} \\ \displaystyle\bigcup_{\substack{(n,X,p)\,\in\,\textsc{Neighbors}(n) \\ p\,\notin\,V}} \textsc{ReachSuppl}^{\natural}(p, V \cup \{n\}) & \text{if } n \notin \mathbf{S} \end{cases} \qquad (\textsc{ReachSuppl}^{\natural}{}_{\text{def}})$$

The set of reachable supplies is hence defined with $\textsc{ReachSuppl}_{\text{def}}$, by setting the initial value of the set of visited nets to the empty set.

$$\textsc{ReachSuppl}\colon \mathbf{N} \longrightarrow \mathscr{P}(\mathbf{S})$$
$$n \longmapsto \textsc{ReachSuppl}^{\natural}(n, \emptyset) \qquad (\textsc{ReachSuppl}_{\text{def}})$$

To illustrate this on the example of Figure 4.7, we have:

$\textsc{ReachSuppl}(\mathtt{A}) = \{\mathtt{G}\}$

$\textsc{ReachSuppl}(\mathtt{B}) = \emptyset$

$\textsc{ReachSuppl}(\mathtt{S}) = \{\mathtt{S}\}$

$\textsc{ReachSuppl}(\mathtt{G}) = \{\mathtt{G}\}$

$\textsc{ReachSuppl}(\mathtt{x}) = \{\mathtt{G}\}$

$\textsc{ReachSuppl}(\mathtt{y}) = \{\mathtt{G}, \mathtt{S}\}$

4.2.1.2 Circuit State Information

This section presents the notations and functions used when referring to the semantics of circuit components.

4.2.1.2.1 Voltage Function

The voltage function (which appears as a dynamic function in the circuit's SMT formula) is encoded, for each net with function $\mathcal{V}\colon \mathbf{N} \longrightarrow \mathbb{V}$, where $\mathbb{V}$ is the set of voltage values. It is defined, later, based on the circuit semantics considered. Additionally, we define the following short notations, to enhance readability:

$$\begin{aligned}\mathcal{V}_{\text{G}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\textsc{Gate}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{S}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\textsc{Src}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{D}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\textsc{Drn}(X))\end{aligned}$$

$$\begin{aligned}\mathcal{V}_{\text{GS}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{G}}(X) - \mathcal{V}_{\text{S}}(X)\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{GD}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{G}}(X) - \mathcal{V}_{\text{D}}(X)\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{SD}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{S}}(X) - \mathcal{V}_{\text{D}}(X)\end{aligned}$$

$$\begin{aligned}\mathcal{V}_{\text{DS}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{D}}(X) - \mathcal{V}_{\text{S}}(X)\end{aligned}$$

$$\begin{aligned}\mathcal{V}_{\text{A}}\colon \mathbf{D}_{\texttt{RES}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\text{A}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{B}}\colon \mathbf{D}_{\texttt{RES}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\text{B}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{AB}}\colon \mathbf{D}_{\texttt{RES}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{A}}(X) - \mathcal{V}_{\text{B}}(X)\end{aligned}$$

$$\begin{aligned}\mathcal{V}_{\text{A}}\colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\textsc{Anode}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{K}}\colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}(\textsc{Cathode}(X))\end{aligned} \qquad \begin{aligned}\mathcal{V}_{\text{AK}}\colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{V} \\ X &\longmapsto \mathcal{V}_{\text{A}}(X) - \mathcal{V}_{\text{K}}(X)\end{aligned}$$

4.2.1.2.2 Current Function

To every triple $(n_1, X, n_2) \in \mathbf{N} \times \mathbf{D} \times \mathbf{N}$ that is part of some neighborhood, we associate a value representing the current intensity from net n_1 to net n_2 via device X. Such value is accessible via function $\mathcal{I}\colon \mathbf{N} \times \mathbf{D} \times \mathbf{N} \longrightarrow \mathbb{I}$, where $\mathbb{I}$ is the set of current values. Definitions of function $\mathcal{I}$ are provided later, whenever relevant, with respect to circuit semantics. The term $\mathcal{I}(n_1, X, n_2)$ is defined only when net n_1 reaches net n_2 via device X (both nets being terminals of the device). By convention, the value of current from n_1 to n_2 is the opposite of the value of current from n_2 to n_1, i.e.,

$$\forall (n_1, X, n_2) \in \mathbf{N} \times \mathbf{D} \times \mathbf{N},\ \mathcal{I}(n_1, X, n_2) = -\mathcal{I}(n_2, X, n_1).$$

We define the following short notations:

$$\begin{aligned}\mathcal{I}_{\text{SD}}\colon \mathbf{D}_{\texttt{MOS}} &\longrightarrow \mathbb{I} \\ X &\longmapsto \mathcal{I}(\textsc{Src}(X), X, \textsc{Drn}(X))\end{aligned}$$

$$\begin{aligned}\mathcal{I}_{\text{AB}}\colon \mathbf{D}_{\texttt{RES}} &\longrightarrow \mathbb{I} \\ X &\longmapsto \mathcal{I}(\text{A}(X), X, \text{B}(X))\end{aligned}$$

$$\begin{aligned}\mathcal{I}_{\text{AK}}\colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{I} \\ X &\longmapsto \mathcal{I}(\textsc{Anode}(X), X, \textsc{Cathode}(X))\end{aligned}$$

4.2.1.2.3 Device States

If we consider a switch-based device abstraction, device states can be encoded with a single Boolean variable.

Transistors. For transistors, the state can be encoded with predicate $\mathcal{O}n\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$, where $\mathbb{B} = \{\top, \bot\}$ is the set of Booleans, with $\top$ denoting 'true' and $\bot$ denoting 'false'. The exact encoding is presented as part of the circuit semantics considered (Sections 4.4, 4.5 and 4.5.4)

Transistors can also be viewed as multi-state devices, rather than binary on/off. We study more elaborate transistor states encodings, beyond switch-based semantics, in Section 4.6.

Resistors. Regardless of the abstractions considered, modeling resistors as 'always on' devices is always valid. In fact, any voltage difference across a resistor's terminals implies a non-null current through the resistor.

Diodes. Diodes are different from resistors in that they can only be viewed as being on if the anode voltage is greater than the cathode's (offset by a threshold), in which case current flows from anode to cathode. Otherwise, no current flows through the diode. That is a binary behavior which can also be modeled with the same predicate $\mathcal{O}n$, initially defined for transistors, by extending it to diodes. We present the detailed encodings, as we introduce circuit semantics, starting Section 4.4.

4.2.2 Circuit Environment

In a CDL file (circuit description), the circuit environment refers to the set of pins ($\mathbf{P}$). The definition of inputs and outputs depends on the context of use and on the intention of the designer. Often, the supplies are defined in some power configuration file. Sometimes, a circuit may have no supply pins, and is thus not powered at all, e.g., switches which act as connectors between different pins of different circuits, or multiplexers which select an output signal based on a selection signal. An example of a supply-less circuit—a multiplexer—is shown in Figure 4.8. Based on the voltage value of the selection signal `S`, the output `Z` is either connected to `A` or `B`. A low value of $\mathcal{V}(\texttt{S})$ turns transistor `M1` on and turns `M2` off, resulting in $\mathcal{V}(\texttt{Z}) = \mathcal{V}(\texttt{A})$. When $\mathcal{V}(\texttt{S})$ is high, `M1` is turned-off and `M2` is turned-on, resulting in $\mathcal{V}(\texttt{Z}) = \mathcal{V}(\texttt{B})$.

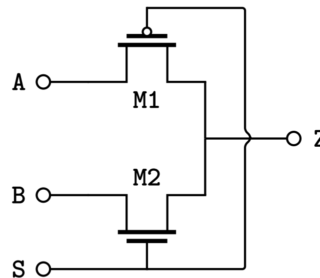


Figure 4.8: Two-input multiplexer schematic.

In many circuits, including the example from Figure 4.8, identifying inputs and outputs—functionally speaking—is not trivial. The circuit schematic or netlist alone provides no information to distinguish inputs from outputs. Telling which is which highly depends on the context of use. In the two-input multiplexer example, when one of the transistors `M1` or `M2` is switched-on, the circuit simply establishes a connection either between `A` and `Z` (via `M1`) or between `B` and `Z` (via `M2`). Instead of considering `Z` as the output, one may, for example, force a value on `Z` and release all external constraints on `A` and `B`. The circuit then changes its function, as it can now be viewed as transferring the voltage value of `Z` *backwards* to either `A` or `B`.

In the absence of strict agreed-on definitions of I/O pins in transistor level circuit netlists, we come up with our own definitions, which are rather relaxed. In our definitions, we remain generic, and we avoid strong hypotheses about the way circuit pins may be used.

Another aspect related to input modeling relies in determining which constraints to apply regarding their voltage values. Based on common use cases, inputs voltages can be seen as direct connections to supplies. For digital circuits, each input is modeled as being connected to a single supply (in a given circuit state). However, for analog circuits, where input pins are also allowed intermediate values between every two supplies, the constraint regarding the number of supplies

inputs are connected to is more relaxed. In this case, an input is modeled as being connected to multiple supplies. When these supplies are different, a short-circuit is created on the input, modeling thus some intermediate voltage value on the input. Such connections can be modeled with switches, as shown for input `A` of the buffer circuit in Figure 4.9. If more than one switch is closed, that creates a short-circuit whose value is bounded by the voltage values of the supplies involved (between the minimum and maximum values of connected supplies).

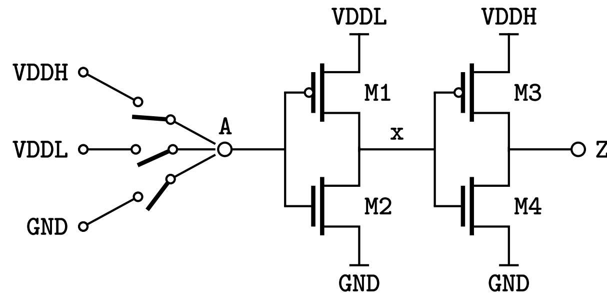


Figure 4.9: Buffer circuit input environment.

In general, inputs are used as transistor gates, but this is not always the case. For example, a known-to-be-output pin may as well be forced from the exterior by connecting it to some supply. Typically, it is common practice in CMOS-based designs to expose, for a circuit describing some logic function ϕ, two outputs ϕ—itself and its negation $\neg\phi$. An example is depicted in Figure 4.10, showcasing a CMOS-based AND gate, where the output of both AND and NAND functions can be *read* from the exterior—as their corresponding wires (nets `NAND` and `AND`) are exposed as pins. However, as pin `NAND` is also the gate of some transistors (`M5` and `M6`), its value could be forced from the exterior as well, based on our previous intuition on identifying inputs (i.e., inputs corresponding to circuit pins that are used as transistor gates). This is illustrated in Figure 4.11. If the `NAND` pin is not forced to a supply, its voltage value would simply indicate the logical NAND operation between `A` and `B`. Some external assignments, however, would lead to conflicts (short-circuits) if different from the signal the pin gets from within the circuit.

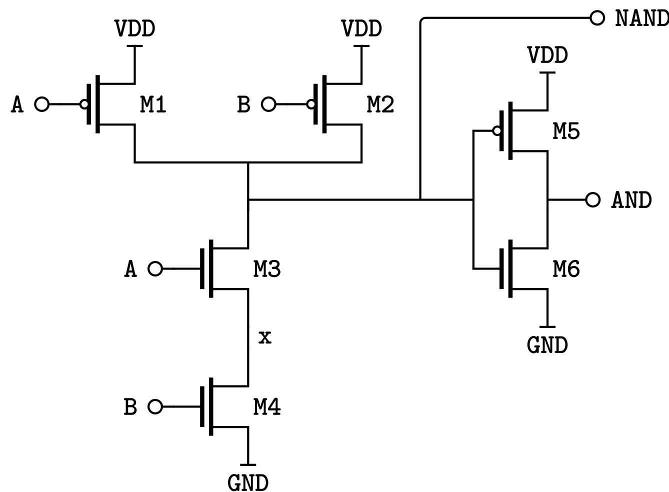


Figure 4.10: AND gate.

In this thesis, we adopt a rather conservative approach, covering thus a larger number of possibilities of circuit environment configurations. Although some of these states may be nonsensical, they are still legitimate with respect to physics, and it does no harm to consider them—especially in the absence of any custom explicit specification of the environment, e.g., provided by the user. Later, when we tackle specific use cases (Chapters 5 and 6), we draw our own hypotheses from the

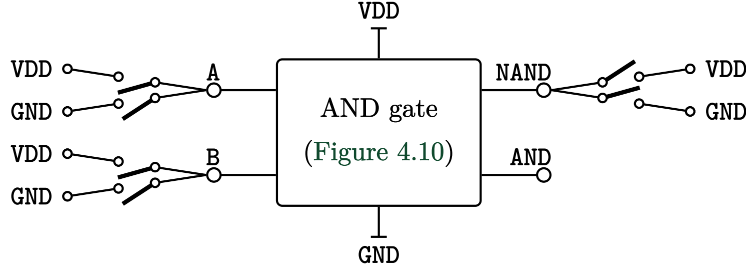


Figure 4.11: AND gate environment, assuming pin NAND can be used as both input and output.

nature of circuit benchmarks used for the experimental studies. We hence adapt the constraints on the environment of the circuit independently from circuit semantics.

4.2.2.1 Supplies

We consider supplies and ground pins as given alongside the circuit description, in addition to their respective possibles values, i.e., a set of possible voltage values is associated to each supply. Typically, each combination of supply voltage values among the circuit's supplies corresponds to some mode of operation. For example, setting part or all of the supplies to 0 V means that the corresponding circuitry is used in the power-down mode. The choice of the supply values may also depend on the context where the sub-circuit is instantiated. The power domains are thus chosen so as to remain coherent with the sub-circuit's environment. We consider a static function

$$\textsc{AvailSuppl}\colon \mathbf{S} \longrightarrow \mathscr{P}(\mathbb{V}),$$

which returns for each supply pin the set of its possible voltage values. Supplies can be constrained, each, so that they are set to one of their available values. In terms of semantics rules, this is encoded as $\mathcal{R}_{\mathbf{S}}$.

$$\bigwedge_{s \in \mathbf{S}} \left(\bigvee_{v \in \textsc{AvailSuppl}(s)} \mathcal{V}(s) = v \right) \tag{$\mathcal{R}_{\mathbf{S}}$}$$

4.2.2.2 Inputs

Inputs are used to *drive* devices to either activate or deactivate them. We thus assume that inputs are used as transistor gates. We define inputs as the set of pins excluding supplies (i.e., $\mathbf{P} \setminus \mathbf{S}$) who correspond to some transistor gate net. This is defined with $\mathbf{I}_{\text{def}}$.

$$\mathbf{I} \triangleq \bigcup_{\substack{n \in \mathbf{P} \setminus \mathbf{S},\\ n \in \mathbf{G}}} \{n\}, \text{ with } \mathbf{G} \triangleq \bigcup_{X \in \mathbf{D}_{\texttt{MOS}}} \{\textsc{Gate}(X)\} \tag{$\mathbf{I}_{\text{def}}$}$$

We consider inputs' voltages to be restricted to the context of the circuit's supply values. Every input pin can be set to the same voltage of any available supply ($\mathcal{R}_{\mathbf{I}}$).

$$\bigwedge_{i \in \mathbf{I}} \left(\bigvee_{s \in \mathbf{S}} \mathcal{V}(i) = \mathcal{V}(s) \right) \tag{$\mathcal{R}_{\mathbf{I}}$}$$

Later (in Chapters 5 and 6), we customize constraints on the environment ($\mathcal{R}_{\mathbf{S}}$ and $\mathcal{R}_{\mathbf{I}}$) when running experiments, based on the hypotheses provided by each case study.

4.2.2.3 Outputs

We define the set of outputs as the set of pins for which the set of reachable supplies is not empty. This is given by $\mathbf{O}_{\text{def}}$.

$$\mathbf{O} \triangleq \bigcup_{\substack{n \in \mathbf{P}, \\ \textsc{ReachSuppl}(n) \neq \emptyset}} \{n\} \qquad (\mathbf{O}_{\text{def}})$$

Note. Input-outputs (in-out pins) are defined as the elements of $\mathbf{I} \cap \mathbf{O}$.

4.3 Intuition Behind the Semantics of Circuits

In this work, we do not study temporal properties of circuits. Instead, we focus on steady-state analysis (i.e., DC analysis, in the terminology of SPICE), where the circuit's operating points are equilibrium points which correspond to solutions of the systems of equations modeling the circuit that do not change with time [Kundert, 1995; Hałgas, 2023]. In particular, we propose various transistor-level circuit modeling approaches, where circuits' steady-states are symbolically defined. This level of abstraction makes it possible to analyze some low-level properties of interest (e.g., the reliability of the components of a circuit) without dealing with the complexity of the layout level.

In order to understand which circuit properties should be used to define its behavior, we first take a look at simple circuit examples to help us refine our abstractions for reasoning about transistor level circuit descriptions. Later we present the notations used in the formalism, before we go through each of the semantics variants.

The behavior of any circuit can be understood by breaking it down into its constituting components (i.e., devices). To start simple, we consider the two building blocks on an Inverter circuit—a PMOS device and an NMOS device in isolation—as shown in Figure 4.12. We also consider the devices to be ideal, with a null operating threshold voltage ($V_t = 0\,\text{V}$). We vary the input voltage $\mathcal{V}(\texttt{A})$ between $\mathcal{V}(\texttt{GND}) - 1\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) + 1\,\text{V}$. The simulation output is shown in Figure 4.13. What we obtain is a constant voltage value for both pins `Z1` (at $\mathcal{V}(\texttt{VDD})$) and `Z2` (at $\mathcal{V}(\texttt{GND})$). However, the *status* of `Z1` and `Z2` changes for some values of $\mathcal{V}(\texttt{A})$. When a device is on, the drain (`Z1` or `Z2`) is *consistently* connected to the source (`VDD` or `GND`), while when it is off, the drain is floating. When the net is floating, the simulator keeps it at the same voltage as the only reachable supply. This, in our case, is coherent with the fact that no current *flows* through the device (i.e., between source and drain) when the device is off.

The Inverter circuit is made by creating a connection between `Z1` and `Z2`, named `Z` (Figure 4.14). The corresponding behavior can be drawn from individual components' behaviors from Figure 4.13. When either one of `Z1` and `Z2` is floating, the outcome of `Z` is that of the non-floating (i.e., consistent) net. When both `Z1` and `Z2` are consistent, a short-circuit is created (i.e., current leakage), with the maximum current value being attained at $\mathcal{V}(\texttt{Z}) = \mathcal{V}(\texttt{A}) = 0.5\,\text{V}$. The voltage value of `Z` gradually changes from $1\,\text{V}$ to $0\,\text{V}$ as $\mathcal{V}(\texttt{A})$ increases (Figure 4.15).

Abstracting the circuit's behavior must preserve information about net status (consistent, floating, short-circuit). When it comes to voltage modeling, abstractions can be made, for example by associating symbolic values to intervals. Apart from that, circuit semantics need to be defined in order to describe the global circuit behavior from components' semantics and their connections.

4.4 First Attempt: Oriented Signal Semantics ($\mathcal{S}^o$)

The transition from knowing the behavior of each of the devices composing a circuit (Figures 4.12 and 4.13) to constructing its global behavior (Figures 4.14 and 4.15) is what inspired our earliest attempt to circuit modeling. The main idea was to encode the behavior of each of the devices independently, and then defining the *rules* to merge information, e.g., the voltage of a net is defined by the wires it is connected to. We refer to this approach as oriented signal semantics, and we denote it $\mathcal{S}^o$.

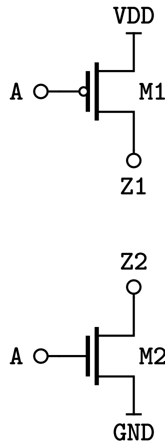


Figure 4.12: Inverter circuit components. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$.

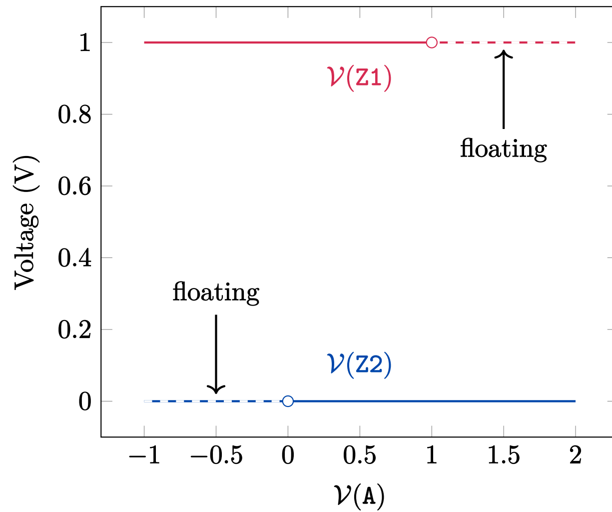


Figure 4.13: Drain values (Z1 and Z2) of devices M1 and M2 from Figure 4.12.

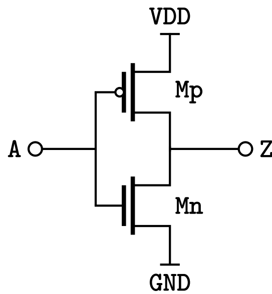


Figure 4.14: (Reminder) Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$.

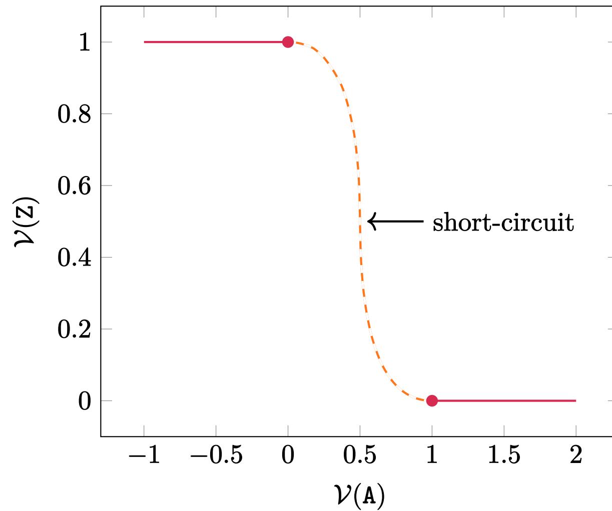


Figure 4.15: Inverter circuit (Figure 4.14) simulation output.

In $\mathcal{S}^o$, the modeling of devices considered is switch-based. So, this first attempt led to switch-based semantics, with particular hypotheses. As we show later in this section, the idea of constructing the circuit behavior from its components only works on circuits which comply with the strong assumption that information flows in a specific direction. It is not reasonable to apply on some circuit topologies, namely ones which have electrical paths on which information does not flow in a single direction. We illustrate these limitations with examples, later, in Section 4.4.2. In fact, oriented signal semantics may not be useful in real-life. We only present it here as it was our first attempt to model the steady states of a circuit.

Our initial assumption is that, in a given circuit, current can only flow in one direction—from source to drain. Such assumption covers, but is not limited to, complementary MOSFET (CMOS) designs. CMOS is a widely used design technique. It consists in using complementary pairs of PMOS and NMOS types to implement logic functions. In a CMOS design, transistors are 'oriented' in such a way that information (i.e., electrical signal) flows from source to drain. The corresponding semantics can be broken down into semantics rules, where each rule states the clauses to encode a

specific aspect of circuits. We present the intuition behind $\mathcal{S}^o$ in the following section, but keep the formalism of the semantics in Chapter B, as it is not essential for following the rest of this chapter.

4.4.1 Intuition

Initially, we explored a circuit modeling approach where information is propagated through the circuit in specific directions. This is inspired by the behavior of CMOS circuitry. Some previously shown examples are the Inverter (Figure 3.2), buffer (Figure 3.7), and AND (Figure 4.10) circuits. We have seen that, in these circuits, information flows from source to drain, and that some 'merge reasoning' can be applied on the nets where information 'meets' so that the resulting voltage can be defined. In this regard, the Inverter circuit can be viewed as two separate devices operating independently, and whose outputs are merged to construct the output voltage. An operation of circuit slicing on the basis of neighborhood connections is performed: all nets connecting two devices by source or drain are cut, creating thus *virtual* net naming. Gate nets, however, are preserved. Figure 4.17 illustrates this operation on the Inverter circuit (Figure 4.16).

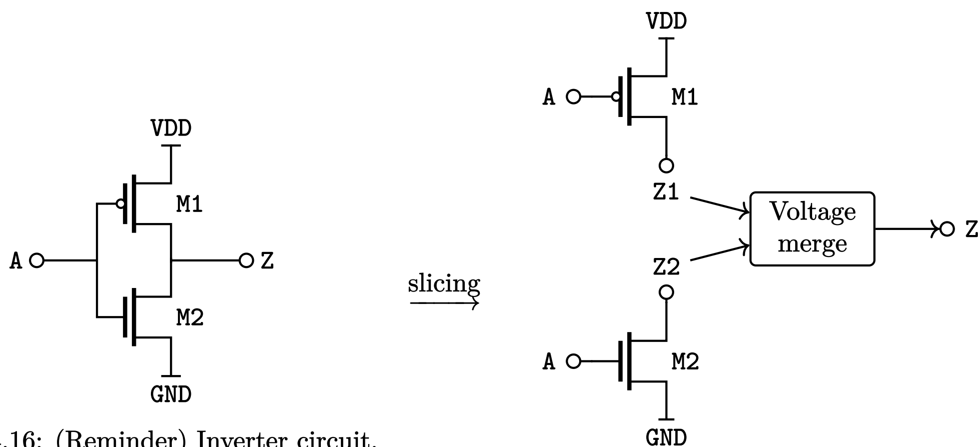


Figure 4.16: (Reminder) Inverter circuit.

Figure 4.17: Inverter circuit (Figure 4.16) output construction with the oriented signal abstraction.

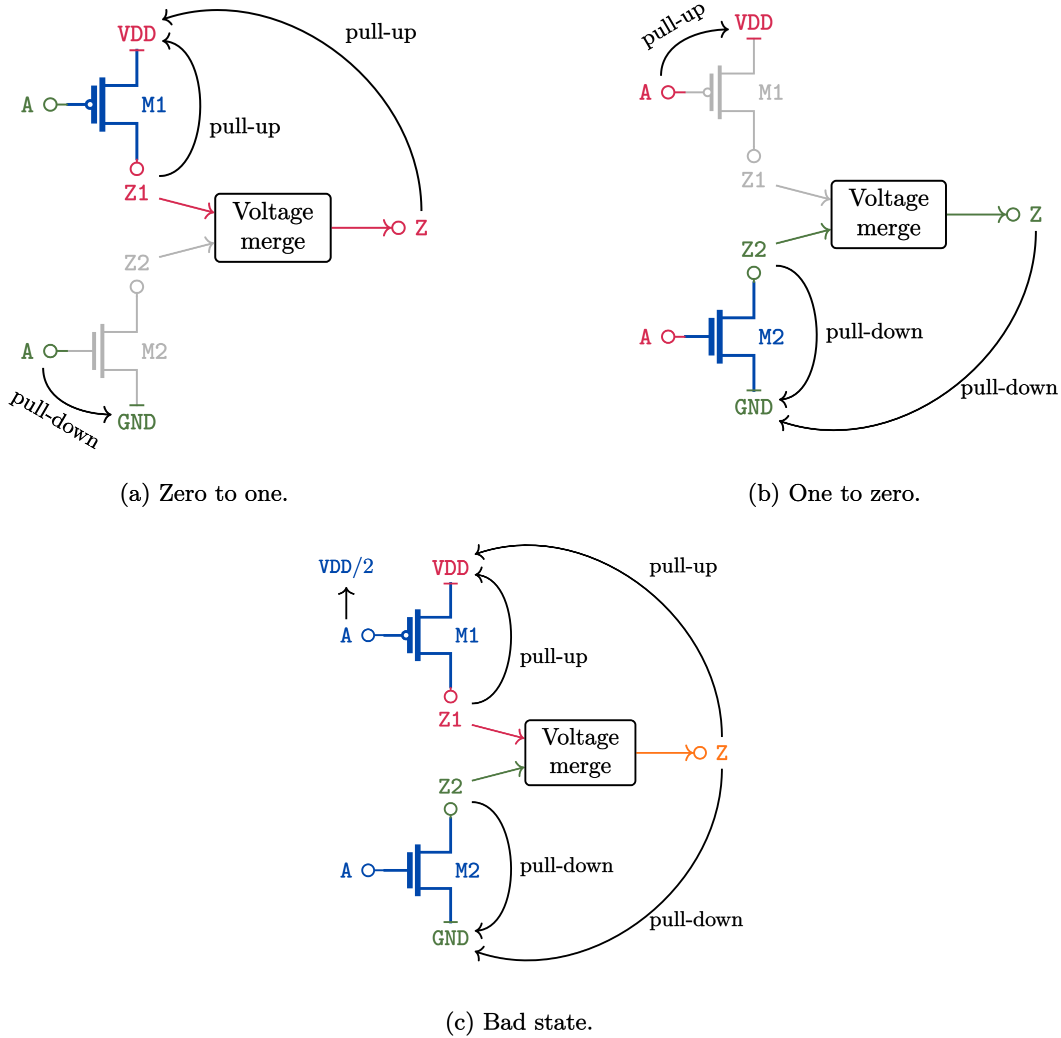
Voltages of Z1 and Z2 can be separately computed respectively based on the state of transistors M1 and M2. Figure 4.18 shows how such abstraction can be used to determine the voltage of Z. In state (a), the high voltage of Z1 overrides the floating net Z2, hence the high voltage of Z. Similarly, a grounded Z2 overrides the floating Z1 in state (b). Intermediate input voltages, like VDD/2 in state (c), activates both devices, resulting in a pull-up of Z1 to a high voltage and a pull-down of Z2 to the ground. The outcome is a short-circuited output Z.

We model net voltages on a discrete (as opposed to continuous) support. We opt for abstracting actual voltage values into 'voltage levels', represented with integers. Supply values are represented with even numbers (with preserving the order), keeping thus odd numbers to represent voltages between every two supplies. Further details on voltage abstraction are provided in Section B.2. Transistor states are modeled according to the switch-based (on/off) abstraction.

Circuit semantics $\mathcal{S}^o$ were a mere first attempt which presents several limitations, presented in the following section. They are not used much in this work. Hence, the details of this functional-like encoding of circuit behavior are presented in Appendix B rather than in this chapter. The curious reader may want to look more in-depth into the details of $\mathcal{S}^o$, but they are not required to in order to be able to follow the remainder of the manuscript.

4.4.2 Limitations

Oriented semantics' model of short-circuits is limited, due to strong hypotheses on circuit topology. It only notices the short-circuit at the net where two different voltage values meet, and only ensures



(a) Zero to one.

(b) One to zero.

(c) Bad state.

Figure 4.18: Three states of the Inverter circuit. With $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$.

coherent voltage values on nets used as transistor gates.

Consider the NAND gate example of Figure 4.19. In the case where the voltages of nets A and B belong to $]\mathcal{V}(\texttt{GND}), \mathcal{V}(\texttt{VDD})[$, all four transistors are switched-on. Our oriented semantics properly infers that the output NAND is in short-circuit state, but incorrectly considers x to be connected only to GND. The short-circuit information does not back-propagate from drain to source on device M3. The circuit's behavior (in terms of device states) is still properly modeled as long as x is not used as the gate of some transistor.

Figure 4.20 depicts a variant of the NAND circuit, where a device (M5) is added and is driven by x. This circuit cannot be modeled properly with $\mathcal{S}^o$: both the voltage and status of the gate of M5 would be incorrect. The state (on/off) of M5 would be incorrect, too. Designs following the CMOS convention are not affected by this limitation, but many real-life circuits are not CMOS-compliant. Sometimes, a CMOS design ceases to be so, simply by adding a power-down mode to it, e.g., using an NMOS device as the power-down switch at the interface between a supply and a PMOS-based pull-up network. Oriented signal semantics does, in fact, not take into account the fact that transistor source and drain are symmetrical, as it considers information flows from source to drain.

In fact, since the oriented semantics encodes each net as a function of other nets without taking into consideration existing circular dependencies, we end up missing some information. As a solution, we propose an alternate encoding in a relational fashion, i.e., encoding a relation between

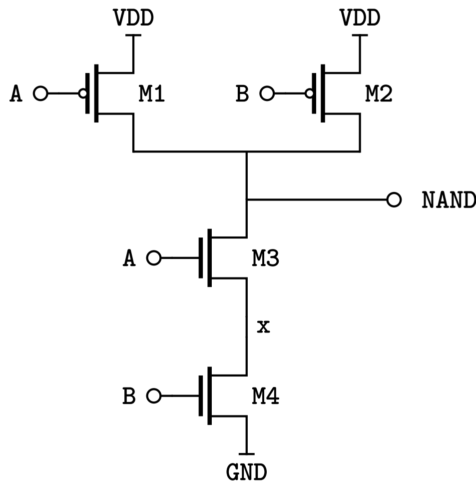


Figure 4.19: CMOS-based NAND gate.

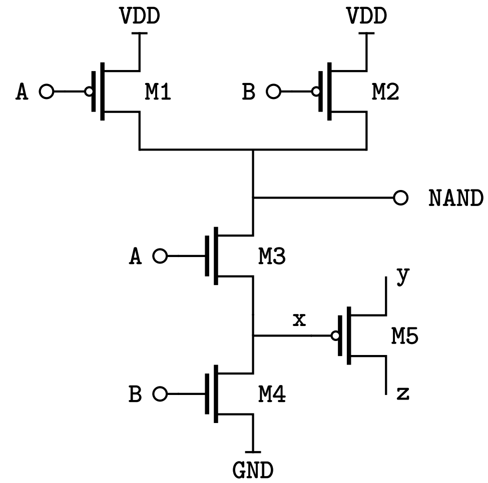


Figure 4.20: Variant of a non CMOS based NAND circuit.

pins instead of defining the drain as a function of gate and source. Such encoding would ensure information propagation among all nets that share the same electrical path. Accordingly, we define new switch-based semantics. To show the differences of the encoding flow between the two approaches, we consider the same NAND gate as in Figure 4.19. When applying the oriented semantics, transistor drains are computed as a function of gate and source values. A process of 'driver merging' uses these values to compute the resulting net values for nets `NAND` and `x`, see Figure 4.21. The virtual drain nets are denoted $\widehat{\mathrm{DRN}}(M)$ for each transistor M. They are represented with gray boxes in the drawing of Figure 4.21. Green boxes represent driver merging processes (R_{merge}), defined in Section B.5. Such functional encoding only ensures computation of what we consider to be relevant nets of the circuit; in particular, gate nets and output pins. In contrast, a relational encoding would take into consideration all dependencies between circuit nets. Therefore, it is more sensible. The essence of a relational switch-based semantics is depicted in Figure 4.22, considering the same NAND circuit. The details of such relational semantics are then presented in the next section.

Furthermore, as we show later in Section 5.1.3—where we study performance of different circuit semantics on synthetic benchmarks, oriented signal semantics is not advantageous in terms of scaling. The limitations mentioned make $\mathcal{S}^o$ really hard, or even uninteresting, to integrate in verification flows for electrical rule checking. Except part of the experimental results we present later in Chapter 5, we simply discard this modeling approach in the rest of this manuscript, so that we focus on better alternative models. This explains why $\mathcal{S}^o$ is excluded from Section 4.7's formal comparisons of circuit semantics.

We start with relational switch-based semantics, in the next section, where the stakes are to remain topology-agnostic, and where we use a continuous support for voltage modeling instead of the interval-based abstractions considered in our first attempt.

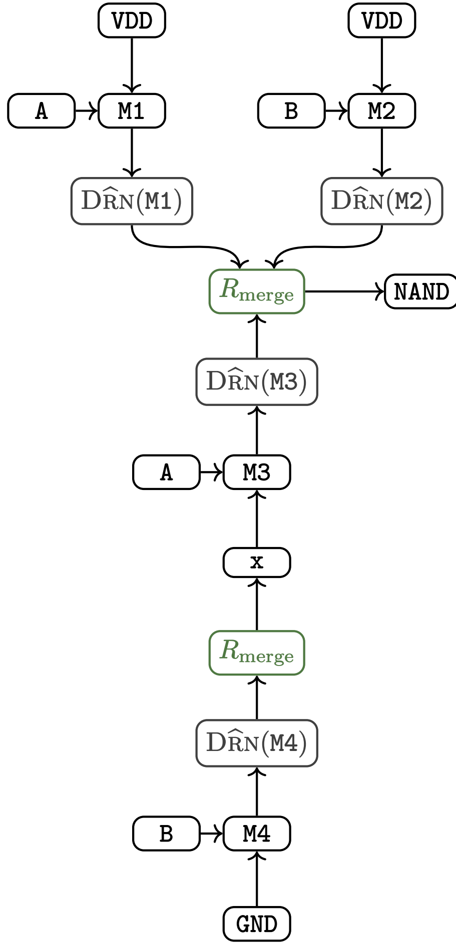


Figure 4.21: NAND gate (Figure 4.19) encoding flow using oriented signal semantics.

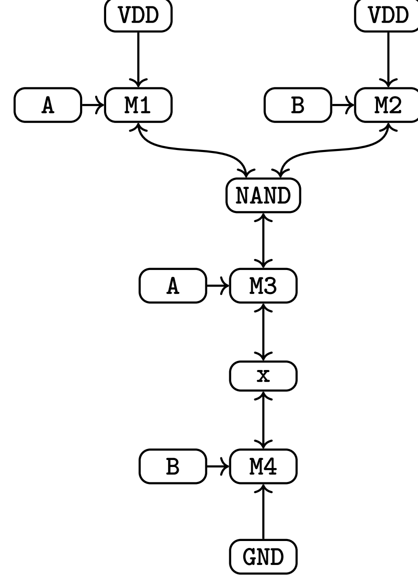


Figure 4.22: NAND gate (Figure 4.19) encoding flow using relational switch-based semantics.

4.5 Relational Switch-Based Semantics ($\mathcal{S}^s$)

We present circuit semantics where the logical clauses describe relations, rather than functions, between nets and devices in a circuit. We put no prior restriction on the circuit topology. Therefore, such semantics is made to cover all sorts of circuit topologies (including non-CMOS and analog circuits), which makes it more general than the oriented signal semantics.

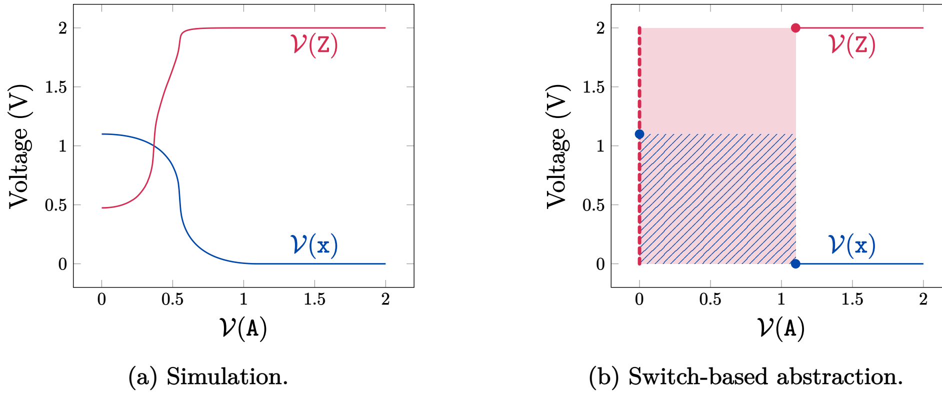
From a transistor-level circuit description and associated power supplies configuration, we generate an encoding in terms of logic formulas representing the circuit's steady-states. We restrict ourselves to logical formulas suitable for use with typical SMT solvers, i.e., first-order logic formulas involving predicates over finite domains, Boolean and numerical variables. We also restrict ourselves to the linear real arithmetic (LRA) theory of the solver. To each device type, we associate semantics rules defining its behavior, and each net is constrained based on its connectivity. Following LRA, constraints are created by adding variables together, or multiplying them by constants, but not multiplying them together.

4.5.1 Switch-Based Abstraction Put in Perspective Against Simulation

A net's voltage can be non-deterministic according to the switch-based abstraction [Randal E Bryant, 1984; Zwerger and Graeb, 2012]. Consider, for example, the buffer circuit from Figure 3.7 (page 25). Having $\mathcal{V}(\texttt{A}) = \mathcal{V}(\texttt{GND})$ results in $\mathcal{V}(\texttt{x}) = \mathcal{V}(\texttt{VDDL})$, which, according to the switch-based abstraction turns both devices M3 and M4 on. In such case, the output Z is associated an interval of possible values. A more precise approach, however, would consider that devices M3 and M4 are in

some intermediate state between on and off. Solving the systems of equations modeling the physical laws behind the circuit's behavior (e.g., using SPICE) actually leads to a solution where the voltage of `Z` is deterministically computed, as a single value, not as an interval.

In simulation, a common way of obtaining the circuit's set of steady-states is to vary combinations of the input vector until the whole state space (in terms of the input vector) is covered. Equilibrium points found by the simulation engine are then retrieved from nets' voltage traces. Accordingly, we simulate the buffer circuit example (Figure 3.7) using the LTSPICE program [Analog Devices, 2024], by varying the input voltage, with supplies being set as $\mathcal{V}(\texttt{VDDL}) = 1.1\,\text{V}$ and $\mathcal{V}(\texttt{VDDH}) = 2\,\text{V}$, and with considering the threshold voltage to be null ($V_t = 0\,\text{V}$) for all devices. Figure 4.23a plots the voltage values of nets `x` and `Z` with respect to that of `A`, as obtained from simulation traces. For $\mathcal{V}(\texttt{A}) = \mathcal{V}(\texttt{GND})$, the equilibrium voltage of `Z` is found to be around 0.47 V. In the switch-based abstraction, $\mathcal{V}(\texttt{Z})$ is allowed an interval represented with the vertical red dashed line of Figure 4.23b. This is also the case for $\mathcal{V}(\texttt{A}) \in]\mathcal{V}(\texttt{GND}), \mathcal{V}(\texttt{VDDL})[$ (red-colored area). While $\mathcal{V}(\texttt{x})$ is deterministic for $\mathcal{V}(\texttt{A}) = \mathcal{V}(\texttt{GND})$, it is abstracted with the interval $]\mathcal{V}(\texttt{GND}), \mathcal{V}(\texttt{VDDL})[$ for $\mathcal{V}(\texttt{A}) \in]\mathcal{V}(\texttt{GND}), \mathcal{V}(\texttt{VDDL})[$ (blue dashed area). Only for input values $\mathcal{V}(\texttt{A}) \geq \mathcal{V}(\texttt{VDDL})$ the valuations of both voltages of `x` and `Z` in the switch-based abstraction are aligned with SPICE.



(a) Simulation. (b) Switch-based abstraction.

Figure 4.23: Nets voltages of the buffer circuit (Figure 3.7, page 25) with respect to $\mathcal{V}(\texttt{A})$.

What we can draw from this example is that the voltage of a net depends on the other nets it is dynamically connected to. When a net n_1 is connected to a single other net n_2 whose voltage is 'well defined', net n_1 can then only have the same voltage as n_2 (e.g., $\mathcal{V}(\texttt{x})$ when $\mathcal{V}(\texttt{A}) = 0\,\text{V}$, and $\mathcal{V}(\texttt{x})$ and $\mathcal{V}(\texttt{Z})$ when $\mathcal{V}(\texttt{A}) \geq 1.1\,\text{V}$ in Figure 4.23). However, when the net is connected to multiple other nets, its voltage depends on the exact voltages of its neighbors, which leads to multiple case scenarios to consider (e.g., $\mathcal{V}(\texttt{x})$ and $\mathcal{V}(\texttt{Z})$ when $\mathcal{V}(\texttt{A}) \in]0\,\text{V}, 1.1\,\text{V}[$ in Figure 4.23).

We detail, in the following section, how we define and then generalize the relations which enable us to infer voltage abstractions among the circuit nets.

4.5.2 Switch-Based Semantics for Transistors

Circuit semantics are build as a conjunction of different semantics rules, as we present in the following sections.

4.5.2.1 Transistor States

Relational circuit semantics are introduced based on the switch-based abstraction of transistors. PMOS transistors are switched-on when a negative voltage difference is applied between the gate and source (or drain), while a positive voltage difference activates NMOS transistors. Transistor states are defined by the predicate $\mathcal{O}n\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$ in $\mathcal{O}n^{\texttt{PMOS}}_{\text{def}}$ for PMOS devices, and in $\mathcal{O}n^{\texttt{NMOS}}_{\text{def}}$ for NMOS devices.

$$\begin{aligned} \mathcal{O}n\colon \mathbf{D}_{\mathtt{PMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \big(\mathcal{V}_{\mathrm{G}}(X) < \mathcal{V}_{\mathrm{S}}(X)\big) \vee \big(\mathcal{V}_{\mathrm{G}}(X) < \mathcal{V}_{\mathrm{D}}(X)\big) \end{aligned} \tag{$\mathcal{O}n^{\mathtt{PMOS}}_{\mathrm{def}}$}$$

$$\begin{aligned} \mathcal{O}n\colon \mathbf{D}_{\mathtt{NMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \big(\mathcal{V}_{\mathrm{G}}(X) > \mathcal{V}_{\mathrm{S}}(X)\big) \vee \big(\mathcal{V}_{\mathrm{G}}(X) > \mathcal{V}_{\mathrm{D}}(X)\big) \end{aligned} \tag{$\mathcal{O}n^{\mathtt{NMOS}}_{\mathrm{def}}$}$$

4.5.2.2 Net Voltages

Supply and input voltages are determined by the environment. We use the exact same constraints $\mathcal{R}_{\mathbf{S}}$ and $\mathcal{R}_{\mathbf{I}}$ defined previously (page 52). Conversely, voltages of internal nets and outputs are constrained with respect to their respective neighbors. The key constraint defining a net's voltage relationally is formalized with the $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ rule, which is an abstraction of Kirchhoff's current law (the proof for this will be shown later in Lemma 4.7.1, page 74). It asserts that current flows in the net (i.e., the voltage of the net is at least less than one of its dynamically connected neighboring nets) if and only if it flows out of the net (i.e., the voltage of the net is at least greater than one of its dynamically connected neighboring nets). The absence of current corresponds to a state where the net is only dynamically connected to nets all having the same voltage, including the case where it is connected to no other net.

$$\bigwedge_{self \in \mathbf{N} \setminus (\mathbf{S} \cup \mathbf{I})} \left(\begin{array}{ll} \displaystyle\bigvee_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} & \mathcal{O}n(X) \wedge \big(\mathcal{V}(n) < \mathcal{V}(self)\big) \\ \Leftrightarrow \displaystyle\bigvee_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} & \mathcal{O}n(X) \wedge \big(\mathcal{V}(self) < \mathcal{V}(n)\big) \end{array} \right) \tag{$\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$}$$

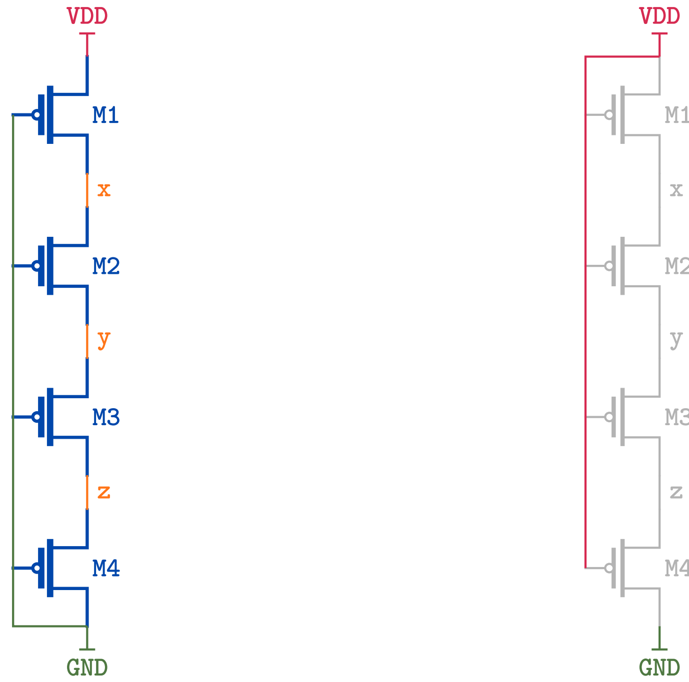
When a net is floating (i.e., transitively connected to no supply or input), its voltage value is not constrained by its neighbors. However, unrealistic values (e.g., a net having a voltage of 220 V in a floating state) can be avoided by supplementing the semantics with more constraints (Section 4.5.2.3).

By construction, for each net, the $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ formula evaluates to $\top$ in two possible ways. The first (1) is by having both sides of the equivalence evaluate to $\top$ (i.e., $\top \Leftrightarrow \top$). This corresponds to the case where current flows through the net. The second (2) is by having both sides evaluate to $\bot$. In this case, no current flows through the net. $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ can be decomposed in a way that encodes that 'either current flows through a net or no current is flowing'—with defining the conditions for each case (as a logical disjunction instead of an equivalence). Such decomposition will be introduced later in Lemma 4.7.1 (page 74).

In case (1), the rule enforces that the voltage value of the net is strictly between the voltage values of its neighbouring nets, if the net is in short-circuit. This case corresponds to the existence of a voltage drop across some of the electrical paths the net is part of. We say that current is flowing thereof. The voltage value of a net belongs to the open interval defined by the minimum and maximum voltage value of the neighbouring nets it is dynamically connected to. Strict voltage monotony is ensured along the nets that are part of any short-circuited electrical path (i.e., a path composed of dynamically switched-on devices).

In case (2), the voltage value of the net is equal to that of all of its dynamically connected neighbouring nets. This case corresponds to the absence of current through all electrical paths the net is part of. This happens either when all neighbouring devices are switched-off (i.e., the net is floating), or when no voltage drops between the net and the switched-on devices in its neighbourhood (e.g., the net is connected to one and only one supply).

To illustrate how $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ works, we consider a simple circuit which consists of four PMOS transistors connected is series by source-drain, with a ground (`GND`) in one end of the chain, and a supply (`VDD`) on the other end (Figure 4.24). Devices `M1`, `M2`, `M3`, and `M4` are activated when all their gates are grounded. This leads to a short-circuit on nets `x`, `y`, and `z` (Figure 4.24a). $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ ensures that $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{z}) < \mathcal{V}(\texttt{y}) < \mathcal{V}(\texttt{x}) < \mathcal{V}(\texttt{VDD})$, which emulates non-null voltage drops (and thus non-null current) across each device's source-drain. When all transistor gates are connected to the supply, all devices are turned-off, and all source-drain connections are cut (Figure 4.24b). Nets `x`, `y`, and `z` are therefore floating. At this stage, their voltages are not constrained (they could take any value, and the circuit semantics would remain satisfiable). The voltage integrity of floating nets in our modeling must be preserved in some way: a net's voltage cannot go beyond its interval of reachable supplies. The next section defines such constraints.



(a) All gates connected to `GND`, resulting in all transistors being switched-on, and a short-circuit along the `GND`-to-`VDD` path.

(b) All gates connected to `VDD`, resulting in all transistors being switched-off, and all nets in the `GND`-to-`VDD` path to be floating.

Figure 4.24: Two device gate configurations of the PMOS transistors chain. With $\mathcal{V}(\texttt{GND}) + V_t < \mathcal{V}(\texttt{VDD})$.

4.5.2.3 Net Static Domain

Every net must be bounded by the supplies it can possibly reach, which are given by some function ReachSuppl (ReachSuppl$_{\text{def}}$, page 48). We constrain the static voltage domain, for every internal or output net, as being bounded by the minimum and maximum (statically) reachable supplies, encoded as $\mathcal{R}_{\substack{\text{static}\\\text{domain}}}$.

$$\bigwedge_{self \in \mathbf{N} \setminus (\mathbf{S} \cup \mathbf{I})} \left(\begin{array}{l} \mathcal{V}(self) \geq \min\limits_{s \,\in\, \textsc{ReachSuppl}(self)} \mathcal{V}(s) \\ \wedge\ \mathcal{V}(self) \leq \max\limits_{s \,\in\, \textsc{ReachSuppl}(self)} \mathcal{V}(s) \end{array} \right) \qquad (\mathcal{R}_{\substack{\text{static}\\\text{domain}}})$$

4.5.2.3.1 Circuit Formula

Finally, the circuit semantics, denoted $\mathcal{S}^s$ (for switch-based semantics), is defined as the conjunction of previously introduced rules ($\mathcal{S}^s{}_{\text{def}}$).

$$\mathcal{S}^s \triangleq \mathcal{R}_{\substack{\text{local}\\\text{voltage}}} \wedge \mathcal{R}_{\substack{\text{static}\\\text{domain}}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}^s{}_{\text{def}})$$

Using an SMT solver, like Z3 [De Moura and Bjørner, 2008], we can check, for example, whether some safety property $\mathcal{P}$ (i.e., there is no possible error $\mathcal{E} = \neg\mathcal{P}$) is verified while preserving the circuit semantics $\mathcal{S}^s$. Property $\mathcal{P}$ is verified by $\mathcal{S}^s$ when the expression $\mathcal{S}^s \wedge \neg\mathcal{P}$ (with $\neg\mathcal{P}$ being an error condition) is unsatisfiable (or UNSAT)—meaning that there exists no state where $\mathcal{P}$ is not verified. We present applications for which we use this approach in Chapter 5.

4.5.3 Extension of Switch-Based Semantics to Other Devices

$\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ only covers cases where the $\textsc{Neighbors}(n)$ of a net n are all transistors. It can nevertheless easily be extended to cover other device types, namely, resistors and diodes. Recall that, as our approach reasons about the circuit in terms of steady-states, it is not possible to extend the semantics to handle capacitors and coils, as their non trivial behavior is intrinsically transient. We could, however, model capacitor terminals as always-disconnected and that of coils as short-circuited, since this corresponds to their behavior in steady-states.

No additional predicates are defined for resistors and diodes, as the only information relevant about them is whether or not current flows through them. Such information can be added by extending $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$. To do so, we extend predicate $\mathcal{O}n\colon \mathbf{D} \longrightarrow \mathbb{B}$, initially restricted to transistors, to all elements of $\mathbf{D}$.

Diodes are modeled such that they are on when the anode's voltage is greater than the cathode's voltage, which is a simplification of the condition of the existence of a non-null current flowing through a diode. The extension of the definition of predicate $\mathcal{O}n$ in $\mathcal{O}n^{\texttt{DIO}}_{\text{def}}$ encodes such abstraction.

$$\begin{aligned}\mathcal{O}n\colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{B}\\ X &\longmapsto \mathcal{V}(\textsc{Anode}(X)) > \mathcal{V}(\textsc{Cathode}(X))\end{aligned} \qquad (\mathcal{O}n^{\texttt{DIO}}_{\text{def}})$$

Resistors, on the other hand, are considered to behave as 'always on' devices ($\mathcal{O}n^{\texttt{RES}}_{\text{def}}$).

$$\begin{aligned}\mathcal{O}n\colon \mathbf{D}_{\texttt{RES}} &\longrightarrow \mathbb{B}\\ X &\longmapsto \top\end{aligned} \qquad (\mathcal{O}n^{\texttt{RES}}_{\text{def}})$$

The extended version of the circuit semantics is obtained from the same definition in $\mathcal{S}^s{}_{\text{def}}$, as the predicate $\mathcal{O}n$ is now implemented for diodes and resistors too.

4.5.4 Variant of Switch-Based Semantics with Non-null Threshold ($\mathcal{S}^t$)

The initial definition of the predicate $\mathcal{O}n$ ($\mathcal{O}n^{\texttt{PMOS}}_{\text{def}}$, $\mathcal{O}n^{\texttt{NMOS}}_{\text{def}}$, and $\mathcal{O}n^{\texttt{DIO}}_{\text{def}}$) considers a null threshold value for transistor and diode activation. For example, any negative value $\mathcal{V}_{\text{GS}}(X)$ or $\mathcal{V}_{\text{GD}}(X)$ forces the predicate $\mathcal{O}n(X)$ for a PMOS transistor X to be $\top$, whereas a realistic behavior involves a non-null threshold in the activation condition. For simplification of the formalism, the threshold voltage—denoted V_t—is considered to be the same for all transistors and diodes, but an implementation can easily use a different V_t value for each device if needed. Typically, the threshold voltage is between 0.1 V and 0.3 V [Sedra and Smith, 2010].

Definitions $\mathcal{O}n^{t}{}^{\texttt{PMOS}}_{\text{def}}$, $\mathcal{O}n^{t}{}^{\texttt{NMOS}}_{\text{def}}$, and $\mathcal{O}n^{t}{}^{\texttt{DIO}}_{\text{def}}$ are improved versions—taking into account the value of the threshold voltage—of Definitions $\mathcal{O}n^{\texttt{PMOS}}_{\text{def}}$, $\mathcal{O}n^{\texttt{NMOS}}_{\text{def}}$, and $\mathcal{O}n^{\texttt{DIO}}_{\text{def}}$, constraining the $\mathcal{O}n$ predicate. For resistors, we rename the on predicate for notation consistency, but its definition remains unchanged ($\mathcal{O}n^{t}{}^{\texttt{RES}}_{\text{def}}$).

$$\begin{aligned} \mathcal{O}n^t : \mathbf{D}_{\texttt{PMOS}} &\rightharpoonup \mathbb{B} \\ X &\longmapsto \big(\mathcal{V}_{\text{GS}}(X) < -V_t\big) \vee \big(\mathcal{V}_{\text{GD}}(X) < -V_t\big) \end{aligned} \qquad (\mathcal{O}n^{t\,\texttt{PMOS}}_{\text{def}})$$

$$\begin{aligned} \mathcal{O}n^t : \mathbf{D}_{\texttt{NMOS}} &\rightharpoonup \mathbb{B} \\ X &\longmapsto \big(\mathcal{V}_{\text{GS}}(X) > V_t\big) \vee \big(\mathcal{V}_{\text{GD}}(X) > V_t\big) \end{aligned} \qquad (\mathcal{O}n^{t\,\texttt{NMOS}}_{\text{def}})$$

$$\begin{aligned} \mathcal{O}n^t : \mathbf{D}_{\texttt{DIO}} &\rightharpoonup \mathbb{B} \\ X &\longmapsto \mathcal{V}_{\text{AK}}(X) > V_t \end{aligned} \qquad (\mathcal{O}n^{t\,\texttt{DIO}}_{\text{def}})$$

$$\begin{aligned} \mathcal{O}n^t : \mathbf{D}_{\texttt{RES}} &\rightharpoonup \mathbb{B} \\ X &\longmapsto \top \end{aligned} \qquad (\mathcal{O}n^{t\,\texttt{RES}}_{\text{def}})$$

We re-define local constraints on the circuit nets, using the $\mathcal{O}n^t$ predicate, as shown in $\mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}}$.

$$\bigwedge_{self \in \mathbf{N} \smallsetminus (\mathbf{S} \cup \mathbf{I})} \left(\begin{array}{c} \displaystyle\bigvee_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(n) < \mathcal{V}(self)\big) \\ \Leftrightarrow \displaystyle\bigvee_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(self) < \mathcal{V}(n)\big) \end{array} \right) \qquad (\mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}})$$

The corresponding extension of the semantics, denoted $\mathcal{S}^t$, is defined by $\mathcal{S}^t{}_{\text{def}}$. The only change compared with the definition of $\mathcal{S}^s$ ($\mathcal{S}^s{}_{\text{def}}$) is the rule locally constraining net voltages, $\mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}}$.

$$\mathcal{S}^t \triangleq \mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}} \wedge \mathcal{R}_{\substack{\text{static}\\\text{domain}}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}^t{}_{\text{def}})$$

4.6 Quantitative Semantics ($\mathcal{S}^q$)

The semantics introduced so far ($\mathcal{S}^o$, $\mathcal{S}^s$, and $\mathcal{S}^t$) are based on the switch abstraction and are non-quantitative with respect to modeling voltage and current values. This section defines a new semantics, denoted $\mathcal{S}^q$, where the quantitative aspect is important.

4.6.1 Intuition

The purpose of developing a quantitative circuit modeling is to be more precise about voltage valuations, namely in the case of short-circuits or floating nets. Quantitative semantics serves the study of quantitative properties, for which the quality of the analysis largely relies on model precision. Examples of quantitative properties we are interested in are "what is the maximum voltage between two given nets?" and "does the current through a device exceed its allowed upper bound?"

In our initial attempts, we tried modeling voltage drops across transistors with resistors of different resistance values, based on whether the transistor (M) is turned on or off. Basically, a high resistance $r_{\texttt{OFF}}$ is associated with $\mathcal{O}n^t(M) = \bot$, and a low resistance $r_{\texttt{ON}}$ is associated with $\mathcal{O}n^t(M) = \top$. This modeling idea, as well as other variants, were not successful in practice, but

they were the first steps which led to the quantitative semantics finally adopted, which we introduce here. We provide more details and counterexamples on why such modeling attempts did not work out, in Appendix C.

There are two takeaways from modeling trials previously conducted. The first one, and the most important, is to ensure that all circuit formulas following $\mathcal{S}^q$ are satisfiables. For instance, relations between circuit variables must be continuous (solutions correspond to the space of intersections between all variables). The second one is that the encoding must be limited to the LRA theory, so that formula solving is more scalable.

The common idea among previous trials is to model transistors as resistors based on their state. In its simplest form, it consists in associating a relatively high resistance $r_{\texttt{OFF}}$ to a switched-off device, and a relatively low resistance $r_{\texttt{ON}}$ to a switched-on device. The fact that continuous relations are required in order for a solution to exist can be illustrated on a simple circuit example, shown Figure 4.25.

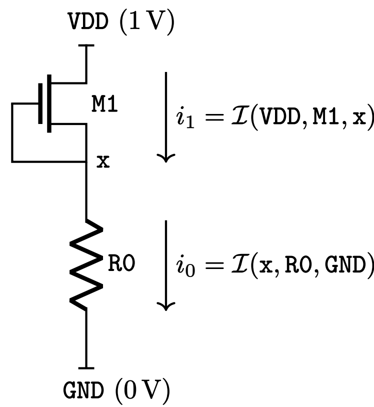


Figure 4.25: Diode-connected transistor in series with a resistor.

The circuit is made of one diode-connected NMOS transistor (`M1`), put in series with a resistor `R0` whose resistance value is r_0. One trivial property of this circuit, is that $i_0 = i_1$, according to Kirchhoff's current law. To find the value of $\mathcal{V}(\texttt{x})$ for which Kirchhoff's current law is respected on net `x`, and for which also Ohm's law is respected on devices `R0` and `M1`, we can first study each of the two devices in isolation, and then identify the intersection between their individual solutions (that would be the solution of the circuit as a whole).

Figure 4.26 plots I-V relations for devices `R0` and `M1`. The I-V relation across `R0` is made of a single segment determined with Ohm's law. For `M1`, however, we need two segments to define such relation. One segment describes the I-V relation when the transistor is switched-off (according to the switch-based abstraction, and taking into account the threshold voltage V_t). When the device is switched-on, the I-V relation is described with a second segment. Here, choices of the exact values of $r_{\texttt{ON}}$ and $r_{\texttt{OFF}}$ do matter. In our example, these values are chosen to show a case where the solutions of each device in isolation do not intersect. It is a case where no steady-state of the circuit (according to these modeling choices) is possible: the semantics of the circuit is unsatisfiable. In order for the semantics to be satisfiable, there must be a non-empty intersection between I-V relations. This is the fundamental issue encountered while designing previous circuit semantics shown in Chapter C.

In this section, we make sure the relations introduced in quantitative circuit semantics $\mathcal{S}^q$ ensure continuous relations between current and voltage variables when transitioning from one transistor state to another.

The quantitative model we create is based on linear approximations of I-V device characteristics. It approximates the relation between current ($\mathcal{I}_{\text{SD}}$) and voltage ($\mathcal{V}_{\text{GS}}$ and $\mathcal{V}_{\text{GD}}$) in each region of operation linearly. This choice represents a compromise between precision and scalability, since an SMT solver deals better with formulas which consist of relations between linear constraints instead of quadratic ones (actual I-V characteristics curves are quadratic).

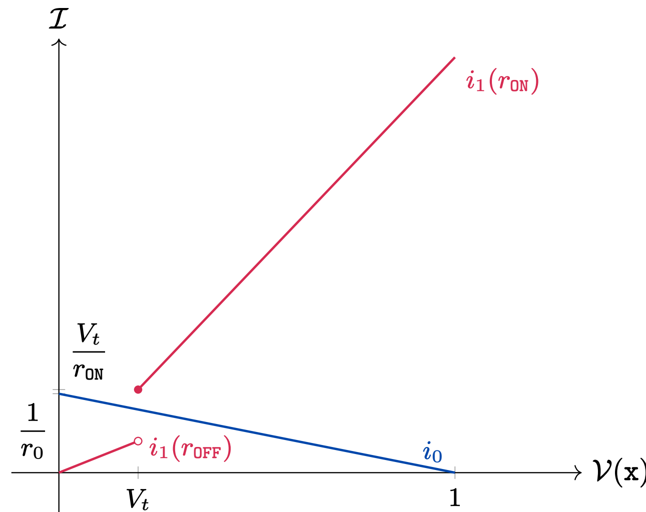


Figure 4.26: I-V relations of devices `R0` (in blue) and `M1` (in red) of the circuit of Figure 4.25, where the space of solutions of the circuit is empty.

In order to provide an idea of the expected output of a working quantitative semantics, we proceed by showing the results obtained by $\mathcal{S}^q$—which we present in detail later in this section. Instead of showing examples from real-life circuits—which tend to be complicated—we choose to illustrate this on our usual buffer circuit (Figure 3.7, page 25), by considering the input-output voltage difference. Simulation output is shown in Figure 4.27a. Figure 4.27b shows the output of our quantitative modeling. Figure 4.27c and Figure 4.27d, show, the absolute voltage difference between input and output, respectively, for simulation and for quantitative semantics.

In the following sections, we present I-V characteristics from which we derive our abstractions. We show how it works on simple examples, and we assess modeling precision using SPICE as a reference.

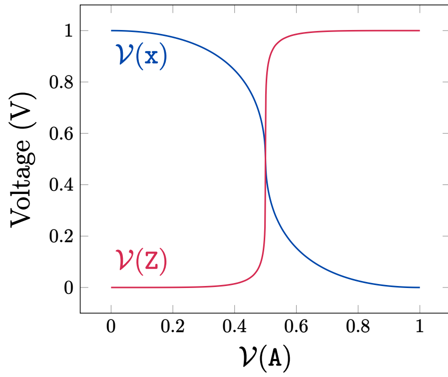
4.6.2 Device Characteristics

Relational switch-based circuit semantics introduced in Section 4.5 does not give precise quantitative information about circuits. It uses an abstract voltage modeling for nets in short-circuit. For example, the buffer circuit accepts any value in $]\mathcal{V}(\mathtt{GND}), \mathcal{V}(\mathtt{VDDH})[$ as the voltage of `Z` in case of short-circuit. These non-deterministic semantics properly over-approximate the actual behavior, but lack precision: a more precise analysis could yield a narrower interval for the value of `Z`. A more precise approximation of voltages is indeed required to tackle properties such as EOS, which are intrinsically quantitative.

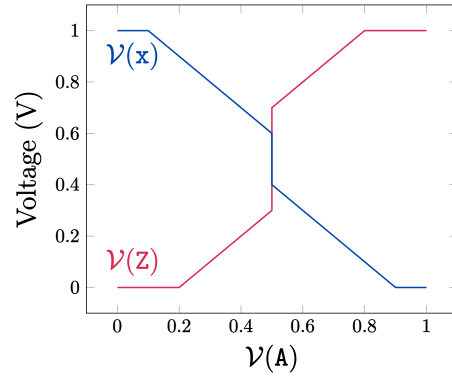
To better model voltage values in similar cases, we aim at using quantitative semantics where voltage values are deterministically assigned. A close work has been presented by Miller and Brewer [2013], where the goal is to determine operational bounds of circuit steady-states. Their approach uses SAT solving to check the reachability of circuit steady-states for a given circuit description and a set of imposed constraints. Current and voltage parameters are constrained, each, by a lower and upper bound, and, on every net, Kirchhoff's current law is applied.

In this work, we introduce a similar approach, by taking into account device characteristics equations, approximated and formalized in terms of semantics rules. In contrast to the SAT solving based approach of Miller and Brewer [2013], we use an SMT solver. This allows us to directly write formulas using actual voltage and current values, encoded as rational numbers in the solver, avoiding thus the need for Boolean encoding of numerical variables (bit-blasting). The same voltage function $\mathcal{V} \colon \mathbf{N} \longrightarrow \mathbb{V}$ is used to refer to a net's voltage.

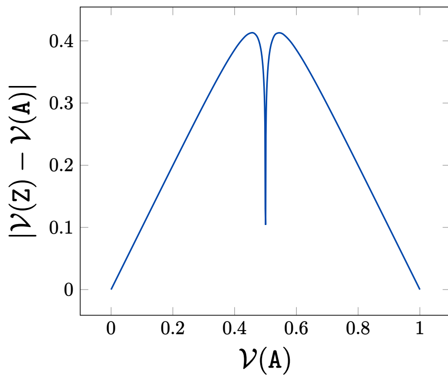
In contrast to switch-based semantics—where we only consider the presence or absence of current and its direction (see $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$, page 60), quantitative semantics associates a value to such current.



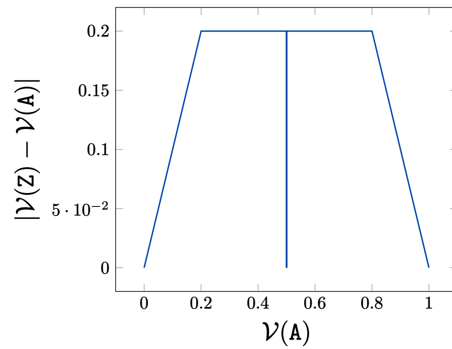
(a) SPICE simulation.



(b) Quantitative semantics.



(c) SPICE simulation's input-output margin.



(d) Quantitative semantics' input-output margin.

Figure 4.27: Comparison of nets' voltage modeling of the buffer circuit (Figure 3.7, page 25) with respect to $\mathcal{V}(\mathtt{A})$ between SPICE and quantitative semantics $\mathcal{S}^q$.

4.6.2.1 Kirchhoff's Current Law

For every net of the circuit, currents are constrained by Kirchhoff's current law. On every internal net or output $self \in \mathbf{N} \setminus (\mathbf{S} \cup (\mathbf{I} \setminus \mathbf{O}))$, the sum of all incoming currents (i.e., currents from neighboring nets to *self*) is null. $\mathcal{R}_{\text{Kirchhoff}}$ defines such constraint for all internal nets and outputs. For pure inputs (i.e., elements of $\mathbf{I} \setminus \mathbf{O}$), no current is involved, while supplies act as a current source in case of short-circuit.

$$\bigwedge_{self \in \mathbf{N} \setminus (\mathbf{S} \cup (\mathbf{I} \setminus \mathbf{O}))} \left(\sum_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} \mathcal{I}(n, X, self) = 0 \right) \qquad (\mathcal{R}_{\text{Kirchhoff}})$$

Every device constrains the current between every two neighboring nets with respect to the applied voltages. We define how currents and voltages are related for transistors, diodes, and resistors, respectively, in Sections 4.6.2.2 to 4.6.2.4.

4.6.2.2 Transistor Modeling

In the literature, SPICE device models[1] refer to mathematical models used to simulate a circuit's behavior. There exist several device models, classified based on the level of physical detail they

1 The term 'device models' in the terminology of SPICE is not to be confused with the term 'model' in the terminology of SMT solving. The latter means a valuation of the variables of a formula which make the formula satisfiable.

capture [Synopsys, 2013]. We consider the Level 1 model, also called Schichman-Hodges model, as it is simpler (while being sufficiently detailed for EOS errors detection) and allows to express the behavior of a physical MOSFET model in terms of current and voltage relations [Patel, 2014]. Modeling in this case depends on various physical device parameters: V_t (the threshold voltage), $\frac{W}{L}$ (channel's width-to-length ratio), K_p (the transconductance parameter), and λ (charge-carrier effective mobility parameter).

Current and voltage are defined with respect to three regions of operation: *cut-off*, *linear*, and *saturation* [Ytterdal et al., 2003]. Table 4.1 summarizes NMOS regions of operation for the Level 1 model, as well as associated source-to-drain current definitions, with respect to the gate-source applied voltage on a device X—as usually considered in the literature. Definitions of PMOS regions of operation are complementary to the NMOS type.

Table 4.1: NMOS regions of operation for a device $X \in \mathbf{D}_{\texttt{NMOS}}$.

Region	Condition	$\mathcal{I}_{\text{SD}}(X)$
Cut-off	$\mathcal{V}_{\text{GS}}(X) \leq V_t$	0
Linear	$\mathcal{V}_{\text{GS}}(X) > V_t$ and $\mathcal{V}_{\text{DS}}(X) < \mathcal{V}_{\text{GS}}(X) - V_t$	$K_p \frac{W}{L} (1 + \lambda \mathcal{V}_{\text{DS}}(X)) \mathcal{V}_{\text{DS}}(X) \times \left(\mathcal{V}_{\text{GS}}(X) - V_t - \frac{\mathcal{V}_{\text{DS}}(X)}{2} \right)$
Saturation	$\mathcal{V}_{\text{GS}}(X) > V_t$ and $\mathcal{V}_{\text{DS}}(X) \geq \mathcal{V}_{\text{GS}}(X) - V_t$	$\frac{1}{2} \frac{W}{L} K_p (1 + \lambda \mathcal{V}_{\text{DS}}(X)) \times (\mathcal{V}_{\text{GS}}(X) - V_t)^2$

Note. As source and drain are interchangeable, substituting $\mathcal{V}_{\text{GS}}(X)$ with $\mathcal{V}_{\text{GD}}(X)$ in the conditions in Table 4.1 is also valid. Figure 4.28 depicts a 3D plot, for NMOS and PMOS, of the source-to-drain current as a relation of gate-source and gate-drain voltages, based on Level 1 SPICE model. The plot is obtained with $V_t = 0.2\,\text{V}$, $\frac{W}{L} = 1$, $K_p = 2 \times 10^{-5}\,\text{A}\,\text{V}^{-2}$, and $\lambda = 0.02\,\text{V}^{-1}$ [LTwiki, 2024].

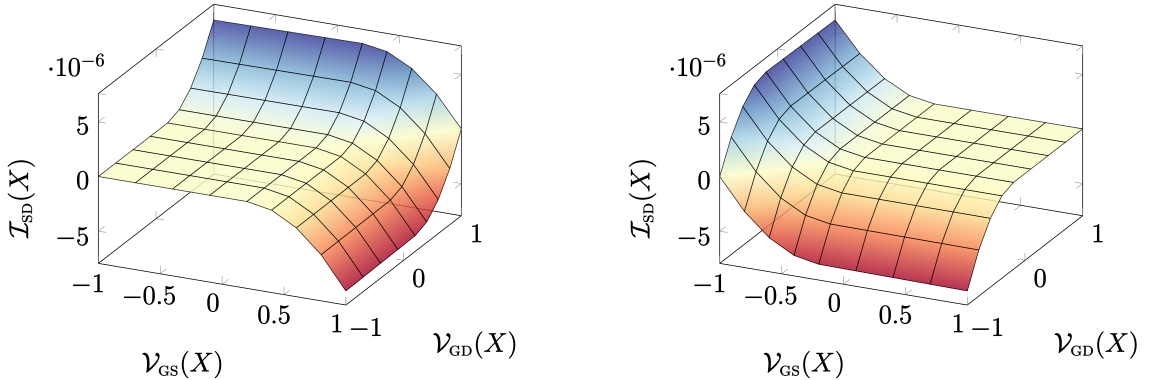


Figure 4.28: Level 1 SPICE model based I-V characteristics of a device X, for NMOS (left) and PMOS (right).

We propose linear approximations to Level 1 SPICE equations of non-linear current formulas (i.e., current in linear and saturation regions), in order to allow the encoding of function $\mathcal{I}$ in the LRA subset of SMT formula. These approximations are acceptable in our context because we don't target high precision analyses. The stakes are neither to be sound nor complete, but rather to be close to concrete semantics. A sound extension of this approach, using union of polyhedra to avoid inexact approximations is discussed in Section 4.8.2.

In practice, λ is small, so we accept the approximation $1 + \lambda \mathcal{V}_{\text{DS}}(X) \approx 1$. In the linear region, as $\mathcal{V}_{\text{DS}}(X) < \mathcal{V}_{\text{GS}}(X) - V_t$, a lower-bound of the source-to-drain current is achieved with $\mathcal{V}_{\text{DS}}(X) = \mathcal{V}_{\text{GS}}(X) - V_t$. Therefore, $\mathcal{I}_{\text{SD}}(X) > \frac{1}{2} \frac{W}{L} K_p \mathcal{V}^2_{\text{DS}}(X)$. For small voltage values (i.e., less than 1 V) of voltage drops across transistors, we have $\mathcal{V}^2_{\text{DS}}(X) < \mathcal{V}_{\text{DS}}(X)$. Hence, we approximate current in the linear region as $\mathcal{I}_{\text{SD}}(X) \sim C \times \mathcal{V}_{\text{DS}}(X)$—where $C \triangleq \frac{1}{2} \frac{W}{L} K_p$. Similarly, we approximate $(\mathcal{V}_{\text{GS}}(X) - V_t)^2$,

in saturation, with $\mathcal{V}_{\text{GS}}(X) - V_t$. Hence, in the saturation region, $\mathcal{I}_{\text{SD}}(X) \sim C \times (\mathcal{V}_{\text{GS}}(X) - V_t)$. The same reasoning applies to PMOS devices, with complementary conditions to the NMOS. Such approximations are depicted in Figure 4.29—where we consider, by convention, the source-to-drain current sign to be determined by the voltage difference $\mathcal{V}_{\text{SD}}(X)$, i.e., $\mathcal{I}_{\text{SD}}(X)$ is positive when $\mathcal{V}_{\text{S}}(X) > \mathcal{V}_{\text{D}}(X)$.

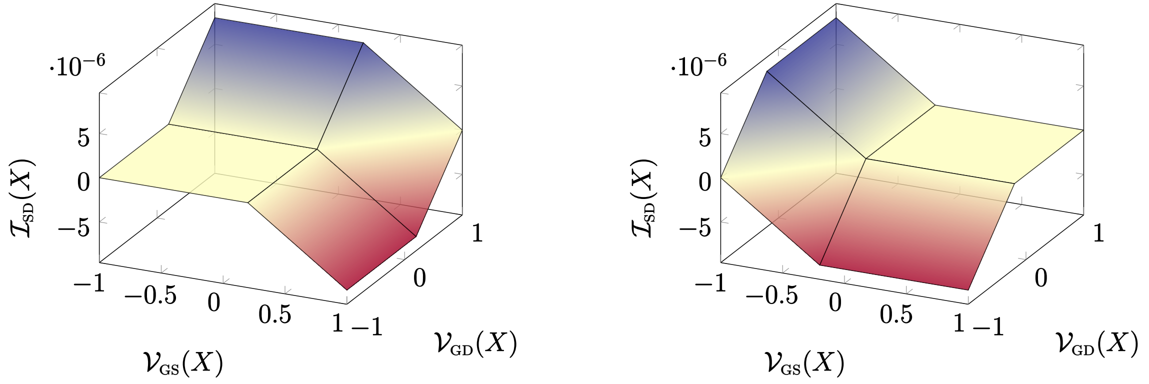


Figure 4.29: Linear approximation of I-V characteristics of a device X, for NMOS (left) and PMOS (right).

To formalize these approximations, instead of the activation predicate ($\mathcal{O}n$) used in switch-based semantics (Section 4.5.2), we consider predicates $\mathcal{C}ut\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$, $\mathcal{L}in\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$, and $\mathcal{S}at\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$, which tell, respectively, whether a transistor is in the cut-off, linear, or saturation region of operation and constrain $\mathcal{I}$ accordingly.

We approximate MOSFET regions of operation with the union of 4 quarter Euclidean planes, denoted $\mathcal{C}ut_{\text{P}}$, $\mathcal{L}in_{\text{P}}$, $\mathcal{S}at^1_{\text{P}}$ and $\mathcal{S}at^2_{\text{P}}$. Figure 4.30 shows the regions of operation of NMOS device types as a relation of gate, source, and drain voltages, where a threshold voltage (V_t) is considered. Figure 4.31 shows an analogous abstraction for PMOS device types.

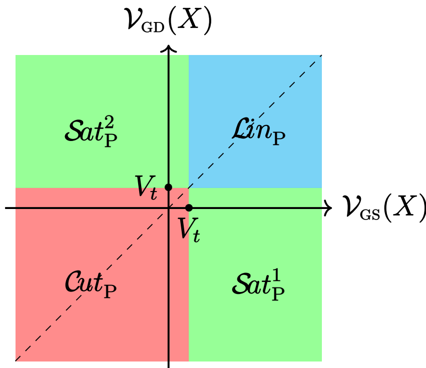


Figure 4.30: Abstraction of NMOS regions of operation for a transistor X.

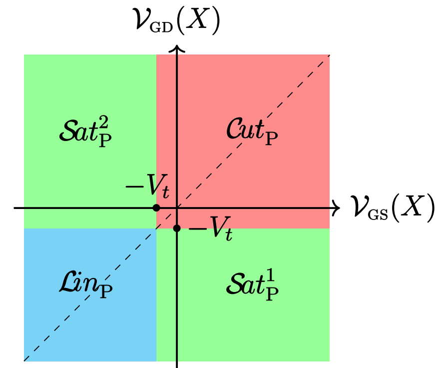


Figure 4.31: Abstraction of PMOS regions of operation for a transistor X.

Based on MOSFET characteristics equations, and on the abstractions in Figures 4.30 and 4.31, a transistor X is in the cut-off mode if neither $\mathcal{V}_{\text{GS}}(X)$ nor $\mathcal{V}_{\text{GD}}(X)$ exceed the threshold voltage (V_t), if $X \in \mathbf{D}_{\texttt{NMOS}}$. In such case, $\mathcal{I}_{\text{SD}}(X) = 0$. Current is also null when $\mathcal{V}_{\text{GS}}(X) = \mathcal{V}_{\text{GD}}(X)$, which corresponds to the diagonal (dashed line). Definition $\mathcal{C}ut^{\texttt{NMOS}}_{\text{def}}$ defines the $\mathcal{C}ut$ predicate for NMOS devices. The definition both indicates the (pre-)conditions under which a transistor in cut-off, as well as the impact of being in cut-off (by forcing the current to be null, which is somehow a post-condition).

$$\begin{aligned} \mathcal{C}ut\colon \mathbf{D}_{\texttt{NMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) \leq V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) \leq V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = 0 \end{pmatrix} \end{aligned} \qquad (\mathcal{C}ut^{\texttt{NMOS}}_{\text{def}})$$

$\mathcal{L}in^{\texttt{NMOS}}_{\text{def}}$ defines the predicate $\mathcal{L}in(X)$, for an NMOS device X. The constraints involve a deterministic computation of the source-to-drain current ($\mathcal{I}_{\text{SD}}(X)$).

$$\begin{aligned} \mathcal{L}in\colon \mathbf{D}_{\texttt{NMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) > V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) > V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = -C \times \mathcal{V}_{\text{DS}}(X) \end{pmatrix} \end{aligned} \qquad (\mathcal{L}in^{\texttt{NMOS}}_{\text{def}})$$

We observe from $\mathcal{L}in^{\texttt{NMOS}}_{\text{def}}$ that when $\mathcal{V}_{\text{GS}}(X) = \mathcal{V}_{\text{GD}}(X)$, the current $\mathcal{I}_{\text{SD}}(X)$ is null, which is coherent with the diagonal in MOSFET characteristics curves in the linear mode.

The saturation region corresponds to the union of the two domains $\mathcal{S}at^1_{\text{P}}$ and $\mathcal{S}at^2_{\text{P}}$ (Figure 4.30). $\mathcal{S}at^{\texttt{NMOS}}_{\text{def}}$ defines such constraints, including how the source-to-drain current is computed in each case, for NMOS devices.

$$\begin{aligned} \mathcal{S}at\colon \mathbf{D}_{\texttt{NMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) > V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) \leq V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = -C \times (\mathcal{V}_{\text{GS}}(X) - V_t) \end{pmatrix} \\ \vee & \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) \leq V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) > V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = C \times (\mathcal{V}_{\text{GD}}(X) - V_t) \end{pmatrix} \end{pmatrix} \end{aligned} \qquad (\mathcal{S}at^{\texttt{NMOS}}_{\text{def}})$$

Analogously, we define the predicates for PMOS devices as shown in $\mathcal{C}ut^{\texttt{PMOS}}_{\text{def}}$, $\mathcal{L}in^{\texttt{PMOS}}_{\text{def}}$, and $\mathcal{S}at^{\texttt{PMOS}}_{\text{def}}$.

$$\begin{aligned} \mathcal{C}ut\colon \mathbf{D}_{\texttt{PMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) \geq -V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) \geq -V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = 0 \end{pmatrix} \end{aligned} \qquad (\mathcal{C}ut^{\texttt{PMOS}}_{\text{def}})$$

$$\begin{aligned} \mathcal{L}in\colon \mathbf{D}_{\texttt{PMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) < -V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) < -V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = -C \times \mathcal{V}_{\text{DS}}(X) \end{pmatrix} \end{aligned} \qquad (\mathcal{L}in^{\texttt{PMOS}}_{\text{def}})$$

$$\begin{aligned} \mathcal{S}at\colon \mathbf{D}_{\texttt{PMOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} & \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) < -V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) \geq -V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = -C \times (\mathcal{V}_{\text{GS}}(X) + V_t) \end{pmatrix} \\ \vee & \begin{pmatrix} & \mathcal{V}_{\text{GS}}(X) \geq -V_t \\ \wedge & \mathcal{V}_{\text{GD}}(X) < -V_t \\ \wedge & \mathcal{I}_{\text{SD}}(X) = C \times (\mathcal{V}_{\text{GD}}(X) + V_t) \end{pmatrix} \end{pmatrix} \end{aligned} \qquad (\mathcal{S}at^{\texttt{PMOS}}_{\text{def}})$$

Because $\mathcal{L}in$, $\mathcal{S}at$, and $\mathcal{C}ut$ are mutually exclusive, it is sufficient to assert, with logical disjunction, that either one of the three predicates is $\top$ ($\mathcal{R}^{\texttt{MOS}}_{\text{regions}}$).

$$\bigwedge_{X \in \mathbf{D}_{\texttt{MOS}}} \mathcal{C}ut(X) \vee \mathcal{L}in(X) \vee \mathcal{S}at(X) \qquad (\mathcal{R}^{\texttt{MOS}}_{\text{regions}})$$

4.6.2.3 Diode Modeling

Shockley diode equation models the I-V relation of diodes [Shockley, 1949], as shown in E_{Shockley}, where I_D is the diode current, I_S is the scale current, V_D is the voltage across the diode, V_T is the thermal voltage, and n is the material constant. In numerical applications, we use $I_S = 1\,\text{A}$, $n = 1$, and $V_T = 25.852\,\text{mV}$ (value at 300 K) [Cataldo et al., 2016].

$$I_D = I_S\left(e^{\frac{V_D}{nV_T}} - 1\right) \tag{E_{Shockley}}$$

We approximate the current in Shockley diode equations with an affine function (so that we remain in the LRA fragment of SMT) when the anode-cathode voltage exceeds V_t (Figure 4.32), based on the maximum anode-cathode voltage applied on a diode, denoted $V_{\max}$. The value $V_{\max}$ can be obtained from a static voltage propagation process (which was already used to define static voltages in switch-based semantics). It is associated to a maximum current, denoted $I_{\max}$ ($E_{I_{\max}}$). When the anode-cathode voltage is less than V_t, the current is null. Note that, in our linear approximation, exact values of $V_{\max}$ and $I_{\max}$ are numerically computed in floating point and then approximated with rational numbers.

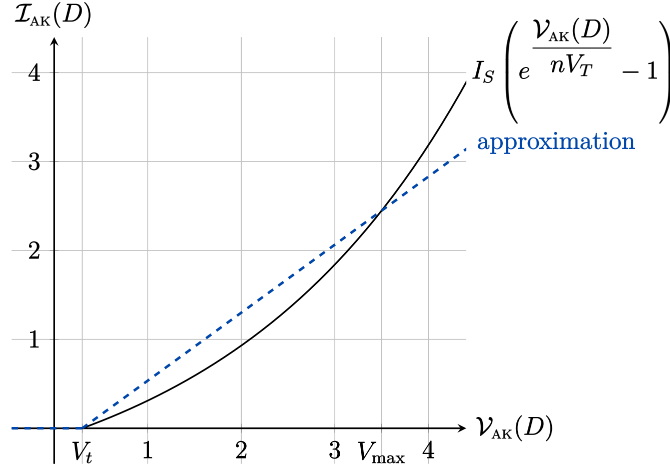


Figure 4.32: Linear approximation of the current of a diode $D \in \mathbf{D}_{\texttt{DIO}}$.

$$I_{\max} = I_S\left(e^{\frac{V_{\max}}{nV_T}} - 1\right) \tag{$E_{I_{\max}}$}$$

We define predicates $\mathcal{C}ut$ and $\mathcal{F}wd$ for diodes, respectively in $\mathcal{C}ut^{\texttt{DIO}}_{\text{def}}$ and $\mathcal{F}wd^{\texttt{DIO}}_{\text{def}}$, which constrain the anode-to-cathode current based on the voltage applied. A diode is either cut-off or in the *forward* region ($\mathcal{R}^{\texttt{DIO}}_{\text{regions}}$).

$$\begin{aligned} \mathcal{C}ut \colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} \mathcal{V}_{\text{AK}}(X) \le V_t \\ \wedge\, \mathcal{I}_{\text{AK}}(X) = 0 \end{pmatrix} \end{aligned} \tag{$\mathcal{C}ut^{\texttt{DIO}}_{\text{def}}$}$$

$$\begin{aligned} \mathcal{F}wd \colon \mathbf{D}_{\texttt{DIO}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \begin{pmatrix} \mathcal{V}_{\text{AK}}(X) > V_t \\ \wedge\, \mathcal{I}_{\text{AK}}(X) = I_{\max} \cdot \dfrac{\mathcal{V}_{\text{AK}}(X) - V_t}{V_{\max} - V_t} \end{pmatrix} \end{aligned} \tag{$\mathcal{F}wd^{\texttt{DIO}}_{\text{def}}$}$$

$$\bigwedge_{X \in \mathbf{D}_{\texttt{DIO}}} \mathcal{C}ut(X) \vee \mathcal{F}wd(X) \tag{$\mathcal{R}^{\texttt{DIO}}_{\text{regions}}$}$$

4.6.2.4 Resistor Modeling

Each resistor has a resistance value, given by Function $\mathrm{R}\colon \mathbf{D}_{\mathtt{RES}} \longrightarrow \mathbb{Q}_{\geq 0}$. Resistors' current is defined by Ohm's law. $\mathcal{R}^{\mathtt{RES}}_{\text{current}}$ constrains resistors' currents with such definition.

$$\bigwedge_{X \in \mathbf{D}_{\mathtt{RES}}} \mathcal{I}_{\text{AB}}(X) = \frac{\mathcal{V}_{\text{AB}}(X)}{\mathrm{R}(X)} \qquad (\mathcal{R}^{\mathtt{RES}}_{\text{current}})$$

4.6.2.5 Circuit Formula

When combined (via logical conjunctions), previously defined semantics rules define the circuit formula encoding its behavior.

Regions of operation. Constraints on regions of operation and current definitions are summarized, for all devices, in $\mathcal{R}_{\text{regions}}$.

$$\mathcal{R}^{\mathtt{MOS}}_{\text{regions}} \wedge \mathcal{R}^{\mathtt{DIO}}_{\text{regions}} \wedge \mathcal{R}^{\mathtt{RES}}_{\text{current}} \qquad (\mathcal{R}_{\text{regions}})$$

Current sign. In practice, semantics define either $\mathcal{I}(n_1, X, n_2)$ or $\mathcal{I}(n_2, X, n_1)$, while the symmetry is enforced with $\mathcal{R}_{\substack{\text{current}\\\text{sign}}}$.

$$\bigwedge_{self \in \mathbf{N}} \ \bigwedge_{\substack{(self, X, n) \in \\ \textsc{Neighbors}(self)}} \mathcal{I}(self, X, n) = -\mathcal{I}(n, X, self) \qquad (\mathcal{R}_{\substack{\text{current}\\\text{sign}}})$$

From the modeling abstractions introduced, we construct our quantitative circuit semantics as the conjunction of the different semantics rules. This new encoding is denoted $\mathcal{S}^q$, defined in ${\mathcal{S}^q}_{\text{def}}$. We recall (from Section 4.5.2) that both $\mathcal{R}_{\mathbf{S}}$ and $\mathcal{R}_{\mathbf{I}}$ rules can be used to respectively constrain the voltage values of the circuit's supplies (i.e., nets in $\mathbf{S}$) and inputs (i.e., nets in $\mathbf{I}$). Those rules can be simple variations of the ones previously defined (Section 4.2.2)—where supplies are constrained to pre-defined values (i.e., possible power configurations), and where inputs can take any value between ground and the highest supply value in the circuit.

$$\mathcal{S}^q \triangleq \mathcal{R}_{\text{regions}} \wedge \mathcal{R}_{\text{Kirchhoff}} \wedge \mathcal{R}_{\substack{\text{current}\\\text{sign}}} \wedge \mathcal{R}_{\substack{\text{static}\\\text{domain}}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad ({\mathcal{S}^q}_{\text{def}})$$

4.7 Formal Comparison of Circuit Semantics

Given a circuit semantics $\mathcal{S}$, a valuation is a function which associates a value to each of the variables used in $\mathcal{S}$. In our case, a valuation consists in a $\mathcal{V}$ function associating to each net a voltage value, plus, for quantitative semantics $\mathcal{S}^q$, an $\mathcal{I}$ function associating a current intensity to each triple (n_1, X, n_2), where n_1 and n_2 are neighboring nets to device X. Other variables and predicates (e.g., $\mathcal{C}ut$, $\mathcal{L}in$, $\mathcal{S}at$, $\mathcal{F}wd$, $\mathcal{O}n$) are directly computed from the valuation of nets' voltages and devices' currents. They can be replaced by their definition in the formula without changing the semantics, and are therefore not considered as actual variables of the formula. A *model*[2] of $\mathcal{S}$ is a valuation for which $\mathcal{S}$ evaluates to $\top$. We write $\mathcal{V} \models \mathcal{S}^t$ (resp. $(\mathcal{V}, \mathcal{I}) \models \mathcal{S}^q$) when $\mathcal{V}$ (resp. $(\mathcal{V}, \mathcal{I})$) is a model of $\mathcal{S}^t$ (resp. $\mathcal{S}^q$).

We compare all three semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$, in terms of their models in Figure 4.33. For a given circuit, (1) possible models in $\mathcal{S}^s$ are not necessarily possible in $\mathcal{S}^t$, and vice versa, (2) possible models in $\mathcal{S}^s$ are not necessarily possible in $\mathcal{S}^q$, and vice versa, and (3) all models of $\mathcal{S}^q$ are also models of $\mathcal{S}^t$.

2 The term 'model' in the terminology of SMT solving is not to be confused with the 'device models' as used in previous sections.

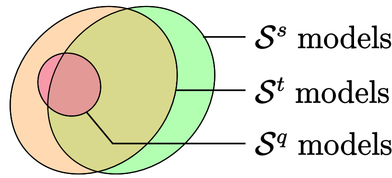


Figure 4.33: Graphical representation of the relation between circuit models in the switch-based semantics ($\mathcal{S}^s$), switch-based semantics with threshold ($\mathcal{S}^t$), and quantitative semantics ($\mathcal{S}^q$).

First, we illustrate all three semantics adequacy with SPICE (Section 4.7.1). Second, we prove that $\mathcal{S}^t \not\subseteq \mathcal{S}^s$ and $\mathcal{S}^s \not\subseteq \mathcal{S}^t$ (Section 4.7.2). Third, we prove that $\mathcal{S}^s \not\subseteq \mathcal{S}^q$ and $\mathcal{S}^q \not\subseteq \mathcal{S}^s$ (Section 4.7.3). Fourth, we prove that $\mathcal{S}^q \subseteq \mathcal{S}^t$, and moreover, there exists circuits for which $\mathcal{S}^q \neq \mathcal{S}^t$ (Section 4.7.4).

Table 4.2 summarizes the comparisons between circuit semantics, and outlines the corresponding theorems—which we prove next.

Relation	Meaning	Corresponding theorem
$\mathcal{S}^s \not\subseteq \mathcal{S}^t$	Not all models of $\mathcal{S}^s$ are models of $\mathcal{S}^t$	Theorem 4.7.1
$\mathcal{S}^t \not\subseteq \mathcal{S}^s$	Not all models of $\mathcal{S}^t$ are models of $\mathcal{S}^s$	Theorem 4.7.2
$\mathcal{S}^s \not\subseteq \mathcal{S}^q$	Not all models of $\mathcal{S}^s$ are models of $\mathcal{S}^q$	Theorem 4.7.3
$\mathcal{S}^q \not\subseteq \mathcal{S}^s$	Not all models of $\mathcal{S}^q$ are models of $\mathcal{S}^s$	Theorem 4.7.4
$\mathcal{S}^t \not\subseteq \mathcal{S}^q$	Not all models of $\mathcal{S}^t$ are models of $\mathcal{S}^q$	Theorem 4.7.5
$\mathcal{S}^q \subseteq \mathcal{S}^t$	All models of $\mathcal{S}^q$ are necessarily models of $\mathcal{S}^t$	Theorem 4.7.6

Table 4.2: Summary of formal semantics comparison.

4.7.1 Illustration of Semantics Adequacy with SPICE

We consider the Inverter circuit in Figure 4.34. We compare the output voltage $\mathcal{V}(\texttt{Z})$, in $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$, by varying the input voltage $\mathcal{V}(\texttt{A})$ between $\mathcal{V}(\texttt{GND})$ and $\mathcal{V}(\texttt{VDD})$. We set $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$ and $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$.

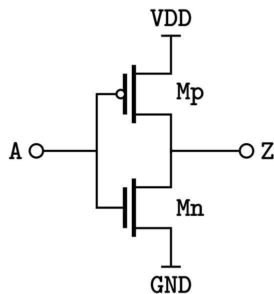


Figure 4.34: (Reminder) Inverter circuit schematic.

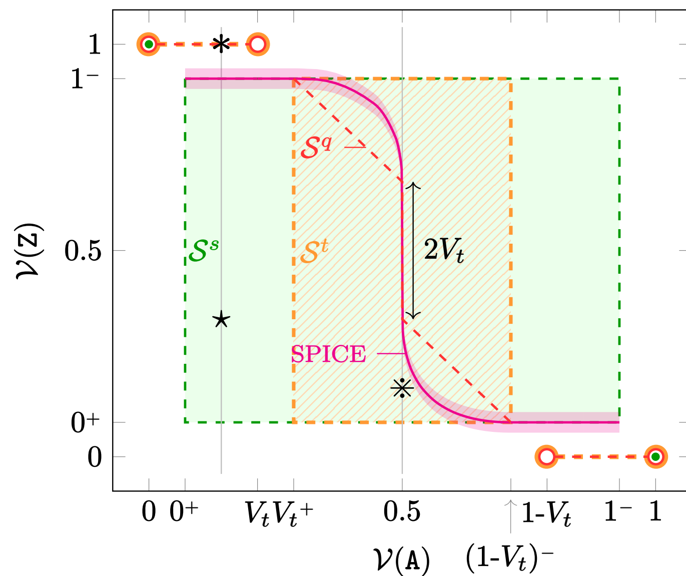


Figure 4.35: Input-output relation for the Inverter circuit (Figure 4.34), from semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$, and from simulation (SPICE).

A value $\mathcal{V}(\mathtt{Z})$ which corresponds to some $\mathcal{V}(\mathtt{A}) = a_{\text{ref}}$, in $\mathcal{S}^s$, is obtained from a model that satisfies $\mathcal{S}^s \wedge (\mathcal{V}(\mathtt{A}) = a_{\text{ref}})$. For semantics $\mathcal{S}^t$ and $\mathcal{S}^q$, as well as for simulation, we set $V_t = 0.2\,\text{V}$. We compare obtained models with simulation (SPICE), as shown in Figure 4.35 (the different marks correspond to counterexamples, which we detail below). We refer to simulation as it constitutes a good basis for assessing the precision of different abstractions.

In $\mathcal{S}^s$, when $0 < \mathcal{V}(\mathtt{A}) < 1$, the output voltage $\mathcal{V}(\mathtt{Z})$ is non-deterministic as all values in the interval $]0,1[$ are possible for $\mathcal{V}(\mathtt{Z})$. There are two cases where the voltage of Z is deterministic: $\mathcal{V}(\mathtt{A}) = 0 \Leftrightarrow \mathcal{V}(\mathtt{Z}) = 1$ and $\mathcal{V}(\mathtt{A}) = 1 \Leftrightarrow \mathcal{V}(\mathtt{Z}) = 0$. In $\mathcal{S}^t$, we gain more in terms of accuracy, as the threshold voltage is taken into account. We have: $0 \leq \mathcal{V}(\mathtt{A}) \leq V_t \Leftrightarrow \mathcal{V}(\mathtt{Z}) = 1$ and $1 - V_t \leq \mathcal{V}(\mathtt{A}) \leq 1 \Leftrightarrow \mathcal{V}(\mathtt{Z}) = 0$, while remaining non-deterministic for $\mathcal{V}(\mathtt{A}) \in]V_t, 1 - V_t[$. On the other hand, $\mathcal{S}^q$ is indeed a linear approximation of the curve obtained from SPICE. We use the x^-/x^+ notation to graphically represent the bounds of open intervals, e.g., the orange rectangle covers $]V_t, 1 - V_t[\times]0,1[$.

4.7.2 Comparison of $\mathcal{S}^s$ with $\mathcal{S}^t$

We show that a circuit model $\mathcal{V}$ of $\mathcal{S}^s$ does not necessarily satisfy $\mathcal{S}^t$, and vice versa—formally shown in Theorems 4.7.1 and 4.7.2, which we prove with counterexamples from the Inverter circuit.

Theorem 4.7.1. $\exists \mathcal{V},\ \mathcal{V} \models (\mathcal{S}^s \wedge \neg \mathcal{S}^t)$

Theorem 4.7.2. $\exists \mathcal{V},\ \mathcal{V} \models (\mathcal{S}^t \wedge \neg \mathcal{S}^s)$

Proof (Theorems 4.7.1 and 4.7.2). *Consider the Inverter in Figure 4.34, with* $0 = \mathcal{V}(\mathit{GND}) < V_t < \mathcal{V}(\mathit{VDD})$. *We have in* $\mathcal{S}^s$*:*

$$\begin{aligned} 0 < \mathcal{V}(A) < V_t &\Rightarrow \mathcal{O}n(M1) \wedge \mathcal{O}n(M2) && \text{(from } \mathcal{O}n^{\mathtt{PMOS}}_{\text{def}} \text{ and } \mathcal{O}n^{\mathtt{NMOS}}_{\text{def}}\text{)} \\ &\Rightarrow \mathcal{V}(\mathit{GND}) < \mathcal{V}(Z) < \mathcal{V}(\mathit{VDD}) && \text{(from } \mathcal{R}_{\substack{local\\voltage}}\text{)} \end{aligned}$$

and we have in $\mathcal{S}^t$*:*

$$\begin{aligned} 0 < \mathcal{V}(A) < V_t &\Rightarrow \mathcal{O}n^t(M1) \wedge \neg \mathcal{O}n^t(M2) && \text{(from } \mathcal{O}n^{t\,\mathtt{PMOS}}_{\text{def}} \text{ and } \mathcal{O}n^{t\,\mathtt{NMOS}}_{\text{def}}\text{)} \\ &\Rightarrow \mathcal{V}(Z) = \mathcal{V}(\mathit{VDD}) && \text{(from } \mathcal{R}^t_{\substack{local\\voltage}}\text{)} \end{aligned}$$

If we set $\mathcal{V}(Z) \in]\mathcal{V}(\mathit{GND}), \mathcal{V}(\mathit{VDD})[$, *we have* $\mathcal{V} \models \mathcal{S}^s$ *but* $\mathcal{V} \not\models \mathcal{S}^t$ *(see marker* $\star$ *in Figure 4.35). If we set* $\mathcal{V}(Z) = \mathcal{V}(\mathit{VDD})$, *we have* $\mathcal{V} \models \mathcal{S}^t$ *but* $\mathcal{V} \not\models \mathcal{S}^s$ *(see marker* $*$ *in Figure 4.35).* □

4.7.3 Comparison of $\mathcal{S}^s$ with $\mathcal{S}^q$

We show that a circuit model $\mathcal{V}$ of $\mathcal{S}^s$ does not necessarily satisfy $\mathcal{S}^q$, and vice versa. Formally, this is expressed with the following two theorems:

Theorem 4.7.3. $\exists \mathcal{V},\ \mathcal{V} \models (\mathcal{S}^s \wedge \neg \mathcal{S}^q)$

Theorem 4.7.4. $\exists \mathcal{V},\ \mathcal{V} \models (\mathcal{S}^q \wedge \neg \mathcal{S}^s)$

Proof (Theorems 4.7.3 and 4.7.4). *The proof is similar to the one for Theorems 4.7.1 and 4.7.2. The counter-example used also holds for Theorems 4.7.3 and 4.7.4.* □

4.7.4 Comparison of $\mathcal{S}^q$ with $\mathcal{S}^t$

We first show that a valuation $\mathcal{V}$ that satisfies $\mathcal{S}^t$ does not necessarily satisfy $\mathcal{S}^q$ (Theorem 4.7.5). Second, we prove that, for every valuation $(\mathcal{V}, \mathcal{I})$ that satisfies $\mathcal{S}^q$, $\mathcal{V}$ also satisfies $\mathcal{S}^t$ (Theorem 4.7.6).

Theorem 4.7.5. $\exists \mathcal{V},\ \forall \mathcal{I},\ (\mathcal{V}, \mathcal{I}) \models (\mathcal{S}^t \wedge \neg \mathcal{S}^q)$

Proof (Theorem 4.7.5). *Consider the Inverter in Figure 4.34, with* $0 = \mathcal{V}(\texttt{GND}) < V_t < \frac{1}{2}\mathcal{V}(\texttt{VDD})$ *and* $\mathcal{V}(\texttt{A}) = \frac{1}{2}\mathcal{V}(\texttt{VDD})$*. In* $\mathcal{S}^t$*, the only constraint on* $\mathcal{V}(\texttt{Z})$ *comes from* $\mathcal{O}n^t(\texttt{M1}) \wedge \mathcal{O}n^t(\texttt{M2})$*, which gives* $\mathcal{V}(\texttt{GND}) < \mathcal{V}(\texttt{Z}) < \mathcal{V}(\texttt{VDD})$*. All values in the interval* $]\mathcal{V}(\texttt{GND}), \mathcal{V}(\texttt{VDD})[$ *are valid for* $\mathcal{V}(\texttt{Z})$ *in* $\mathcal{S}^t$*. In* $\mathcal{S}^q$*, we have:*

$$\begin{aligned}\mathcal{V}(\texttt{A}) = \frac{1}{2}\mathcal{V}(\texttt{VDD}) &\Rightarrow \mathcal{S}at(\texttt{M1}) \wedge \mathcal{S}at(\texttt{M2}) && \textit{(from } Sat^{\texttt{PMOS}}_{\text{def}} \textit{ and } Sat^{\texttt{NMOS}}_{\text{def}}\textit{)}\\ &\Rightarrow (\mathcal{V}_{GS}(\texttt{M1}) < -V_t) \wedge (\mathcal{V}_{GD}(\texttt{M1}) \geq -V_t)\\ &\quad \wedge (\mathcal{V}_{GS}(\texttt{M2}) > V_t) \wedge (\mathcal{V}_{GD}(\texttt{M2}) \leq V_t)\\ &\Rightarrow \mathcal{V}(\texttt{A}) - V_t \leq \mathcal{V}(\texttt{Z}) \leq \mathcal{V}(\texttt{A}) + V_t\end{aligned}$$

If we set $\mathcal{V}(\texttt{Z}) = \dfrac{\mathcal{V}(\texttt{A}) - V_t}{2}$*, we have* $\mathcal{V} \models \mathcal{S}^t$ *but* $\forall \mathcal{I}, (\mathcal{V}, \mathcal{I}) \not\models \mathcal{S}^q$ *(see marker ⁜ in Figure 4.35).* □

Theorem 4.7.6. $\forall \mathcal{V},\ \forall \mathcal{I},\ (\mathcal{V}, \mathcal{I}) \models \mathcal{S}^q \Rightarrow \mathcal{V} \models \mathcal{S}^t$

As semantics rules $\mathcal{R}_{\substack{\text{static}\\\text{domain}}}$, $\mathcal{R}_{\mathbf{S}}$, and $\mathcal{R}_{\mathbf{I}}$ are common among $\mathcal{S}^t$ and $\mathcal{S}^q$ (i.e., they are syntactically and semantically the same), we only need to show that $\mathcal{V} \models \mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}}$, using rules $\mathcal{R}_{\text{regions}}$, $\mathcal{R}_{\text{Kirchhoff}}$, and $\mathcal{R}_{\substack{\text{current}\\\text{sign}}}$ from $\mathcal{S}^q$.

First, we rewrite the local voltage rule $\mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}}$, to express it, for every net $\textit{self} \in \mathbf{N}$, as a disjunction of two terms *currentFlows*(*self*) and *noCurrent*(*self*). This is shown in the following lemma:

Lemma 4.7.1. $\mathcal{R}^t_{\substack{\textit{local}\\\textit{voltage}}} = \bigwedge_{\textit{self} \in \mathbf{N}} (\textit{currentFlows}(\textit{self}) \vee \textit{noCurrent}(\textit{self}))$*, where*

$$\textit{currentFlows}(\textit{self}) \triangleq \left(\begin{array}{ll} \bigvee_{\substack{(\textit{self}, X, n)\\ \in \textsc{Neighbors}(\textit{self})}} & \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(\textit{self}) < \mathcal{V}(n)\big) \\ \wedge \bigvee_{\substack{(\textit{self}, X, n)\\ \in \textsc{Neighbors}(\textit{self})}} & \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(n) < \mathcal{V}(\textit{self})\big) \end{array}\right)$$

and

$$\textit{noCurrent}(\textit{self}) \triangleq \bigwedge_{\substack{(\textit{self}, X, n)\\ \in \textsc{Neighbors}(n)}} \neg\mathcal{O}n^t(X) \vee \mathcal{V}(n) = \mathcal{V}(\textit{self})$$

Intuitively, the original formulation states that current enters a net through a device if and only if current also exits this net through another device. The formulation of Lemma 4.7.1 equivalently states that either no current flows through a net, or current enters through a device and exits through another.

Proof (Lemma 4.7.1). *This is a straightforward application of* $(a \Leftrightarrow b) = (a \wedge b) \vee (\neg a \wedge \neg b) = \textit{currentFlows}(\textit{self}) \vee \textit{noCurrent}(\textit{self})$*. The term noCurrent(self) is obtained by the application of De Morgan's laws, then by trivially transforming* $(x \leq y) \wedge (x \geq y)$ *into* $x = y$*.* □

Second, we introduce, for different current ($\mathcal{I}$) values in $\mathcal{S}^q$, implications in $\mathcal{S}^t$. Lemma 4.7.2 says that the absence of current between two neighboring nets in $\mathcal{S}^q$ implies that either the device connecting the two nets is off or that there is no voltage drop between the two nets.

Lemma 4.7.2. $\forall \textit{self} \in \mathbf{N},\ \forall (n, X, \textit{self}) \in \textsc{Neighbors}(\textit{self})$,

$$\mathcal{I}(n, X, \textit{self}) = 0 \Rightarrow \neg\mathcal{O}n^t(X) \vee \big(\mathcal{V}(\textit{self}) = \mathcal{V}(n)\big)$$

In case the current between two neighboring nets is non-null, the connecting device is necessarily on, and the current sign is given by the order of their voltages (Lemmas 4.7.3 and 4.7.4).

Lemma 4.7.3. $\forall self \in \mathbf{N},\ \forall (n, X, self) \in \textsc{Neighbors}(self)$,

$$\mathcal{I}(n, X, self) > 0 \Rightarrow \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(self) < \mathcal{V}(n)\big)$$

Lemma 4.7.4. $\forall self \in \mathbf{N},\ \forall (n, X, self) \in \textsc{Neighbors}(self)$,

$$\mathcal{I}(n, X, self) < 0 \Rightarrow \mathcal{O}n^t(X) \wedge \big(\mathcal{V}(self) > \mathcal{V}(n)\big)$$

For now, we accept all Lemmas 4.7.2 to 4.7.4. Their proofs are presented, for transistors, diodes, and resistors, respectively, in Sections 4.7.4.1 to 4.7.4.3. We immediately proceed with the proof for Theorem 4.7.6.

Proof (Theorem 4.7.6). *Let $(\mathcal{V}, \mathcal{I})$ be a model of $\mathcal{S}^q$, i.e., $(\mathcal{V}, \mathcal{I}) \models \mathcal{S}^q$. We show that $\mathcal{V}$ is a model of $\mathcal{S}^t$. We show that $\mathcal{V}$ verifies ${\mathcal{R}^t}_{\substack{local\\voltage}}$, which is equivalent (by Lemma 4.7.1) to showing that for each net $self \in \mathbf{N}$, either $\mathcal{V}$ satisfies $currentFlows(self)$ or $\mathcal{V}$ satisfies $noCurrent(self)$.*

Let $self \in \mathbf{N}$. Since $(\mathcal{V}, \mathcal{I}) \models \mathcal{S}^q$, from $\mathcal{R}_{Kirchhoff}$, we have:

$$\sum_{\substack{(n, X, self)\\ \in \textsc{Neighbors}(self)}} \mathcal{I}(n, X, self) = 0$$

Either all terms are null (A),

$$\left(\bigwedge_{\substack{(n, X, self)\\ \in \textsc{Neighbors}(self)}} \mathcal{I}(n, X, self) = 0 \right) \tag{A}$$

or there exists at least a positive term and a negative one (B).

$$\left(\begin{array}{l} \displaystyle\bigvee_{\substack{(n, X, self)\\ \in \textsc{Neighbors}(self)}} \mathcal{I}(n, X, self) > 0 \\ \wedge \displaystyle\bigvee_{\substack{(n, X, self)\\ \in \textsc{Neighbors}(self)}} \mathcal{I}(n, X, self) < 0 \end{array} \right) \tag{B}$$

In case (A), by Lemma 4.7.2, $noCurrent(self)$ is satisfied, and in case (B), by Lemmas 4.7.3 and 4.7.4, $currentFlows(self)$ is satisfied. Finally, from both cases (A) and (B), we showed that $\mathcal{R}_{Kirchhoff}$ implies ${\mathcal{R}^t}_{\substack{local\\voltage}}$. Thus $\mathcal{V} \models \mathcal{S}^t$. □

The proof is valid for any device for which Lemmas 4.7.2 to 4.7.4 hold.

4.7.4.1 Lemmas Proofs for Transistors

Proof (Lemma 4.7.2 for transistors). *From the definitions of predicates $\mathcal{C}ut$, $\mathcal{L}in$, and $\mathcal{S}at$ (Section 4.6.2.2)—which constrain the current intensity, we have:*

$$\mathcal{I}(n, X, self) = 0 \Leftrightarrow \mathcal{C}ut(X) \vee \big(\mathcal{V}(n) = \mathcal{V}(self)\big)$$

Note that in the case of transistors, nets n and self are, in any order, the source and drain nets. From the definitions of $\mathcal{C}ut$ *(*$\mathcal{C}ut^{\mathtt{NMOS}}_{\mathrm{def}}$*) and* $\mathcal{O}n^t$ *(*$\mathcal{O}n^{t\,\mathtt{NMOS}}_{\mathrm{def}}$*), we have, for NMOS devices:*

$$\mathcal{C}ut(X) \Rightarrow \big(\mathcal{V}_{GS}(X) \le V_t\big) \wedge \big(\mathcal{V}_{GD}(X) \le V_t\big) \Leftrightarrow \neg\mathcal{O}n^t(X)$$

and for PMOS devices (from $\mathcal{C}ut^{\mathtt{PMOS}}_{\mathrm{def}}$ *and* $\mathcal{O}n^{t\,\mathtt{PMOS}}_{\mathrm{def}}$*):*

$$\mathcal{C}ut(X) \Rightarrow \big(\mathcal{V}_{GS}(X) \ge -V_t\big) \wedge \big(\mathcal{V}_{GD}(X) \ge -V_t\big) \Leftrightarrow \neg\mathcal{O}n^t(X)$$

Hence: $\mathcal{I}(n, X, \mathit{self}) = 0 \Rightarrow \neg\mathcal{O}n^t(X) \vee \big(\mathcal{V}(n) = \mathcal{V}(\mathit{self})\big)$. □

Proof (Lemma 4.7.3 for transistors). *Let* $X \in \mathbf{D}_{\mathtt{MOS}}$*. Let* $n = \textsc{Src}(X)$*, and* $\mathit{self} = \textsc{Drn}(X)$*. We have* $\mathcal{I}(n, X, \mathit{self}) = \mathcal{I}_{SD}(X) > 0$ *under specific conditions in the linear and saturation modes. Note that the union of the linear and saturation modes corresponds to* $\mathcal{O}n^t(X)$ *in* $\mathcal{S}^t$*, so* $\mathcal{I}(n, X, \mathit{self}) = \mathcal{I}_{SD}(X) > 0$ *implies* $\mathcal{O}n^t(X)$*. We still need to show* $\mathcal{V}(\mathit{self}) < \mathcal{V}(n)$*. In the linear mode (*$\mathcal{L}in^{\mathtt{NMOS}}_{\mathrm{def}}$ *and* $\mathcal{L}in^{\mathtt{PMOS}}_{\mathrm{def}}$*), we have:*

$$\begin{aligned}\mathcal{I}_{SD}(X) > 0 \ &\Rightarrow -C \times \mathcal{V}_{DS}(X) > 0\\ &\Rightarrow \mathcal{V}_D(X) < \mathcal{V}_S(X)\\ &\Rightarrow \mathcal{V}(\mathit{self}) < \mathcal{V}(n)\end{aligned}$$

In saturation ($\mathcal{S}at^{\mathtt{NMOS}}_{\mathrm{def}}$ *and* $\mathcal{S}at^{\mathtt{PMOS}}_{\mathrm{def}}$*),* $\mathcal{I}_{SD}(X) > 0$ *corresponds to* $\mathcal{S}at^2_{\mathrm{P}}$ *(Figures 4.30 and 4.31). For NMOS, that is when* $(\mathcal{V}_{GS}(X) \le V_t) \wedge (\mathcal{V}_{GD}(X) > V_t)$*. Notice that this corresponds to* $\mathcal{V}_{GS}(X) - \mathcal{V}_{GD}(X) < 0$*. Hence, in saturation:*

$$\begin{aligned}\mathcal{I}_{SD}(X) > 0 \ &\Rightarrow \mathcal{V}_D(X) < \mathcal{V}_S(X)\\ &\Rightarrow \mathcal{V}(\mathit{self}) < \mathcal{V}(n)\end{aligned}$$

In all cases, $\mathcal{V}(\mathit{self}) < \mathcal{V}(n)$*, which concludes the proof for NMOS devices. The same result is obtained for PMOS devices, under the condition* $(\mathcal{V}_{GS}(X) < -V_t) \wedge (\mathcal{V}_{GD}(X) \ge -V_t)$*.* □

Proof (Lemma 4.7.4 for transistors). *The proof is symmetrical to the one of Lemma 4.7.3.* □

4.7.4.2 Lemmas Proofs for Diodes

Proof (Lemma 4.7.2 for diodes).

$$\begin{aligned}\mathcal{I}(n, X, \mathit{self}) = 0 &\Leftrightarrow \mathcal{C}ut(X) && \text{(from } \mathcal{C}ut^{\mathtt{DIO}}_{\mathrm{def}} \text{ and } \mathcal{F}wd^{\mathtt{DIO}}_{\mathrm{def}}\text{)}\\ &\Rightarrow \mathcal{V}_{AK}(X) \le V_t && \text{(definition of } \mathcal{C}ut\text{)}\\ &\Leftrightarrow \neg\mathcal{O}n^t(X) && \text{(definition of } \mathcal{O}n^t\text{)}\\ &\Rightarrow \neg\mathcal{O}n^t(X) \vee \big(\mathcal{V}(n) = \mathcal{V}(\mathit{self})\big)\end{aligned}$$

□

Proof (Lemma 4.7.3 for diodes). *Let* $X \in \mathbf{D}_{\mathtt{DIO}}$*. Let* $n = \textsc{Anode}(X)$*, and* $\mathit{self} = \textsc{Cathode}(X)$*.*

$$\begin{aligned}\mathcal{I}(n, X, \mathit{self}) > 0 &\Rightarrow \mathcal{F}wd(X) \wedge \big(\mathcal{V}_{AK}(X) - V_t > 0\big) && \text{(from } \mathcal{F}wd^{\mathtt{DIO}}_{\mathrm{def}}\text{)}\\ &\Leftrightarrow \mathcal{F}wd(X) \wedge \big(\mathcal{V}(n) - \mathcal{V}(\mathit{self}) > V_t > 0\big)\\ &\Rightarrow \mathcal{F}wd(X) \wedge \big(\mathcal{V}(\mathit{self}) < \mathcal{V}(n)\big)\\ &\Rightarrow \mathcal{O}n^t(X) \wedge (\mathcal{V}(\mathit{self}) < \mathcal{V}(n)) && \text{(as } \mathcal{F}wd(X) \Leftrightarrow \neg\mathcal{C}ut(X) \Leftrightarrow \mathcal{O}n^t(X)\text{)}\end{aligned}$$

□

Proof (Lemma 4.7.4 for diodes). *The proof is symmetrical to the one of Lemma 4.7.3.* □

4.7.4.3 Lemmas Proofs for Resistors

Proof (Lemma 4.7.2 for resistors).

$$\begin{aligned}\mathcal{I}(n, X, self) = 0 &\Rightarrow \mathcal{V}(n) = \mathcal{V}(self) && \textit{(from } \mathcal{R}^{\text{RES}}_{current}\textit{)}\\ &\Rightarrow \neg \mathcal{O}n^t(X) \lor (\mathcal{V}(n) = \mathcal{V}(self))\end{aligned}$$

□

Proof (Lemma 4.7.3 for resistors).

$$\begin{aligned}\mathcal{I}(n, X, self) > 0 &\Rightarrow \mathcal{V}(self) < \mathcal{V}(n) && \textit{(from } \mathcal{R}^{\text{RES}}_{current}\textit{)}\\ &\Rightarrow \mathcal{O}n^t(X) \land \mathcal{V}(self) < \mathcal{V}(n) && \textit{(from } {\mathcal{O}n^t}^{\text{RES}}_{\text{def}}\textit{)}\end{aligned}$$

□

Proof (Lemma 4.7.4 for resistors). *The proof is symmetrical to the one of Lemma 4.7.3.* □

4.8 Extensions

This section presents alternate choices, based on alternate hypotheses, in terms of circuit modeling. These extensions can be viewed as enhancements to existing modelings, or as a way to provide insights to implementers to adapt circuit semantics to their needs.

4.8.1 Alternate Definition of Inputs

The initial definition of inputs ($\mathbf{I}_{\text{def}}$, page 52) only considers circuit pins that are transistor gates as inputs. That is, however, a simplistic definition. A more realistic one should also consider what circuit designers call, in practice, inputs. In fact, inputs can drive transistor gates *via* some other devices, like transistors and diodes. An example is depicted in Figure 4.36, where input `A` of the Inverter reaches transistor gates via resistor `R1`.

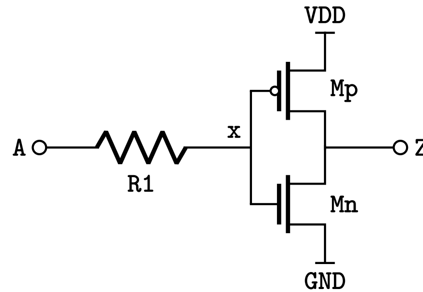


Figure 4.36: Inverter circuit with a resistor at the input interface.

An alternate definition we propose hereafter takes into account the fact that inputs are used as transistor gates, or as pins which statically reach some transistor gate. We define the set of inputs as the set of pins excluding supplies (i.e., $\mathbf{P} \setminus \mathbf{S}$) whose reachable nets include transistor gate nets, either by being a gate net itself or reaching some gate(s) through some other devices. We consider Function ReachGates: $(\mathbf{P} \setminus \mathbf{S}) \longrightarrow \mathscr{P}(\mathbf{G})$ which gives for each net the set of reachable gates (among the set of gates $\mathbf{G}$), defined in $\text{ReachGates}_{\text{def}}$ using helper Function $\text{ReachGates}^{\natural}{}_{\text{def}}$.

$$\textsc{ReachGates}^{\natural} \colon (\mathbf{P} \setminus \mathbf{S}) \times \mathscr{P}(\mathbf{N}) \longrightarrow \mathscr{P}(\mathbf{G})$$
$$(n, V) \longmapsto \left(\left(\bigcup_{\substack{(n, X, p) \in \textsc{Neighbors}(n) \\ p \notin V}} \textsc{ReachGates}^{\natural}(p, V \cup \{n\}) \right) \bigcup \left(\begin{cases} \{n\} & \text{if } n \in \bigcup_{X \in \mathbf{D}_{\texttt{MOS}}} \{\textsc{Gate}(X)\} \\ \{\} & \text{if } n \notin \bigcup_{X \in \mathbf{D}_{\texttt{MOS}}} \{\textsc{Gate}(X)\} \end{cases} \right) \right) \quad (\textsc{ReachGates}^{\natural}{}_{\text{def}})$$

$$\textsc{ReachGates} \colon (\mathbf{P} \setminus \mathbf{S}) \longrightarrow \mathscr{P}(\mathbf{G})$$
$$n \longmapsto \textsc{ReachGates}^{\natural}(n, \emptyset) \qquad (\textsc{ReachGates}_{\text{def}})$$

The set of inputs ($\mathbf{I}$) would thus be defined with the alternate definition $\mathbf{I}_{\substack{\text{alt} \\ \text{def}}}$.

$$\mathbf{I} \triangleq \bigcup_{\substack{n \in \mathbf{P} \setminus \mathbf{S}, \\ \textsc{ReachGates}(n) \neq \emptyset}} \{n\} \qquad (\mathbf{I}_{\substack{\text{alt} \\ \text{def}}})$$

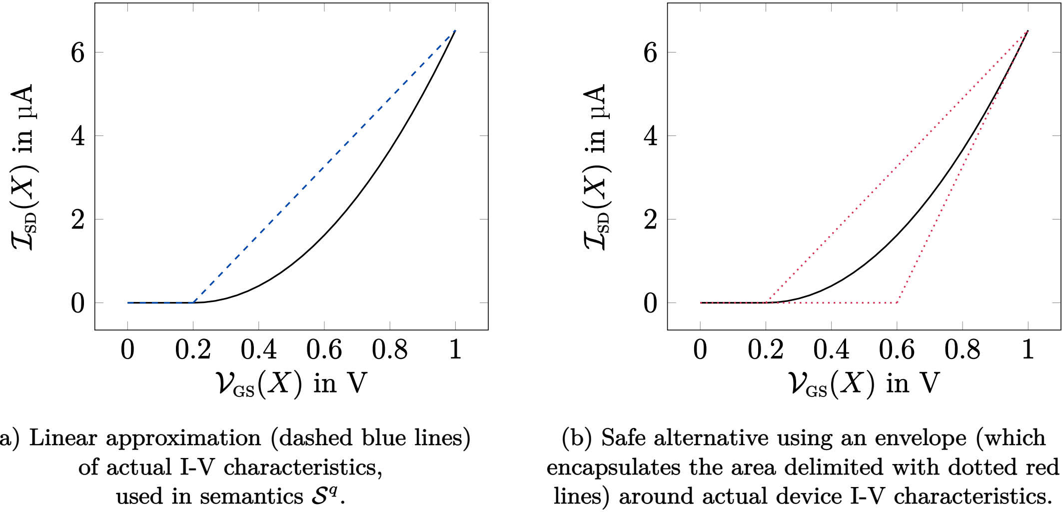
4.8.2 Towards Sound Quantitative Modeling

A promising extension is to define a sound modeling approach as an alternative to the linear approximations of device models used in quantitative semantics ($\mathcal{S}^q$). This can be achieved with approximating device I-V relations with an envelope of the actual curve. For transistors, apart from cut-off, each region of operation can be delimited with a polyhedron. Figure 4.37 shows how such conservative approximation (b) differs from the one used in the encoding of $\mathcal{S}^q$ (a). It is a 2-dimensional cut of the complete 3-dimensional figure. There are many ways such envelope can be constructed. Intuitively, we would expect that with a large number of constraints (i.e., hyper-planes making the envelope of the actual curve), solver performance will decrease.

4.9 Conclusion

The comparisons of circuit semantics conducted in this chapter are primarily based on the precision criterion. We have shown how our first attempted circuit semantics $\mathcal{S}^o$ is limited in terms of circuit topology coverage, making it hardly useful. The modeling obtained with $\mathcal{S}^s$ remains the most abstract, yet it is not guaranteed to be sound. $\mathcal{S}^t$ uses similar abstractions as used in $\mathcal{S}^s$, while taking into account the threshold voltage for the devices concerned. It is thus less abstract, and includes the behavior of SPICE. Quantitative semantics $\mathcal{S}^q$ uses a radically different modeling, which emphasizes precision. It is strictly included in $\mathcal{S}^t$ (Theorem 4.7.6), and is however not sound (our idea for a sound encoding was presented in the previous section, see Section 4.8.2). It approximates the behavior of SPICE, without exactly including all I-V relations that are legitimate from the point of view of SPICE.

All variants of circuit semantics presented in this chapter serve to translate circuit netlists into logic formulas, which can then be used in formal verification flows. Our particular use consists in formulating properties to verify and verify them using an SMT solver, which either proves the property holds on the circuit, or provides a counterexample showcasing an erroneous steady-state.



(a) Linear approximation (dashed blue lines) of actual I-V characteristics, used in semantics $\mathcal{S}^q$.

(b) Safe alternative using an envelope (which encapsulates the area delimited with dotted red lines) around actual device I-V characteristics.

Figure 4.37: Two modeling approaches of NMOS characteristics for a set value of $\mathcal{V}_{\text{GD}}(X)$, for $X \in \mathbf{D}_{\texttt{NMOS}}$ and $V_t = 0.2\,\text{V}$.

Intuitively, we could speculate that semantics $\mathcal{S}^s$ and $\mathcal{S}^t$ will perform better in terms of runtime and, therefore, in terms of scaling—since the SMT constraints they involve are simple. But given the coarse abstractions they use, they are more useful for analyzing non-quantitative properties. Dealing with quantitative properties is better done with $\mathcal{S}^q$, which is more precise, although it would take a solver more time to solve formulas encoded using $\mathcal{S}^q$. This is all mere guesswork, which can be verified through experimentation, so that we can better assess tradeoffs between performance and precision.

In fact, as future work, different semantics can be applied compositionally to different devices in the same circuit. For example, one can choose to apply the precise $\mathcal{S}^q$ semantics to specific devices on which some quantitative property is being verified, while applying the switch-based semantics $\mathcal{S}^s$ or $\mathcal{S}^t$ ($\mathcal{S}^s$ being a special case of $\mathcal{S}^t$ where the threshold voltage is null, but ideally different thresholds may be considered for different devices) on pure digital parts of the circuit or on parts where modeling the exact current and voltage quantities is less important.

This is made possible as our circuit semantics have a variable in common: they all reason about the same voltage valuation function $\mathcal{V}$. Device states' predicates can be projected from one semantics to the other. For example, Predicate $\mathscr{Cut}$ in $\mathcal{S}^q$ have the same definition of $\neg\mathscr{On}^t$ is $\mathcal{S}^t$. $\mathcal{S}^q$'s $\mathscr{Lin}$ and $\mathscr{Sat}$ region predicates correspond to $\mathcal{S}^t$'s $\mathscr{On}^t$.

It is, in theory, relatively simple to compose circuit semantics in this way. Yet, when it comes to implementation, some adaptations are to be made. Since $\mathcal{S}^q$ reasons, in addition, about current ($\mathcal{I}$), combining $\mathcal{S}^q$ with $\mathcal{S}^t$ (or $\mathcal{S}^s$) requires us to define a current function on every device. This is primarily needed for applying Kirchhoff's current law on the circuit's nets (see $\mathcal{R}_{\text{Kirchhoff}}$). Further constraints on the newly created $\mathcal{I}$ variables are to be added to ensure the integrity of the current sign.

With the nice property $\mathcal{S}^q \subseteq \mathcal{S}^t$ (Theorem 4.7.6), the constraints on the devices on which $\mathcal{S}^q$ is applied are also compatible with $\mathcal{S}^t$. For instance, in case of the absence of short-circuits in a given electrical path, $\mathcal{S}^q$ and $\mathcal{S}^t$ behave exactly the same. When a short-circuit is involved, $\mathcal{S}^t$ allows an interval of values of possible voltage drops across a device, while $\mathcal{S}^q$ is more restrictive in terms of the possible voltages allowed. The voltage drop determined by $\mathcal{S}^q$, however, remains within the interval of values allowed by $\mathcal{S}^t$. When a short-circuit occurs, devices on which $\mathcal{S}^q$ is applied simply restrict the devices on which $\mathcal{S}^t$ is applied further, resulting in the voltage drops in the whole electrical path being in short-circuit to be deterministically computed. Somehow the local precision obtained from applying $\mathcal{S}^q$ on one device results in a global precision along other devices on which other (less precise) semantics are applied.

In Chapter 5, we use the verification framework presented in Figure 4.1 (page 42) to address the problem of verifying specific electrical properties. Analyses are conducted on industrial use cases, on which we discuss analysis precision and scaling considerations.

In Chapter 6, we focus on an extended usage of circuit models to reason about circuit reliability (see Figure 4.2, page 43). We support the corresponding formalism with illustrations on simple examples, and we assess their deployment in analyzing a database of circuits issued from an industrial case study.

5

Leveraging Circuit Semantics for Error Detection

THE ULTIMATE GOAL of a circuit verification tool is to find the entirety of errors in a design, and only *true* errors. Part of that is to miss no true error, even if that means to yield some false alarms. In the terminology of formal methods, that is soundness. A tool which blindly answers "the whole design is erroneous" is, after all, sound. Yet, such a tool is far from being useful in practice. This is why it is also important for the verification tool to be as complete as possible, i.e., the tool finds true design errors and only those. The lower the number of false positives (i.e., false alarms) the tool yields, the most useful the error reports are to circuit designers. It is on the basis of this metric that the precision of the analysis is evaluated: one approach is more precise than another if the former results in less false alarms, while remaining sound. Analysis precision is mainly determined by the circuit modeling approach adopted, and can only be quantified on given benchmarks. In this regard, it is more valuable to study precision on realistic benchmarks rather than on synthetic ones. That is why the experimental evaluations conducted (in this chapter, and in Chapter 6) mostly come from industrial benchmarks. In our analysis framework, we use circuit semantics built in Chapter 4 as the foundation for identifying electrical errors. The choice of circuit semantics is based on performance versus precision tradeoffs, and on suitability with respect to the properties of interest. In this chapter, we present two use cases: (1) missing level-shifter (MLS) detection (see Section 3.2.3), and (2) electrical overstress (EOS) identification (see Section 3.2.2). We present their formalization using variables of circuit semantics, which we then exploit in verifying circuits' properties. The verification flow consists in using satisfiability modulo theories (SMT) solving, to draw conclusions about the correctness of a design with respect to some property. The SMT solver either proves the absence of errors, or shows a counterexample demonstrating the existence of bugs in the design, both outcomes being useful to verification engineers. Each of the two use cases considered is supported with empirical evaluation on large industrial circuits—on which we study relevance of the analysis results and discuss scaling matters. The chapter ends with a brief quantitative comparison against relevant works from the literature—initially introduced in Chapter 3.

5.1 Missing Level-Shifter

A level-shifter is a circuit that is used to connect circuit cells belonging to different power domains, in a way that prevents undesired current leakage on transistors at the interface of power domains. There exist several conventional level-shifter circuits. They are, in general, supplied with voltage values from the two power domains they connect, and are made of some switching circuitry to select one of the supply voltage values as the output, depending on the input value (see the example of a classical level-shifter and its functioning in page 27). Figure 5.1 recalls the use of a level-shifter in a multi-supply circuit example. The example implements a buffer circuit made of two Inverters,

having each a distinct power supply. Information propagates from the `VDDL` domain to the `VDDH` domain, where $\mathcal{V}(\texttt{VDDL}) < \mathcal{V}(\texttt{VDDH})$.

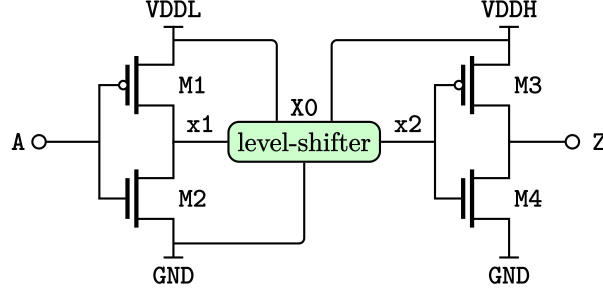


Figure 5.1: (Reminder) Use of a level-shifter to connect two Inverters to build a safe buffer circuit.

The level-shifter in this example is of kind 'Up', as it translates the electrical signal from a low power domain to a high power domain [Koo et al., 2005]. A level-shifter can also be used to connect power domains from high to low, in which case it is of kind 'Down' [Gundala et al., 2022].

Most classical approaches for identifying MLS consist in using voltage propagation to annotate circuit nets with the reachable supplies, and combine it with pattern recognition techniques to eliminate some false alarms based on previous knowledge of *safe* circuit cells.

5.1.1 Error Formalization

For a given circuit, we formalize a simple condition to detect a missing level-shifter with function $\mathcal{E}_{\texttt{MLS}} : \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$, defined in $\mathcal{E}_{\texttt{MLS}\,\mathrm{def}}$. It consists in checking whether a specific PMOS device belongs to two different power domains in a way that results in a non-null current leakage across the transistor. We limit the error formula to PMOS devices, because our case study, provided by our industrial partner, consists of a set of PMOS devices identified as potentially being at the interface between power domains. A dual clause can be expressed on NMOS device types.

$$
\begin{aligned}
&\mathcal{E}_{\texttt{MLS}} : \mathbf{D}_{\texttt{PMOS}} \longrightarrow \mathbb{B} \\
&M \longmapsto \left(\begin{array}{l}
\left(\begin{array}{l} \mathcal{V}_{\mathrm{G}}(M) < \mathcal{V}_{\mathrm{S}}(M) \\ \vee\ \mathcal{V}_{\mathrm{G}}(M) < \mathcal{V}_{\mathrm{D}}(M) \end{array} \right) \\
\wedge\ \mathcal{V}_{\mathrm{S}}(M) \neq \mathcal{V}_{\mathrm{D}}(M) \\
\wedge\ \mathcal{V}_{\mathrm{G}}(M) > \min\limits_{\substack{s\, \in\, \textsc{ReachSuppl}(\textsc{Src}(M)) \\ \cup \textsc{ReachSuppl}(\textsc{Drn}(M))}} \mathcal{V}(s)
\end{array} \right)
\end{aligned}
\tag{$\mathcal{E}_{\texttt{MLS}\,\mathrm{def}}$}
$$

We say that a PMOS transistor M is in a low-to-high power domain interface when its gate voltage is lower than its source (or drain) voltage, without being equal to the minimal reachable value from source and drain nets (which is usually the ground). Since we are dealing with PMOS device types, this is equivalent to the device being switched-on. There is a current leakage if the transistor is switched-on and $\mathcal{V}_{\mathrm{S}}(M) \neq \mathcal{V}_{\mathrm{D}}(M)$ as long as the gate voltage does not correspond to the minimal reachable value from source and drain. This is illustrated in Figure 5.2. Note that predicate $\mathcal{I}(s, M, d)$ is added for the sake of understanding, but it is not necessary—it is not part of the error predicate ($\mathcal{E}_{\texttt{MLS}\,\mathrm{def}}$). Having a non-null voltage difference between nets s and d, with the device M turned on, is sufficient.

If net g is connected to the ground, there is no missing level-shifter at the interface of the transistor, the ground being common to all power domains in our error model. This may lead to current leakage through the transistor, but it does not imply a missing level-shifter. More generally, there may be a current leakage through a device M that is not due to a missing level-shifter if the condition E_2 holds.

$$\mathcal{V}_{\mathrm{G}}(M) = \min_{\substack{s \,\in\, \textsc{ReachSuppl}(\textsc{Src}(M)) \\ \cup \textsc{ReachSuppl}(\textsc{Drn}(M))}} \mathcal{V}(s) \qquad (E_2)$$

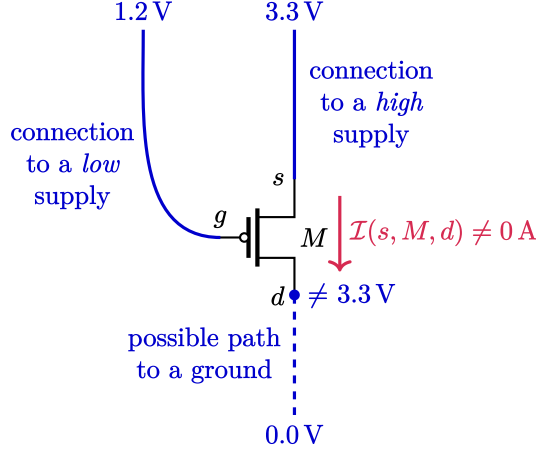


Figure 5.2: A PMOS device at the interface of two power domains, in a steady state corresponding to an MLS error.

Finally, we check the absence of an error at the interface of a specific PMOS device (M) in the circuit with semantics $\mathcal{S}$: if $\mathcal{S} \wedge \mathcal{E}_{\mathtt{MLS}}(M)$ is satisfiable, then we consider that M is indeed at the interface of two power domains where a level-shifter is missing.

Our industrial partner provided us a list of suspicious devices at the interface between power domains that are not protected with a level-shifter. We run the analysis for each suspicious device. In the absence of such information, we can check whether there exists at least one missing level-shifter in the circuit, by checking the satisfiability of

$$\mathcal{S} \wedge \left(\bigvee_{M \,\in\, \mathbf{D}_{\mathtt{PMOS}}} \mathcal{E}_{\mathtt{MLS}}(M) \right) .$$

Note that the error formula can be tweaked to fulfill specific needs, in a fully compositional way. This is possible thanks to the fact that the error formula is independent from the way circuit semantics are defined, as long as we have access to the voltage function $\mathcal{V} \colon \mathbf{N} \longrightarrow \mathbb{V}$. For example, the error formula can be adapted to search for missing level-shifters between specific power domains.

When it comes to the choice of the semantics to use for MLS identification, we evaluate all three semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$, via experimental benchmarks. This will give a clear idea on which one to pick for this specific use case, based on solving performance and relevance of the analysis results. We address this, in detail, next, in the industrial case study of Section 5.1.2.

5.1.2 Industrial Case Study

Our case-study consists of a set of real-life circuits and a set of PMOS devices potentially erroneous with respect to MLS. The circuits database is made of circuit netlists that are part of a 10-bit analog-to-digital converter (ADC). It represents a large variety of topologies that can be found in modern CMOS designs, while exhibiting a strong analog content. We analyze a total of 11 459 circuit instances (i.e., distinct supplies configurations) of 197 unique circuits, among which 20 distinct supplies (i.e., physical supply pins at the top-level circuit) are used.

To have an idea about the complexity of the circuits, we try to define a metric. Intuitively, we consider both the number of transistors and their connectivity. We represent connectivity with the connectivity ratio $\frac{|\mathbf{D}_{\mathtt{MOS}}|}{|\mathbf{N}|}$. As our circuits are only made of transistors, we have for these benchmarks

$\mathbf{D} = \mathbf{D}_{\mathtt{MOS}}$. This ratio only provides an idea about the composition of the benchmarks, but it is not necessarily correlated with the analysis complexity in terms of the solving time. Table 5.1 presents a summary of the analyzed circuits.

	Min	Median	Average	Max
Number of transistors $\lvert\mathbf{D}_{\mathtt{MOS}}\rvert$	2	14	46.2	1303
Connectivity ratio $\frac{\lvert\mathbf{D}_{\mathtt{MOS}}\rvert}{\lvert\mathbf{N}\rvert}$	0.5	1.27	1.46	5.5

Table 5.1: Statistics related to the case-study.

Each pair of circuit instance and transistor represents a suspect MLS configuration. There are 388 transistors in our list of potentially erroneous devices. Our industrial database therefore has a set of 22 598 cases that we proceed to check using our SMT based approach, with all three semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$.

Experiments are run on a machine with 32 GB system memory, using a single core clocked at 3.2 GHz. The analysis consists in building the SMT problem (i.e., logical conjunction of the circuit formula and the error formula, see error detection flow in page 42) and checking its satisfiability with the Z3 SMT solver [De Moura and Bjørner, 2008].

Table 5.2 compares analysis results. In the vast majority of cases, all semantics agree on either that there is an MLS error (line 1) or that there is no error (line 7). As we proved in Theorem 4.7.6 (page 74), it is not possible that $\mathcal{S}^q$ finds an error that is not found with $\mathcal{S}^t$ (line 3).

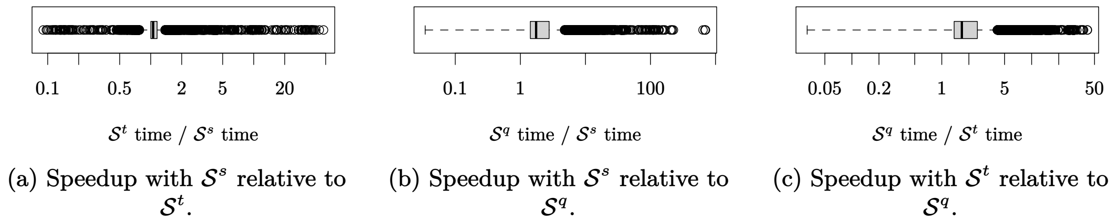
	$\mathcal{S}^s$	$\mathcal{S}^t$	$\mathcal{S}^q$	count	
1	E	E	E	7317	
2	E	E	N	2306	
3	E/N	N	E	0	⇝ *coherent with Theorem 4.7.6*
4	E	N	N	70	
5	N	E	E	572	
6	N	E	N	82	
7	N	N	N	12251	

Table 5.2: MLS analysis results. 'E' denotes an MLS Error, and 'N' denotes absence of MLS (No error).

Figure 5.4 shows a scatter plot of the analysis time, which, in total, takes 11 min 8 s with $\mathcal{S}^s$ (Figure 5.4a) 13 min 26 s with $\mathcal{S}^t$ (Figure 5.4b), and 33 min 47 s with $\mathcal{S}^q$ (Figure 5.4c).

Two-by-two comparisons of the solving time are plotted in Figure 5.5. Note that the different clusters observed represent circuit instances of the same netlists, the difference being the supplies configurations used.

Semantics' relative speedups are summarized in Figure 5.3. In average, $\mathcal{S}^s$ is 1.23 times faster than $\mathcal{S}^t$ (Figure 5.3a), with a median at 1.06. It is 2.64 times faster than $\mathcal{S}^q$ (Figure 5.3b), with a median at 1.67, while $\mathcal{S}^t$ is 3.56 times faster than $\mathcal{S}^q$ (Figure 5.3c), with a median at 1.75.

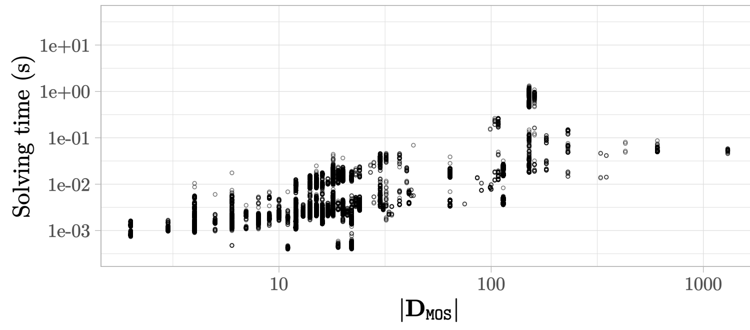


(a) Speedup with $\mathcal{S}^s$ relative to $\mathcal{S}^t$.

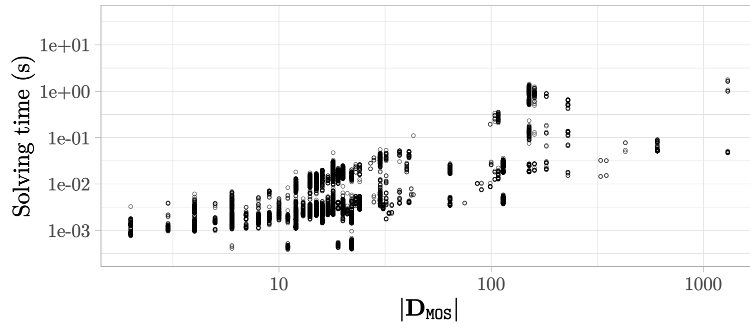
(b) Speedup with $\mathcal{S}^s$ relative to $\mathcal{S}^q$.

(c) Speedup with $\mathcal{S}^t$ relative to $\mathcal{S}^q$.

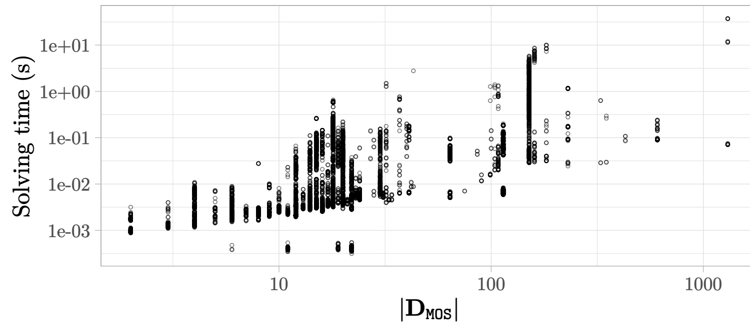
Figure 5.3: Comparing speedups among semantics $\mathcal{S}^q$ vs. $\mathcal{S}^t$, and $\mathcal{S}^s$ vs. $\mathcal{S}^t$.



(a) Semantics $\mathcal{S}^s$.

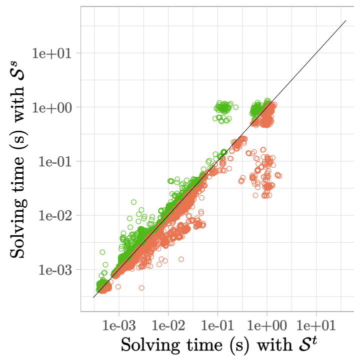


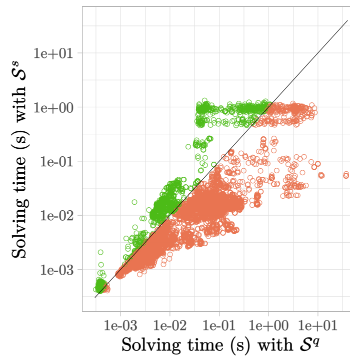
(b) Semantics $\mathcal{S}^t$.

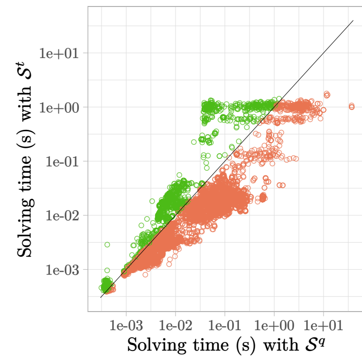


(c) Semantics $\mathcal{S}^q$.

Figure 5.4: Time spent in the solving phase for each analyzed case.







(a) Semantics $\mathcal{S}^t$ against $\mathcal{S}^s$. (b) Semantics $\mathcal{S}^q$ against $\mathcal{S}^s$. (c) Semantics $\mathcal{S}^q$ against $\mathcal{S}^t$.

Figure 5.5: Two-by-two comparisons of time spent in the solving phase for each analyzed case.

5.1.3 Performance Evaluation on Synthetic Benchmarks

Beyond the precision of the analysis, the choice of the circuit semantics also has a significant impact on the solver performance. The Z3 solver issues solving related statistics upon each run, with respect to some metrics. The metrics include solving time, memory consumption, variables created, decisions made, search conflicts, etc. We run experiments to compare some of these metrics for $\mathcal{S}^o$, $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$ on variable circuits' size. The goal of these experiments is only to study the scalability of our approach. For these synthetic benchmarks, we took advantage of the fact that are CMOS-compatible to also run them with $\mathcal{S}^o$ (which we temporarily discarded from experimental evaluation), the goal being to also have an idea on how it scales regardless of its limitations in terms of the *analyzability* of circuits with respect to their topology.

Benchmarks are run on generated full-adder circuits of variable size. The goal is to have comparable results on the same topology with different circuit sizes. The circuits generated are CMOS-based, which means they are analyzable with oriented signal semantics $\mathcal{S}^o$, as they pass the topology filter (Section 4.4.2). A 2-bit full-adder is composed of 9 NAND gates, and each NAND gate is made of 4 transistors (2 PMOS and 2 NMOS). The full-adder is therefore composed of 36 transistors. We generate N-bit adders as a composition of 2-bit full-adder blocks, adding two bit-vectors $a_{N-1} \dots a_1 a_0$ and $b_{N-1} \dots b_1 b_0$, with the input carry c_0 (see Figure 5.6). Each block is assigned a random voltage value ranging from 1 V to NV.

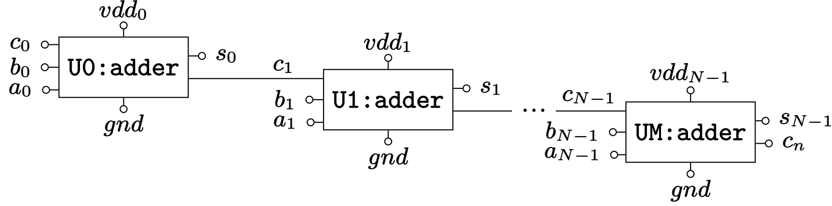


Figure 5.6: N-bit multi-supply full-adder.

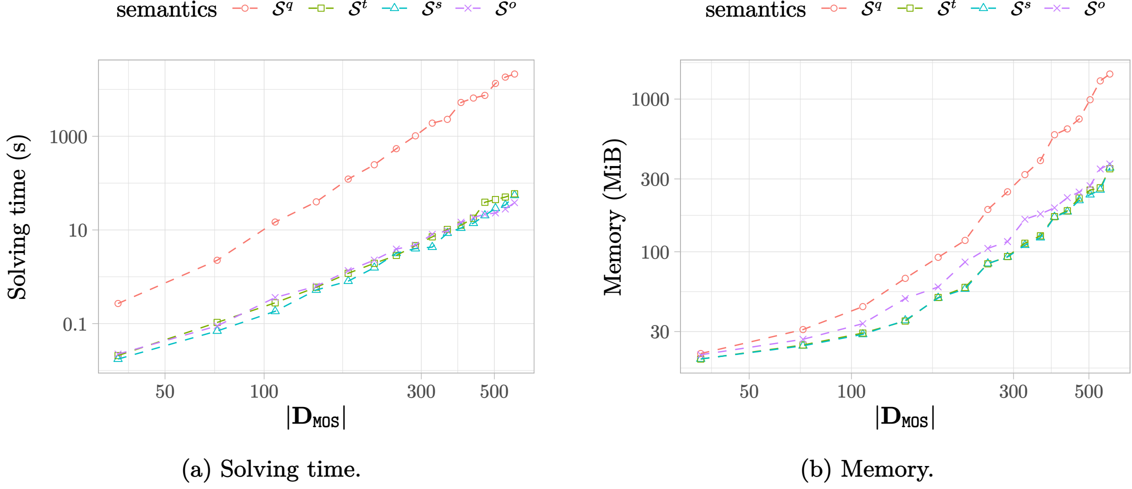
Benchmarks consist in running the analysis to check for missing level-shifters, using the same experimental setup as in Section 5.1.2, where the formula being checked is

$$\mathcal{S} \wedge \left(\bigvee_{M \in \mathbf{D}_{\texttt{PMOS}}} \mathcal{E}_{\texttt{MLS}}(M) \right), \text{with } \mathcal{S} \text{ being either } \mathcal{S}^o, \mathcal{S}^s, \mathcal{S}^t, \text{ or } \mathcal{S}^q.$$

Figure 5.7 presents our benchmarks results in terms of time (a) and memory (b). Semantics $\mathcal{S}^q$ turns out to be way slower than $\mathcal{S}^o$, $\mathcal{S}^s$, and $\mathcal{S}^t$. It is however not a general conclusion, since scalability largely depends on intrinsic properties about the circuit themselves (e.g., supplies, topology, and size).

Obtained results demonstrate that our approach using non-quantitative semantics scales up significantly better than quantitative ones ($\mathcal{S}^q$). With non-quantitative semantics, a 504-transistor circuit is analyzed in 23.06–44.56 s, depending on the semantics used. Memory usage is linear by intervals of different lengths. It remains globally larger with $\mathcal{S}^o$, and much more larger with $\mathcal{S}^q$.

It is however difficult to compare these performances against existing works, as our approach differs from the state of the art, and as the benchmarks used in related works are not publicly available. Yet, similar metrics were studied in [Afonso and Monteiro, 2017b] on a different electrical rule checking (ERC) problem, where a SAT solving based solution is proposed to check the possibility of short-circuits in single-supply designs. Their experimental results show that for a circuit made of 500 nets and 30 inputs, search for short-circuits could be done in 3329 s. In our synthetic benchmarks, checking for missing level-shifters in a multi-supply 16-bit full-adder circuit (made of 33 inputs, 345 nets, and 576 transistors) is done in only 55 s with $\mathcal{S}^s$. Besides, our approach clearly separates the circuit encoding and the property (or error) of interest, which makes it applicable to other ERC problems including (but not limited to) the analysis of short-circuits. The approach proposed by Afonso and Monteiro [2017b], in contrast, cannot be directly applied to other error models.



(a) Solving time. (b) Memory.

Figure 5.7: Full-adder benchmarks analysis results.

5.2 Electrical Overstress

In this section, we study a second type of errors: electrical overstress (EOS). EOS is a more quantitative phenomenon. Through it, we emphasize the interest of having quantitative semantics (namely, $\mathcal{S}^q$).

5.2.1 Property Formulation

A device $X \in \mathbf{D}_{\mathtt{MOS}}$ is in electrical overstress if there exists a circuit state where its applied voltage is beyond its specified voltage range. Such range is usually chosen so as to guarantee the lifespan and reliability of devices [Toshiba Electronic Devices & Storage Corporation, 2018]. Function $\textsc{allowedInterval}\colon \mathbf{D}_{\mathtt{MOS}} \longrightarrow \{[a,b],(a,b) \in \mathbb{V}^2\}$ gives the range of the voltage stress allowed on a device. It depends on technological device parameters, and is extracted from circuit netlists or the metadata accompanying the netlist. The specification of a device is defined, as a property (i.e., respecting the specification proves the $\mathcal{A}$bsence of EOS), in $\mathcal{A}_{\mathtt{EOS}}$def.

$$\begin{aligned} \mathcal{A}_{\mathtt{EOS}} \colon \mathbf{D}_{\mathtt{MOS}} &\longrightarrow \mathbb{B} \\ X &\longmapsto \left(\begin{array}{ll} & \mathcal{V}_{\text{GS}}(X) \in \textsc{allowedInterval}(X) \\ \wedge & \mathcal{V}_{\text{GD}}(X) \in \textsc{allowedInterval}(X) \end{array} \right) \end{aligned} \qquad (\mathcal{A}_{\mathtt{EOS}}\text{def})$$

To check whether there exists at least one violation of devices' specifications in terms of their voltage ratings, one can check whether the expression

$$\mathcal{S} \wedge \neg \left(\bigwedge_{X \in \mathbf{D}_{\mathtt{MOS}}} \mathcal{A}_{\mathtt{EOS}}(X) \right)$$

is satisfiable. If unsatisfiable, then all $\mathcal{A}_{\mathtt{EOS}}$ properties do hold on $\mathcal{S}$. Otherwise (if satisfiable), a counter-example circuit model where the property doesn't hold for at least one device is obtained. It is then possible to run an iterative process to yield all devices violating their respective specification, by re-checking the satisfiability of the formula above, while removing the yielded devices (from previous checks) from the big conjunction over $\mathbf{D}_{\mathtt{MOS}}$. In the worst case, the total number of solver runs is $|\mathbf{D}_{\mathtt{MOS}}|$ (i.e., the number of devices). This is achieved with Algorithm 1, where Function $\textsc{findSomeEOS}(\mathcal{S}, \text{suspects})$ returns a subset of devices in the set of 'suspects' that can be in EOS, for some circuit semantics $\mathcal{S}$. If the set is empty, then the circuit is guaranteed to have no EOS among 'suspects'. Function $\textsc{findAllEOS}(\mathcal{S}, \mathbf{D}_{\mathtt{MOS}})$ then repeatedly invokes $\textsc{findSomeEOS}$ until all devices in EOS are identified.

In fact, multiple EOS devices can be found simultaneously (i.e., with the same solver run) if they happen to be erroneous in the same circuit state. Hence, simply removing the yielded devices from future checks speeds-up the search compared to using some more basic heuristics (e.g., checking the property $\mathcal{A}_{\texttt{EOS}}$ by iterating on all devices in the circuit). Experiments show that an average of 5.5 EOS devices are found per solver run (Figure 5.8).

Algorithm 1 Search for devices in electrical overstress

```
1:  function FINDSOMEEOS(S, suspects)
2:      eos_devices ← ∅
3:      exists_eos ← S ∧ ( ⋁_{X ∈ suspects} ¬A_EOS(X) )
4:      status ← check exists_eos                          ▷ solver (Z3) call
5:
6:      if status = SAT then
7:          M ← get_model exists_eos
8:          DEVICEISERRONEOUS ← λX.(M ⊨ ¬A_EOS(X))
9:          eos_devices ← filter DEVICEISERRONEOUS suspects
10:     end if
11:
12:     return eos_devices
13: end function
14:
15: function FINDALLEOS(S, suspects)
16:     confirmed_eos ← ∅
17:
18:     do
19:         suspects ← suspects ∖ confirmed_eos
20:         new_eos ← FINDSOMEEOS(S, suspects)
21:         confirmed_eos ← confirmed_eos ∪ new_eos
22:     while new_eos ≠ ∅
23:
24:     return confirmed_eos
25: end function
26:
27: return FINDALLEOS(S, D_MOS)
```

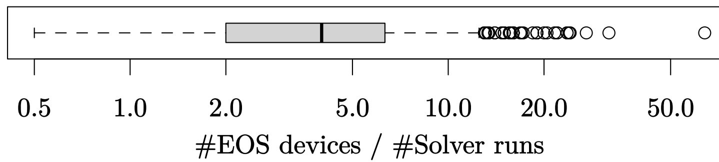


Figure 5.8: Average number of EOS devices per solver run.

We adapt the rules on supplies ($\mathcal{R}_{\mathbf{S}}$) and inputs ($\mathcal{R}_{\mathbf{I}}$) previously defined (page 52). For input voltages we either choose to restrict them (1) to the set of available supply values ($\mathcal{R}_{\mathbf{I}}$ strict) or (2) to the interval between the minimum and maximum available supply values ($\mathcal{R}_{\mathbf{I}}$ relaxed).

$\mathcal{R}_{\mathbf{I}}$ strict corresponds to typical hypotheses on digital input vectors, while $\mathcal{R}_{\mathbf{I}}$ relaxed is more adapted to analog inputs.

$$\bigwedge_{i \in \mathbf{I}} \left(\bigvee_{s \in \mathbf{S}} \mathcal{V}(i) = \mathcal{V}(s) \right) \tag{$\mathcal{R}_\mathbf{I}$ strict}$$

$$\bigwedge_{i \in \mathbf{I}} \left(\begin{array}{ll} & \mathcal{V}(i) \geq \min_{s \in \mathbf{S}} \mathcal{V}(s) \\ \wedge & \mathcal{V}(i) \leq \max_{s \in \mathbf{S}} \mathcal{V}(s) \end{array} \right) \tag{$\mathcal{R}_\mathbf{I}$ relaxed}$$

At a first glance, Algorithm 1 may seem inefficient due to running a 'fresh' solver at each of the EOS detection iterations. However, experiments show that the total solving time goes from 2 h 41 min without the incremental feature to 2 h 17 min with incremental solving (using `push/pop` functions from Z3). While solving performance is globally better with incremental solving, it is only faster on 43.47 % of the analyzed cases—all semantics combined. In 56.53 % of the cases, it turns out the basic use of Z3 leads to faster runs. Figure 5.9 compares Z3 solving time with and without using incremental solving, for both $\mathcal{R}_\mathbf{I}$ strict and $\mathcal{R}_\mathbf{I}$ relaxed. The line plotted corresponds to the diagonal (i.e., the solving time with both input restriction approaches is the same). Incremental solving does thus not always lead to faster analysis. The total gain remains marginal.

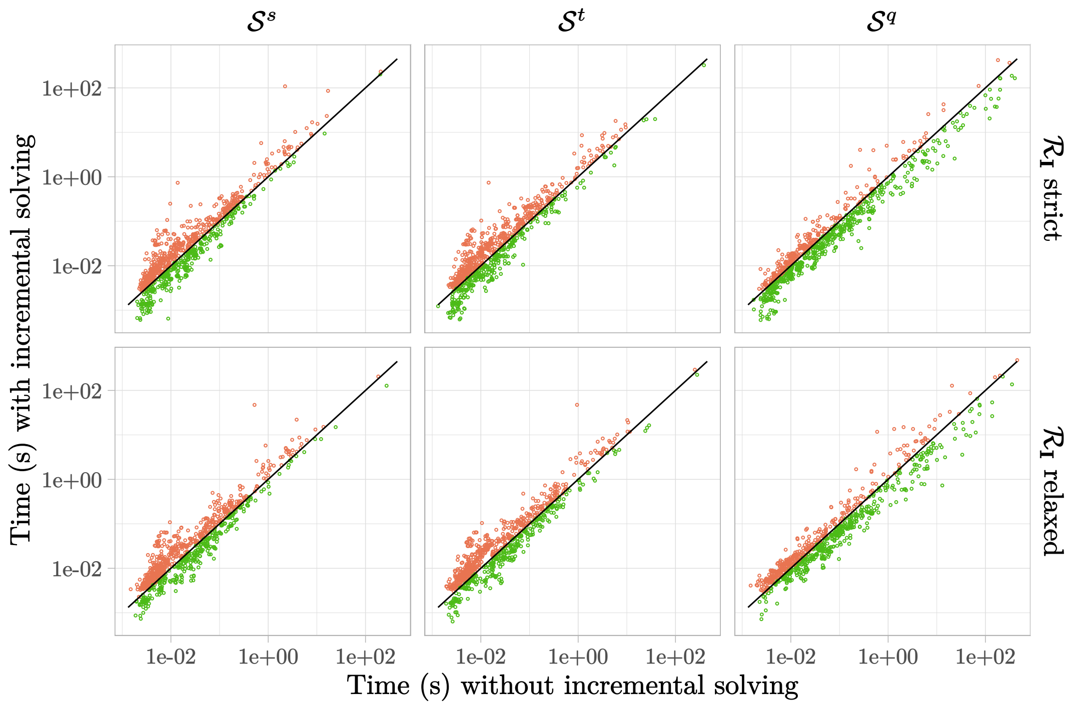


Figure 5.9: Total EOS detection time per circuit, with and without using incremental solving, for both choices of input constraints.

5.2.2 Experimental Evaluation

We conduct experiments on a case-study composed of 549 elementary sub-circuits—part of a radio frequency circuit design. Each sub-circuit is accompanied with its supplies configuration, inferred, each, from the top-level circuit supplies it can statically reach. Such information is obtained from a static voltage propagation algorithm. In total, the circuit uses 11 distinct power supply values ranging from 1 V to 3 V. The size of the sub-circuits ranges from 1 to 6230 transistors.

Table 6.3 presents an overview of the composition of circuits in the case-study, grouped by intervals based on their size. The circuits being made of transistors only, we have $\mathbf{D} = \mathbf{D}_{\texttt{MOS}}$.

$\mathbf{D}_{\texttt{MOS}}$	$[1, 10[$	$[10, 100[$	$[100, 1000[$	1000+
count	240	263	38	8

Table 5.3: Benchmark composition.

On each sub-circuit, we search for devices in electrical overstress, using an implementation of our error enumeration heuristics (Algorithm 1). The analysis is run with the three semantics: $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$. We compare the analysis results from each semantics against a static voltage propagation based analysis approach, denoted 'static', as a means to assess the usefulness of our semantics-based analyses in discarding false alarms. Such 'static' approach is safe (i.e., sound), as the approximations used are pessimistic (all devices are possibly on). In addition, we run SPICE simulations as a means to assess the precision of the approach.

It is indeed practically feasible to exhaustively simulate the circuits of our case study, given the relatively small number of instances to analyze: 549 circuits. Exhaustive simulation would have been intractable to run in the previous case study (MLS detection, Section 5.1.2), which have 22 598 instances—having each possibly multiple suspect devices to check.

With SPICE, each circuit is simulated exhaustively in terms of its supply- and input-vector combinations, where input pins' voltage values are chosen among the list of the available supply values. From the obtained simulation traces, we determine whether a transistor can be in EOS. This is done, for each device, by checking whether there is a case among all simulation traces where the voltage applied exceeds device specification.

5.2.2.1 Functional Comparison

Table 5.4 compares obtained results from different approaches. It shows the number of devices that fall under each combination of verdicts obtained from different methods.

	static	$\mathcal{S}^s$	$\mathcal{S}^t$	$\mathcal{S}^q$	SPICE	$\lvert\mathbf{D}\rvert$ $\mathcal{R}_{\mathbf{I}}$ strict for $\mathcal{S}^q, \mathcal{S}^t, \mathcal{S}^s$	$\lvert\mathbf{D}\rvert$ $\mathcal{R}_{\mathbf{I}}$ relaxed for $\mathcal{S}^q, \mathcal{S}^t, \mathcal{S}^s$
1	N	N	N	N	N	26	26
2	E	N	N	N	N	162	162
3	E	E	E	N	N	152	149
4	E	E	E	E	N	494	497
5	E	E	E	E	E	2310	2310
6	all other combinations					0	0

Table 5.4: EOS analysis results (timeouts excluded). 'E' denotes an Error, and 'N' denotes absence of EOS (No error).

Note that we only report in Table 5.4 results that did not timeout with any approach for both $\mathcal{R}_{\mathbf{I}}$ strict and $\mathcal{R}_{\mathbf{I}}$ relaxed. That leaves us with $\simeq 8.3\,\%$ of the whole benchmark where no approach times-out. Execution times and timeouts are discussed further in Section 5.2.2.2. The columns of the table are arranged so that precision increases as we go from left (static) to right (SPICE). The 'static' approach is coarse and doesn't consider any circuit semantics, which makes it the most false alarm prone approach. SPICE is the most precise approach, in the sense that it is the closest to the real circuit behavior, but is not comparable formally with any of our semantics. As $\mathcal{S}^t$ is more permissive compared with $\mathcal{S}^q$ (Figure 4.33, page 72), all circuits proved correct with respect to EOS using $\mathcal{S}^t$ are also correct in $\mathcal{S}^q$, but conversely $\mathcal{S}^t$ may raise false alarms that do not appear with $\mathcal{S}^q$. In practice, when $\mathcal{S}^t$ answers with 'N' the analysis is conclusive without the need to run an analysis with $\mathcal{S}^q$. Any approach that is more precise than $\mathcal{S}^t$ necessarily answers with 'N' as well in this case. The experimental results summarized in Table 5.4 hence verify experimentally Theorem 4.7.6 (page 74). One can note that we experimentally find the same set of EOS devices

on this benchmark with $\mathcal{S}^s$ and $\mathcal{S}^t$, although we proved that no inclusion relation between the two semantics holds in theory (Theorems 4.7.1 and 4.7.2, page 73).

Each configuration count (the $|\mathbf{D}|$ column of Table 5.4) is displayed for two choices of the input restriction clause ($\mathcal{R}_\mathbf{I}$): when strict, it restricts the inputs to the available supply values ($\mathcal{R}_\mathbf{I}$ strict), and when relaxed, it allows any input to have any voltage value in the continuous range between the minimum and maximum available supplies in the circuit ($\mathcal{R}_\mathbf{I}$ relaxed). Note that relaxing $\mathcal{R}_\mathbf{I}$ is only applicable to the semantics-based analyses—where the state space is explored symbolically. In this regard, semantics-based analyses have the advantage of exploring an infinite number of configurations, which is not achievable with the use of simulation. It is possible for some circuit configurations (e.g., a configuration that leads to a specific device to be in EOS) to be only achievable by relaxing $\mathcal{R}_\mathbf{I}$, e.g., three devices become erroneous by relaxing $\mathcal{R}_\mathbf{I}$ using $\mathcal{S}^q$ (see rows 3 and 4, Table 5.4). Manual investigation shows that these three circuits are indeed buggy circuits, with a net connected to supply and ground only through near-threshold devices, hence almost floating, and connected to the gate of a transistor. This configuration is very sensitive to slight voltage or transistor parameter change, and behaves differently in SPICE and in our linear approximation. It is a good thing to raise a warning in such case since the circuit should indeed be fixed.

5.2.2.2 Performance Comparison

Experiments are run on a 32 GB system memory machine using a single core clocked at 3.2 GHz, where all SPICE- and semantics-based analyses are run with a 500 s time-limit. The 'static' approach is not concerned with a time-limit, as it is very fast to run: it always terminates. SPICE analyses consist in exhaustive simulation of the circuit with respect to input and supply stimuli, while semantics-based analyses consist in running Algorithm 1, varying the semantics used.

Figure 5.10 shows the percentage of analyses that terminate (no timeout), grouped by circuit size. We notice that relaxing $\mathcal{R}_\mathbf{I}$ leads to more timed-out cases for $\mathcal{S}^q$, and that the performance of $\mathcal{S}^s$ and $\mathcal{S}^t$ are similar. They both always terminate before the 500 s time-limit for circuits composed of less than 1000 devices. SPICE, however, almost always performs worse than semantics-based analyses. Moreover, simulations may time-out even for the smallest circuits of the case-study. This is primarily linked to the large number of input combinations to simulate.

Note. Relaxing $\mathcal{R}_\mathbf{I}$ does not apply to SPICE as it cannot consider continuous intervals of input and supply stimuli, hence it was not plotted (Figure 5.10).

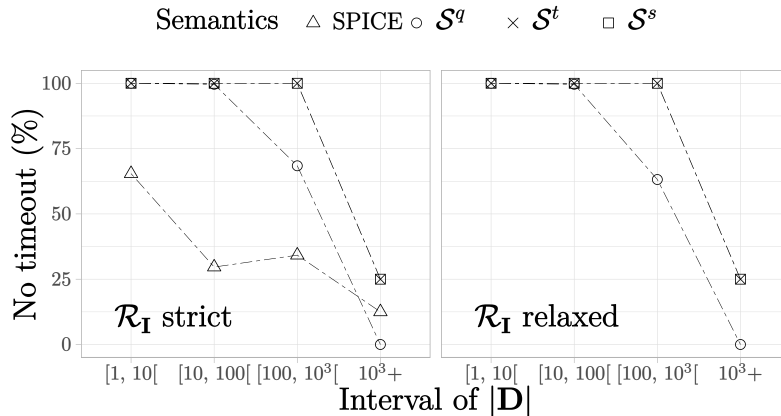


Figure 5.10: Termination of EOS analyses for a 500 s time-limit, with different approaches.

Figure 5.11 shows the solving time using SPICE, grouped by the total number of configurations simulated for each case. The total solving time is globally correlated with the number of simulated configurations of the same circuit, which is indeed conceivable.

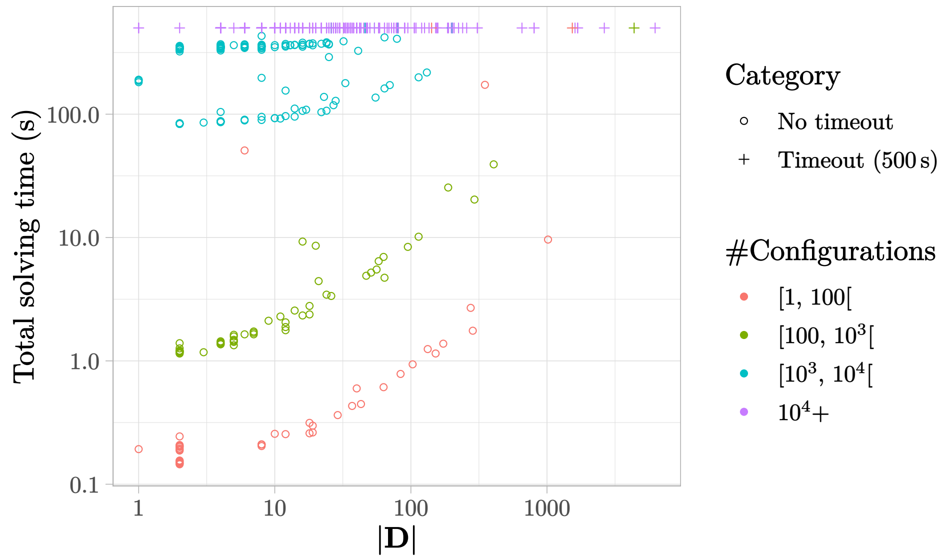


Figure 5.11: SPICE total simulation time, per circuit, for EOS detection. The '+' marks indicate circuits for which analysis did not terminate after the 500 s time-limit is reached.

Figure 5.12 shows the solving time with each of the semantics-based approaches, with $\mathcal{R}_\mathbf{I}$ strict. The analysis may involve multiple invocations of the SMT solver, and the more solver runs are involved, the more the total solving time might grow. When an analysis consists in a single solver run, it means that no EOS error is found (UNSAT), or that all devices in the circuit are found to be in EOS with the same circuit configuration (which may be trivial in the case of a single-device circuit). As previously observed in Figure 5.10, $\mathcal{S}^q$ results in more cases of timeout compared with $\mathcal{S}^s$ and $\mathcal{S}^t$.

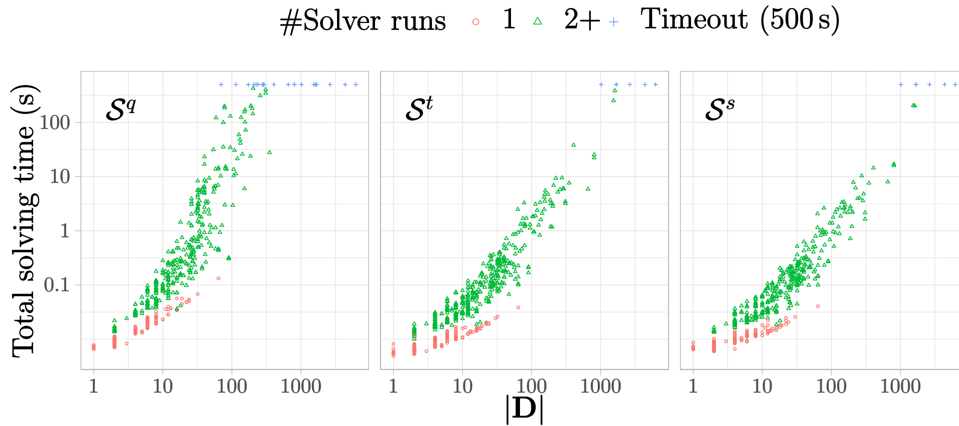
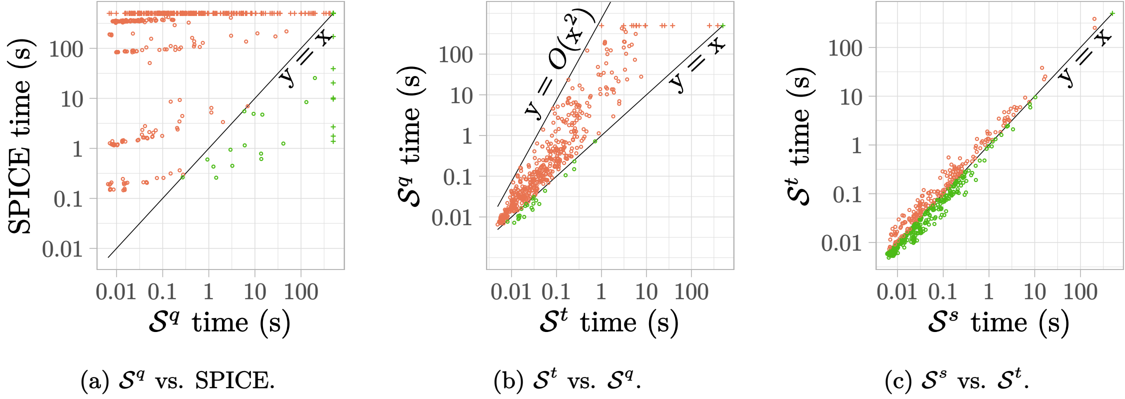


Figure 5.12: Total solving time for EOS analyses, with $\mathcal{R}_\mathbf{I}$ strict, compared among different circuit semantics. The '+' marks indicate circuits for which analysis did not terminate after the 500 s time-limit is reached.

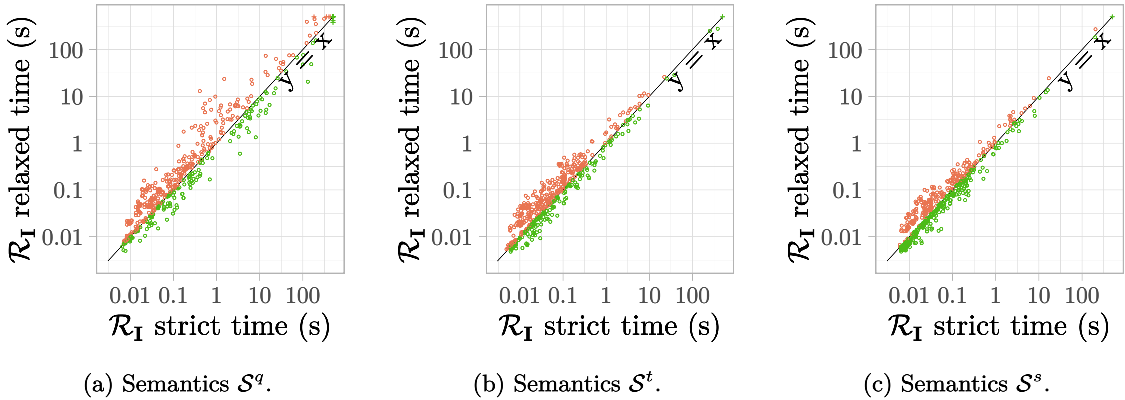
Figure 5.13 shows the overhead in terms of the solving time caused by using one approach over the other, with $\mathcal{R}_\mathbf{I}$ being strict in all cases. Not only does SPICE time-out much more often, but it is also much slower than all three semantics ($\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$) for the vast majority of cases. We observe a pattern of separate plateaux in the SPICE solving time (Figure 5.13a), where each plateau roughly corresponds to the same number of simulated configurations of different circuits. $\mathcal{S}^q$ is globally slower compared with $\mathcal{S}^t$ (Figure 5.13b), while the variants $\mathcal{S}^t$ and $\mathcal{S}^s$ of the switch-based semantics remain very similar in terms of their runtime (Figure 5.13c).



(a) $\mathcal{S}^q$ vs. SPICE. (b) $\mathcal{S}^t$ vs. $\mathcal{S}^q$. (c) $\mathcal{S}^s$ vs. $\mathcal{S}^t$.

Figure 5.13: Two-by-two semantics' comparison of solving time (s), with $\mathcal{R}_\mathbf{I}$ strict.

Using a strict or relaxed version of $\mathcal{R}_\mathbf{I}$ has a relatively low impact on solving time (see Figure 5.14): relaxing it only slows the solver down in a slight majority of cases. This is a key advantage of our approach, which is able to explore a continuous and infinite state space in reasonable time. That is not possible with a simulation-based approach.



(a) Semantics $\mathcal{S}^q$. (b) Semantics $\mathcal{S}^t$. (c) Semantics $\mathcal{S}^s$.

Figure 5.14: Comparison of semantics' solving time (s)
with respect to the choice of $\mathcal{R}_\mathbf{I}$ ($\mathcal{R}_\mathbf{I}$ strict vs. $\mathcal{R}_\mathbf{I}$ relaxed).

5.2.2.3 Brief Comparison With Related Works

Regarding the quantitative SAT-based approach introduced by Miller and Brewer [2013]—Chapter 3, it was not possible to experimentally compare their approach with our semantics-based analysis, as they do not provide a tool. Nevertheless, their experimental evaluation—which consists in inquiring a solver to find specific circuits' operating points—shows a limited scalability. Their analysis is run on a test case composed of 20 NMOS transistors, 20 PMOS transistors, and 10 resistors, and is analyzed in 20 min, while we analyze most of the circuits of the same size in less than a second (see Figure 5.12), with variations in the performance based on the circuit semantics used. This limitation is probably due to the fact that they bit-blast numerical variables, unlike our approach which uses an SMT solver.

5.2.3 Future Extension: Optimizing Enumeration of Errors

As part of the EOS error identification case study, we used Algorithm 1 (page 88) to enumerate all EOS errors in each circuit. The algorithm first checks (by asking the SMT solver) whether there are any EOS errors in the circuit, and, while the answer is 'yes', it keeps inquiring the solver

about the remaining devices (the ones which have not been checked yet), until no more devices in EOS are found. As stated beforehand, such algorithm is not optimized—in the sense that it does not ensure the number of iterations inquiring the SMT solver is minimal. One way it can be optimized is by involving optimization modulo theories (OMT) solving, to maximize the number of EOS devices found at each step. So, in addition to checking whether there exists at least one device $X \in \mathbf{D}_{\texttt{MOS}}$ which violates the 'absence of EOS' ($\mathcal{A}_{\texttt{EOS}}$) property (i.e., $\mathcal{A}_{\texttt{EOS}}(X)$ is $\bot$), the solver takes into account the objective:

$$\texttt{maximize}\left(\sum_{X \in \mathbf{D}_{\texttt{MOS}}} \mathbb{1}(\mathcal{A}_{\texttt{EOS}}(X))\right),$$

with $\mathbb{1}\colon \mathbb{B} \longrightarrow \{0,1\}$ being the Indicator function, defined in $\mathbb{1}_{\text{def}}$.

$$\begin{aligned} \mathbb{1}\colon \mathbb{B} &\longrightarrow \{0,1\} \\ x &\longmapsto \begin{cases} 1 & \text{if } x = \top \\ 0 & \text{if } x = \bot \end{cases} \end{aligned} \tag{$\mathbb{1}_{\text{def}}$}$$

Different versions of the objective function can be imagined, where, for example, different weights are associated with finding errors on particular devices. These will highly depend on the use cases considered, and on the knowledge at hand regarding the circuits under verification.

5.3 Conclusion

Different kinds of circuit semantics, introduced in Chapter 4, were experimentally evaluated in this chapter. Two use cases were considered: (1) identification of missing level-shifters in multi-supply designs, and (2) detection of circuit states involving electrical overstress on some devices. Circuit semantics are compared in terms of performance and precision, on the basis of the analysis results they yield. Having several semantics available allows for performance/precision trade-offs: $\mathcal{S}^s$ is a simplified switch-based approach, the $\mathcal{S}^t$ variant is a slight improvement with marginal performance cost ($\mathcal{S}^t$ should generally be used over $\mathcal{S}^s$), while $\mathcal{S}^q$ allows a much more precise analysis with a larger slow-down but remains much faster than exhaustive simulation. A production tool can combine several approaches, typically running the fast $\mathcal{S}^t$ analysis and possibly falling-back to $\mathcal{S}^q$ only when $\mathcal{S}^t$ fails to prove the absence of a property.

6

Time-Dependent Dielectric Breakdown Worst Case Reliability Analysis

Aging is a phenomenon every integrated circuit (IC) is subject to. Eventually, all circuits will become out of use, as they become less reliable—ceasing to fulfill their intended functionality. Different failure mechanisms explain different forms of circuit degradation. Each failure mechanism describes the evolution of the lifetime of an IC with respect to its conditions of operation, and the technological parameters of its constituting devices. In this chapter, we propose a method for the analysis of circuits lifetime, with respect to the time-dependent dielectric breakdown (TDDB) failure mechanism. TDDB has a statistical nature, where circuit degradation depends on how long the circuit remains in a specific steady state. The circuit semantics developed in this thesis particularly reason about steady states, hence the choice to address such failure mechanism. We use different circuit semantics ($\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$) introduced in Chapter 4, and we assess their usability in the context of TDDB reliability analysis. Analyzing circuit reliability is a broad topic which can be studied under many facets. The use case considered in this thesis consists in leveraging an optimization modulo theories (OMT) solver to yield the circuit state which leads to the fastest aging. This makes it possible to identify the worst steady-state of a circuit, and the corresponding reliability given an aging model, which is valuable information for circuit designers. Based on the circuit semantics used, we either obtain a safe lower-bound of circuit reliability, or a mere (not necessarily safe) approximation of it. Such goal is formalized as an optimization problem (in terms of the variables of the formula encoding the circuit). The optimization problem is then solved with the help of optimization features of modern satisfiability modulo theories (SMT) solvers, like Z3's νZ OMT engine [Bjørner and Phan, 2014]. We also show the potential of the approach to identify the age after which a circuit's reliability falls under a given threshold in the worst case, using binary search. Both applications presented in this chapter are supplemented with experimental benchmarks conducted on a database of industrial circuits. We rely on those benchmarks to draw conclusions regarding performance versus precision tradeoffs.

6.1 Overview of the Approach

For a single transistor, finding the least reliable state boils down to identifying the voltages on the gate, source, and drain which correspond to the lowest reliability with respect to the failure mechanism considered. The core difference with the error detection approach seen in Chapter 5, e.g., identification of electrical overstress (EOS, Section 5.2), lies in the kind of questions asked. In the EOS identification approach we ask "is it possible for the voltage applied on a transistor to exceed a given limit?"—which is a satisfiability problem. Here, the question would be "what is *the* maximum voltage applied on the transistor?"—which is an optimization problem. A lower bound for

a transistor's reliability can be found by using static voltage propagation to determine a hypothetical worst state. However, doing so for all transistors in a circuit would lead to unreasonably pessimistic results, for it is rarely the case that all transistors are locally in their worst possible state when the global circuit state is considered. It is even often impossible for such state to exist. For example, if one of the two transistors of the Inverter circuit (Figure 3.2, page 20) is in its worst possible state, the second transistor is necessarily in its best state. The issue with that result is that it is hardly exploitable by circuit designers, as the actual lower bound for circuit reliability would be much more higher. Also, the fact that the corresponding circuit configuration does not correspond to an actual state makes it less 'debugger-friendly'. For instance, designers cannot really rely on such approach to obtain some helpful *hints* for identifying legitimate circuit states susceptible to correspond to the worst case. It is thus important to increase such lower bound so that it is as close as possible to the actual lower bound. To do so, circuit semantics must be considered.

Our goal is to find the least reliable circuit state so that circuit designers can optimize it, and to simply provide guarantees that any other circuit state is necessarily more reliable than the worst case. We hence aim at finding the state which minimizes circuit reliability.

Given a circuit description and reliability models of its devices, our approach consists in (1) encoding the circuit into a logic formula $\mathcal{S}$ that is built from circuit semantics, and (2) encoding the reliability of the circuit into an expression $\mathscr{R}$ which can be minimized, under the constraint that $\mathcal{S}$ is satisfied.

The νZ extension of the Z3 SMT solver is dedicated to OMT. It lets us formulate objective functions (e.g., reliability $\mathscr{R}$) and combine them with the hard SMT constraints (e.g., semantics $\mathcal{S}$). A high-level overview of the corresponding analysis flow is recalled in Figure 6.1. Note that the encoding of device reliability models must be solver-friendly, especially when they are described with non linear functions (reliability models adopted are presented in Section 6.2). We opt for conservative linear approximations of those functions, for a naive non-linear encoding relies on intractable theories, which would make it hardly solvable. We therefore remain in the theory of linear real arithmetic (LRA). The corresponding encodings, as well as implementation details, are presented in Section 6.3.

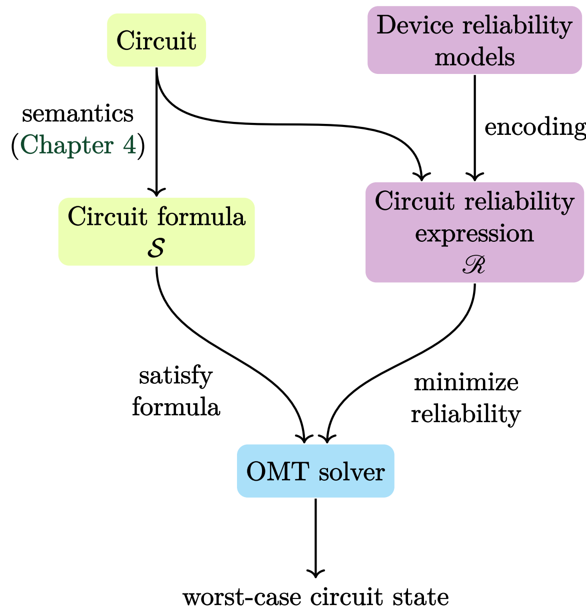


Figure 6.1: (Reminder) OMT based worst-case reliability analysis flow.

6.2 Time-Dependent Dielectric Breakdown Failure Mechanism and JEDEC Model

Time-dependent dielectric breakdown (TDDB), also called time-dependent oxide breakdown (TDOB), is a failure mechanism that has a strong statistical nature. It depends on the applied stress on a transistor, expressed either in terms of voltage or current [Verweij and Klootwijk, 1996]. Failures caused by TDDB are catastrophic ones, meaning that transistors cannot recover from them—the effect being cumulative over time, i.e., the reliability of a transistor depends on for how long it is being stressed. Since we model catastrophic failures, we are not interested in modeling the evolution of the behavior of transistor, but the evolution of the probability of failure as a function of transistor state and age. Before failure, we consider the behavior as constant, and after failure, the transistor ceases to be functional and modeling its behavior is pointless. In addition, with this type of failures, it is sufficient for one device to fail to trigger a failure of the entire circuit in the absence to a fault-tolerant mechanism. In other words, the problem addressed in this chapter is to find the steady-state of a circuit that leads to the worst reliability. As done in previous chapters, we model the behavior of transistors to specify legitimate steady-states, but not to model the *effect* of aging itself. We do not provide a new reliability model, but use an existing one (see Section 6.2.2). Our main contribution is to express the reliability function of such model in terms of linear real arithmetic (LRA).

Note. The TDDB failure mode is only defined for transistors and is not applied to other devices (resistors and diodes) defined in circuit semantics. This chapter, therefore, only considers transistors when studying TDDB.

This section presents the existing JEDEC model. We later show how we use it in our analysis.

6.2.1 Stress Voltage Applied on Transistors

The aging rate of a transistor depends largely on the stress voltage applied on it. We denote $\delta_{\mathcal{V}}(M)$, defined in $\delta_{\mathcal{V}\text{def}}$ below, the maximum applied stress voltage on a transistor M, for some voltage valuation $\mathcal{V}$. It corresponds to the maximum gate-to-source or gate-to-drain voltage difference (as source and drain are physically symmetrical and may be used interchangeably) in a given circuit steady-state. It makes sense to only consider the maximum applied voltage, because it is the one which counts when measuring the degradation of the device (e.g., in terms of time-to-failure).

Note that for PMOS devices, the applied voltage is either gate-to-source or gate-to-drain voltage preceded by a negative sign, since it is the negative values of $\mathcal{V}_{\text{GS}}(M)$ and $\mathcal{V}_{\text{GD}}(M)$ that activate them. The lowest such voltage difference, the fastest is PMOS degradation.

$$\delta\colon (\mathbf{N} \longrightarrow \mathbb{V}) \longrightarrow (\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{V})$$

$$\mathcal{V} \longmapsto \delta_{\mathcal{V}}, \text{where: } \delta_{\mathcal{V}}(M) = \begin{cases} \max\left(\mathcal{V}_{\text{GS}}(M), \mathcal{V}_{\text{GD}}(M)\right) & \text{if } M \in \mathbf{D}_{\texttt{NMOS}} \\ \max\left(-\mathcal{V}_{\text{GS}}(M), -\mathcal{V}_{\text{GD}}(M)\right) & \text{if } M \in \mathbf{D}_{\texttt{PMOS}} \end{cases} \qquad (\delta_{\mathcal{V}\text{def}})$$

6.2.2 Time to Failure

TDDB probability is expressed, for each device, as a function of time-to-failure (TTF). Its value depends on the operating conditions of the device, as well as its technological parameters. A transistor's TTF is mainly determined by its applied stress voltage.

To compute the degradation of a transistor M associated with its applied voltage $\delta_{\mathcal{V}}(M)$, one needs to know the TTF function specific to that device. Typically, such functions are provided by device foundries, taking into account technological parameters and manufacturing details. Those models remain proprietary, and using them is often systematically subject to copyright. We thus rather use an open model, standardized among all devices. This choice also contributes to simplifying the encoding of reliability later, and, perhaps, helps enhancing the readability of this chapter. In an industrial context, the approach may be tailored so that actual TTF models of different devices

are used, as long as they are defined in terms of nets' voltages ($\mathcal{V}$) or device currents ($\mathcal{I}$). The TTF model we use in this work is the one from the Joint Electronic Device Engineering Council (JEDEC).[1]

The JEDEC V-model[2] ('V' stands for Voltage, since in our circuit semantics we reason about voltages) for TTF [JEDEC Publication JEP122H, 2016] defines, for each transistor, the TTF with Function $\alpha\colon \mathbb{V} \longrightarrow \mathbb{R}_{>0}$ based on some applied voltage v (see α_{def}). It is expressed using device parameters E_a (activation energy), γ (voltage acceleration), and T (operating temperature). A_0 represents an arbitrary scale factor dependent upon materials and process details, and K is Boltzmann's constant ($K = 1.380\,649 \times 10^{-23}\,\mathrm{J\,K^{-1}}$). We define the parameters used, later, as part of the use case considered for experimental evaluation, in Section 6.4.

$$\begin{aligned} \alpha\colon \mathbb{V} &\longrightarrow \mathbb{R}_{>0} \\ v &\longmapsto A_0 \times e^{\left(\dfrac{E_a}{K \cdot T}\right)} \times e^{-\gamma \cdot v} \end{aligned} \qquad (\alpha_{\text{def}})$$

6.2.3 Device Reliability

Reliability $\mathscr{R}_{\text{MOS}}(v,t)$ of a transistor is the probability that this transistor remains reliable (i.e., no fault occurring on this particular transistor) at age t when constantly applied a stress voltage v. It follows a Weibull distribution, with factor β [Ghetti, 2004] (see $\mathscr{R}_{\text{MOS}\text{def}}$). For simplicity of the explanation, we consider that all transistors follow the same $\mathscr{R}_{\text{MOS}}$ law, i.e., having the same Weibull factor β. Still, for different transistor technologies, different reliability laws can be used while keeping the same approach. We illustrate, in the following section, how different Weibull factors may impact the evolution of a transistor's (and therefore a circuit's) reliability as a function of age—emphasizing the fact the problem at hand fundamentally depends on age. We denote $\mathbb{T} = \mathbb{Q}_{\geq 0}$ the set of age valuations, and $\mathbb{P} = [0,1]$ the set of probabilities.

$$\begin{aligned} \mathscr{R}_{\text{MOS}}\colon \mathbb{V} \times \mathbb{T} &\longrightarrow \mathbb{P} \\ (v,t) &\longmapsto e^{\left(-\dfrac{t^{\beta}}{\alpha(v)}\right)} \end{aligned} \qquad (\mathscr{R}_{\text{MOS}\text{def}})$$

6.2.4 Circuit Reliability

For a specific steady-state, we express the reliability of the whole circuit after some age t as the product of individual reliabilities of the transistors composing the circuit, evaluated all at the same age $t \in \mathbb{T}$. This is valid under the hypothesis that individual device probabilities of reliability are independent. In fact, failure of one device may induce failure of other devices in the circuit, but such phenomenon is out of scope of our study, because our focus is on the first failure that happens. For this end, we assume circuit reliabilities to be independent, hence the validity of computing circuit reliability as the product of device reliabilities.

Function $\mathscr{R}_{\text{CIRC}}$ (see $\mathscr{R}_{\text{CIRC}\text{def}}$) maps a set of applied stress voltages in a circuit (given by some function $\sigma\colon \mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$), and age, to a reliability value of the whole circuit.

$$\begin{aligned} \mathscr{R}_{\text{CIRC}}\colon (\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}) \times \mathbb{T} &\longrightarrow \mathbb{P} \\ (\sigma,t) &\longmapsto \prod_{M \in \mathbf{D}_{\mathtt{MOS}}} \mathscr{R}_{\text{MOS}}(\sigma(M),t) \end{aligned} \qquad (\mathscr{R}_{\text{CIRC}\text{def}})$$

1 https://www.jedec.org

2 An equivalent model which describes TDDB in terms of the electric field, named the E-model, exists. It is however not suited to our case.

For the sake of illustration, we consider the two-input Inverter circuit in Figure 6.2, composed of a PMOS device (M1) and an NMOS device (M2), with their shape parameter (i.e., Weibull factor) being respectively $\beta_1 = 1$ and $\beta_2 = 2$. The circuit's reliability curves are shown in Figure 6.3 for two circuit states. In State 1, input A is assigned 0 V and input B is assigned 2 V, while in State 2, A is assigned −8 V and B is assigned 0 V.

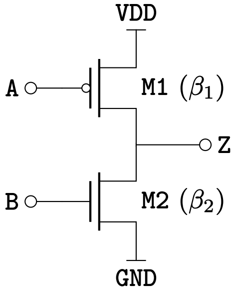


Figure 6.2: Two-input Inverter, with different PMOS and NMOS device shape parameters.

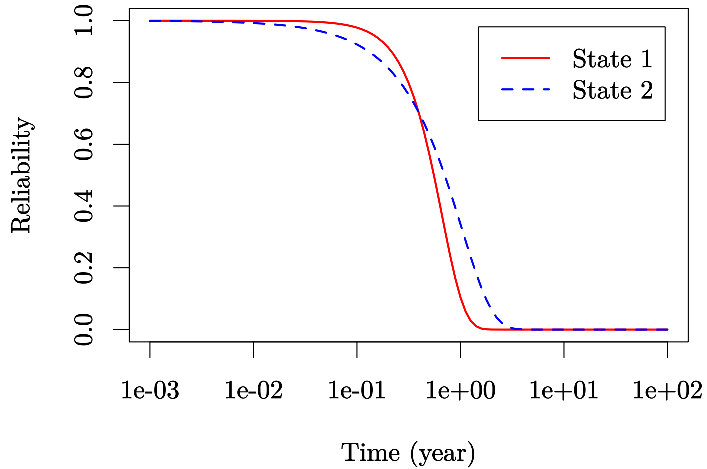


Figure 6.3: Reliability of the two-input Inverter (Figure 6.2) with respect to the circuit state.

This example shows that the reliability curves are not necessarily totally ordered (the curves may cross each other) among different circuit states, namely, when different devices have different Weibull factors. This means that the worst case circuit state (with respect to reliability) depends on the age (typically, in number of years) at which reliability is to be evaluated.

6.2.5 Applications and Related Work

A straightforward application of $\mathscr{R}_{\text{CIRCdef}}$ using the JEDEC model would be to compute, for each transistor M, the maximum stress voltage $\sigma(M)$ the circuit's structure allows, regardless of the possibility of such configuration in one of the circuit states. Of course, as previously mentioned in Section 6.1, this would make for an unrealistically pessimist estimation. Still, it will always yield a safe lower-bound for circuit reliability. We refer to this as the 'baseline approach'. Computation of maximum stress voltages on such basis is feasible via the use of a static voltage propagation algorithm—annotating each net with the supplies it can statically reach. On a regular Inverter circuit (Figure 3.2, page 20), the baseline approach would consider both devices (Mp and Mn) to be fully activated, with $\delta_{\mathcal{V}}(\texttt{Mp}) = \delta_{\mathcal{V}}(\texttt{Mn}) = \mathcal{V}(\texttt{VDD})$. There is no circuit configuration where this is possible (due to the functionality of the Inverter itself), but this approximation can help compute a safe lower-bound of the Inverter's reliability. In normal conditions, either one of the two devices is fully activated while the other is turned off, or both devices are partially activated—when the input A is at some intermediate voltage between $\mathcal{V}(\texttt{GND})$ and $\mathcal{V}(\texttt{VDD})$. To the best of our knowledge, such voltage propagation based approach (i.e., the baseline) was never published. We implement it solely as a basis for comparison, as we discuss later as part of the experimental results conducted in Section 6.4.6. As the reader may have already guessed, the baseline approach is very fast: once static voltage propagation is done, the reliability is computed locally for each transistor, giving us thus a safe lower bound for circuit reliability. The evidence to this is shown in Section 6.4.6. The main limitation of this approach relies in ignoring the semantics of transistors (and hence that of the whole circuit). Stress voltages ($\sigma(M)$) considered may not even follow basic electronics rules

like Kirchhoff's voltage law. So, a circuit state that seems to minimize reliability is very unlikely a correct steady-state with respect to physics.

The antithetical approach to the baseline is one which emphasizes the preservation of circuit semantics the most, while trying to identify the circuit's worst case. Intuitively, one could think of simulation as a suited tool to implement a semantics-aware reliability analyzer. Such analyzer would run *exhaustive* simulation (although true exhaustive simulation is nearly intractable, when continuous intervals of input values are to be considered), and identify the worst case upon termination of the simulation campaign, e.g., using some circuit simulator, like SPICE. There surely exist reliability simulators built on top of SPICE [Ko, 1989; Lee et al., 1990]. Examples of commercial reliability simulators include CADENCE LEGATO [Art Schaldenbrand (Cadence), 2024] and SYNOPSYS PRIMESIM [Synopsys, 2024]. No doubts simulation programs are the top industry-standard tools when it comes to precisely characterizing the behavior of circuits. However, they fall short of handling large designs, as they require large computational resources to solve differential algebraic systems of equations used for modeling [Nichols et al., 1994; Lang, 2010]. Moreover, as to SPICE, the state space of a circuit generally grows exponentially with respect to the size of the input vector. It is even infinite if the inputs range over a continuous domain. This makes it unfeasible to exhaustively simulate a circuit. In fact, SPICE has no representation (namely, symbolic) of domains of voltages. Besides, exhaustive simulation doesn't natively support full state space exploration for circuits with memory points, as each circuit state depends on both the input vector and the previous state.

In contrast, our approach uses an encoding of the semantics of the circuit, and expresses the objective function $\mathscr{R}$ as a function of the state. The OMT engine then finds the worst case reliability among steady-states matching the circuit's semantics only, which is more precise than the baseline approach—as only the relevant subset of circuit steady-states are considered. It is important to note that both the baseline approach and our semantics-based analyses compute a lower-bound for reliability at a specific age—our approach being at worst as pessimistic as the baseline, if the semantics used are sound.

6.3 Circuit Reliability Analysis

The reliability of a circuit is the probability that the circuit remains reliable with respect to a specific failure mechanism. The failure mechanism we consider in this work is TDDB. The reliability of the circuit can be expressed for a circuit state $\mathcal{V}\colon \mathbf{N} \longrightarrow \mathbb{V}$ and age $t \in \mathbb{T}$, using Function $\delta_{\mathcal{V}}\colon \mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{V}$ (from $\delta_{\mathcal{V}\text{def}}$). It is denoted $\mathscr{R}(\mathcal{V}, t)$, defined in $\mathscr{R}_{\text{def}}$. It is a wrapper around $\mathscr{R}_{\text{CIRC}}$, which also helps in simplifying the notation.

$$\begin{aligned} \mathscr{R}\colon (\mathbf{N} \longrightarrow \mathbb{V}) \times \mathbb{T} &\longrightarrow \mathbb{P} \\ (\mathcal{V}, t) &\longmapsto \mathscr{R}_{\text{CIRC}}(\delta_{\mathcal{V}}, t) \end{aligned} \tag{$\mathscr{R}_{\text{def}}$}$$

6.3.1 Formulation of the Worst Case Reliability

The number of possible circuit states is infinite as soon as one input can range over a continuous interval of voltage values, and is otherwise usually very large with respect to the size of the circuit. In practice, it is not feasible to find the worst case state from exhaustive enumeration of all circuit states. Hence the choice to base our analysis on symbolically determining the circuit state that leads to the fastest degradation, i.e., the worst case state with respect to reliability. Doing so, we never explicitly enumerate the state space of the circuit. Our goal is to compute such state by minimizing the objective function $\mathscr{R}$ representing the circuit's reliability.

As the probability of failure ($1 - \mathscr{R}(\mathcal{V}, t)$, the dual of reliability) is a function of age, and as functions $\mathscr{R}(\mathcal{V}, t)$ of different circuit states $\mathcal{V}\colon \mathbf{N} \longrightarrow \mathbb{V}$ are not necessarily totally ordered (see Figure 6.3), the problem of finding 'the absolutely worst case reliability' is not well-defined. We therefore restrict our analysis, at first, to a sub-problem, which is to find the worst case reliability at a specific age. So, we set the value of the age for which we minimize $\mathscr{R}(\mathcal{V}, t)$ at $t = t_{\text{ref}}$. The problem is now to find a valuation $\mathcal{V}$ which corresponds to the worst circuit state. Later, in Section 6.4.7, we

consider the opposite problem: finding the age at which a given reliability in reached in the worst case.

6.3.2 Minimizing Reliability Under SMT Constraints

The OMT solving phase (Figure 6.1) is delegated to an external solver. We use the νZ OMT engine by Bjørner and Phan [2014] to solve our optimization problem in the space of steady-states which preserve the semantics of the circuit. It can be asked, bluntly, to

$$\texttt{minimize}(\mathscr{R}\,(\mathcal{V}, t_{\text{ref}})).$$

Surely, the solver will have a hard time trying to optimize such non-linear expression. Generally, SMT formulas involving non-linear theories over the reals (i.e., non-linear real arithmetic (NRA)) are harder to solve [Gao et al., 2013]. We want to avoid the use of such theories, and so, we transform the non-linear expression $\mathscr{R}\,(\mathcal{V}, t_{\text{ref}})$ into a linear form.

Using the natural logarithm. First, we apply the natural logarithm function to the quantity to minimize. Optimizing the resulting expression, $\ln(\mathscr{R}\,(\mathcal{V}, t_{\text{ref}}))$, is fundamentally the same as optimizing $\mathscr{R}\,(\mathcal{V}, t_{\text{ref}})$, as the natural logarithm is strictly monotonically increasing. Doing so, we turn multiplications (of device reliabilities, in $\mathscr{R}_{\text{CIRCdef}}$) into additions, getting one step closer to a linear expression. The quantity to minimize is now

$$\ln(\mathscr{R}\,(\mathcal{V}, t_{\text{ref}})) = \sum_{M\,\in\,\mathbf{D}_{\texttt{MOS}}} \ln(\mathscr{R}_{\text{MOS}}(\delta_{\mathcal{V}}(M), t_{\text{ref}}))\,.$$

Approximating with a tabulated function. Second, as

$$\omega = \ln(\mathscr{R}_{\text{MOS}}(\delta_{\mathcal{V}}(M), t_{\text{ref}})) = \frac{-t_{\text{ref}}^{\beta}}{A_0 \times e^{\left(\dfrac{E_a}{K \cdot T}\right)} \times e^{-\gamma\,\cdot\,\delta_{\mathcal{V}}(M)}}$$

is not linear either, we consider a piecewise affine function to approximate it. Figure 6.4 presents our piecewise affine approximation function, denoted $\Omega\colon \mathbb{V} \longrightarrow]-\infty, 0]$. It is defined on a global interval $]V_{\min}, V_{\max}]$ representing the possible applied voltage stress values $\delta_{\mathcal{V}}(M)$ on a transistor M in a circuit. The granularity of the approximation depends on the choice of intervals I_k of the piecewise affine approximation, e.g., the intervals can be evenly-divided, or chosen in other custom ways. For negative voltage values, we approximate the value of ω with a constant lower bound, corresponding to the evaluation of ω at $\delta_{\mathcal{V}}(M) = 0\,\text{V}$. Note that the linearised function remains sound regardless of how intervals I_k are chosen, as such approximation is below the actual curve. The choice of intervals I_k only has an impact on precision (and, consequently, on performance).

We choose to define Intervals I_k of the piecewise affine approximation of the logarithm of device reliabilities on the basis of the available supply voltages in the circuit. Interval boundaries correspond to the possible differences between any two available supplies, for positive applied stress voltage values. The bound $\ln(\mathscr{R}_{\text{MOS}}(0\,\text{V}, t_{\text{ref}}))$ is used as an approximation of the function for negative values $\delta_{\mathcal{V}}(M) \leq 0$. We refer to this method of choosing intervals as the '**adaptive**' approach. It is the approach we use for the remainder of this chapter, unless specified otherwise. This choice was made after running several benchmarks with different linearisation approaches, by assessing speed and gain in terms of precision. The adaptive approach have shown to be the best choice, compared with other linearisation strategies. We detail such experiments and discuss obtained results, later, in Section 6.4.5. With the adaptive approach, in case neither short-circuits nor floating nets are possible in a circuit's steady state, the reliability obtained with the approximation function is exactly the same (modulo rounding errors) as if the actual function was used. This is notably the case for pure logic (like CMOS-based) parts of the circuits. An example of this is shown in Table 6.1.

Key sampling values	Intervals I_k	Logarithm of reliability
0 V	$]-\infty, 0]$	constant value: $\ln(\mathscr{R}_{\text{MOS}}(0\,\text{V}, t_{\text{ref}}))$
$1.2\,\text{V} - 0\,\text{V} = 1.2\,\text{V}$	$]0, 1.2]$	piece-wise affine approximation
$3.3\,\text{V} - 1.2\,\text{V} = 2.1\,\text{V}$	$]1.2, 2.1]$	piece-wise affine approximation
$3.3\,\text{V} - 0\,\text{V} = 3.3\,\text{V}$	$]2.1, 3.3]$	piece-wise affine approximation

Table 6.1: Illustration of how linearisation intervals for reliability are chosen according to the adaptive approach, for a circuit having supply values 0 V, 1.2 V, and 3.3 V.

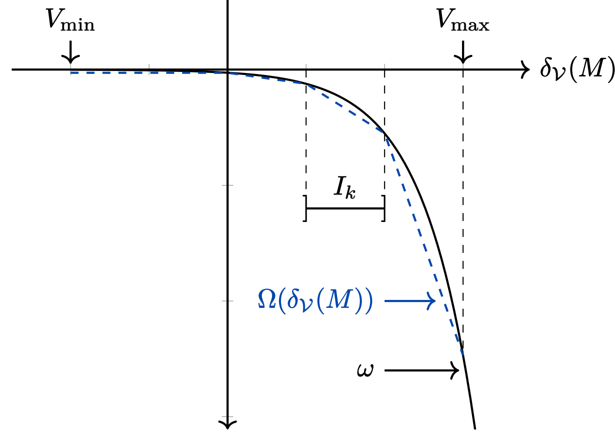


Figure 6.4: Piecewise affine approximation of $\omega = \ln(\mathscr{R}_{\text{MOS}}(\delta_{\mathcal{V}}(M), t_{\text{ref}}))$.

The evaluation of ω at interval bounds is performed numerically using OCaml's native 64-bit `float`, and then converted into a rational number (of sort `Real`, in Z3). The computation is not exact, because of rounding errors, and could anyway not be exact, for ω is usually irrational. However, the rounding errors are very small, and orders of magnitude even smaller than the precision of the numerical values used for physical parameters, so this is not an issue in practice. Expression ω can now be linearised using our approximation, denoted $\Omega(\delta_{\mathcal{V}}(M))$.

In fact, we have

$$\Omega(\delta_{\mathcal{V}}(M)) \lesssim \omega,$$

since each of the piece-wise affine approximation segments falls under the actual curve (ω) modulo rounding errors. This means that Ω is a conservative approximation, making it a lower bound of the actual reliability. However, the circuit's worst case is not necessarily the same among the linearised expression and the original one. Typically, the yielded lower reliability may correspond to a circuit state that (although legitimate) is other than the one which is the actual worst case.

Our objective is now to

$$\texttt{minimize}\left(\sum_{M \in \mathbf{D}_{\texttt{MOS}}} \Omega\left(\delta_{\mathcal{V}}(M)\right)\right).$$

We denote for short

$$\tilde{\mathscr{R}} = \sum_{M \in \mathbf{D}_{\texttt{MOS}}} \Omega\left(\delta_{\mathcal{V}}(M)\right).$$

We use the νZ OMT engine to solve this objective, under the constraint of preserving circuit semantics $\mathcal{S}$. The solver yields the least reliable circuit state at age t_{ref}, as well as the failure probability $\tilde{\mathscr{R}}^{\min}$ associated with it. Linearising the reliability functions is a conservative approximation (i.e., $\Omega(\delta_{\mathcal{V}}(M) \lesssim \omega)$, hence $\tilde{\mathscr{R}}^{\min}$ is a safe lower-bound for the circuit's reliability. Yet, the actual greatest lower-bound may be larger than $\tilde{\mathscr{R}}^{\min}$. Changing the way we linearise the function (e.g., how intervals I_k are chosen) may help us get closer to the greatest lower-bound. We keep the discussion on the effect of linearisation for later in Section 6.4.5, as we essentially base it on experimental findings.

6.3.3 Towards Robust Implementation of Circuit Reliability Minimization

There is one caveat when using the νZ optimizer: the solver is not to be naively trusted. Initial benchmarks have shown that, when asked to minimize reliability, νZ does not always guarantee that the resulting model actually corresponds to the lowest reliability achievable. This is likely a bug in νZ, but we could not isolate it on a non-confidential OMT problem.[3] When such is the case, the yielded reliability value remains nonetheless relatively close to the actual lower bound. We therefore choose not to fully rely on the first answer of the OMT solver. Instead, we use pure SMT reasoning (about which we are much more confident) to verify the obtained OMT solver result. Our reasoning consists on querying Z3 to tell whether it can find a circuit steady-state with a lower reliability, after using the OMT engine to get a starting steady-state. The process can continue repeatedly for multiple iterations—trying to lower circuit reliability, until it cannot be lowered further. Somehow, the initial answer makes for a good starting point to find a safe lower bound for reliability. So, after all, using OMT solving as a first step is still beneficial, even if the first answer is suboptimal.

This workaround is described, in Algorithm 2, for some circuit semantics $\mathcal{S}$ and its corresponding reliability encoding $\tilde{\mathscr{R}}$.

Algorithm 2 Search for safe lower-bound circuit reliability

```
1: function tentativeMinReliability(S, R~)
2:     opt ← Z3.optimize()                          ▷ create a new optimizer context
3:     opt.add(S)                                   ▷ assert circuit formula into the optimizer
4:     opt.minimize(R~)                             ▷ add minimization objective
5:     status ← opt.check()                         ▷ run optimizer
6:
7:     if status is SAT then
8:         R~min ← opt.get_valuation(R~)
9:     else
10:        R~min ← 0 %                              ▷ circuit S is impossible
11:    end if
12:
13:    return R~min
14: end function
15:
16: function minReliability(S, R~)
17:    R~min ← tentativeMinReliability(S, R~)
18:
19:    do
20:        ctx ← Z3.solver()                        ▷ create a new solver context
21:        status ← ctx.check(S ∧ (R~ < R~min − η))  ▷ is it possible to lower reliability more?
22:
23:        if status is SAT then
24:            R~min ← ctx.get_valuation(R~)
25:        end if
26:    while status is SAT
27:
28:    return R~min
29: end function
30:
31: return minReliability(S, R~)
```

Once an initial reliability value $\tilde{\mathscr{R}}^{\min}$ is obtained from the OMT engine, the SMT solver checks the satisfiability of $\mathcal{S} \wedge \left(\tilde{\mathscr{R}} < \tilde{\mathscr{R}}^{\min} - \eta\right)$, with η being an arbitrarily small search precision (which we set in practice at $\eta = 1 \times 10^{-5}$). SMT checks are performed over and over again as long as the

3 A recent issue posted on the Z3 repository on GitHub (`https://github.com/Z3Prover/z3/issues/3595`) reports a similar problem with optimizing objectives over the Reals with strict inequalities.

formula is satisfiable, $\tilde{\mathscr{R}}^{\min}$ being substituted each time with the lowest (i.e., the previous) value yielded. The process stops upon occurrence of an UNSAT answer. The conclusive valuation of $\tilde{\mathscr{R}}^{\min}$ is simply the most recently assigned value.

This algorithm is relatively naive and not optimized, but it is sufficient for our needs, as demonstrated experimentally. When the OMT solver yields the definite worst case, Algorithm 2 only costs one additional `check-sat` call, which is the case for the vast majority of the cases, as the OMT solver rarely misses the worst case. Experiments show that, in average, the time overhead due to additional SMT solving iterations is of 0.45 s for $\mathcal{S}^q$, −0.0009 s for $\mathcal{S}^t$ (which is a speedup), and 0.03 s for $\mathcal{S}^s$.

Function call `get_valuation(`$\tilde{\mathscr{R}}$`)` in Algorithm 2 gives the valuation of $\tilde{\mathscr{R}}$ in a circuit model. The valuation is either given (1) as a rational number or (2) in the form $\tilde{\mathscr{R}}^{\min} - \kappa \times \epsilon$, where ϵ is an uninterpreted term meant to be substituted with some arbitrarily small value. κ is a rational number. There is nothing to be done in case (1). In case (2), we need to substitute a value of the ϵ term, and assert circuit reliability as being less or equal to the corresponding numerical value, so that we have the exact circuit model which corresponds to such reliability. Z3 provides a `substitute()` function which enable us to substitute ϵ with a chosen value. In practice, we set $\epsilon = 1 \times 10^{-10}$.

The reason why in some circuit instances the lower bound contains an ϵ term is due to the nature of the constraints themselves in the SMT formula encoding the circuit. This happens when some of the faces of the polyhedron representing the SMT formula are excluded from the space of solutions, and that the optimum of a quantity to optimize is included in one of those faces (see illustration in Figure 6.5 for a 3-dimensional problem). In our circuit semantics, excluded faces are a representation of strict inequalities, like in $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ (defined in page 60)—where the voltage of short-circuited nets belongs to an open interval delimited by the supplies involved in the short-circuit.

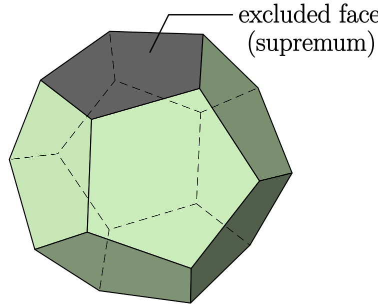


Figure 6.5: Polyhedron representing a 3-dimensional SMT problem, with an excluded face representing the optimum of some objective function.

6.4 Industrial Case Study

In this section, we experimentally evaluate the reliability analysis approach presented in Section 6.3 on a database of industrial circuits. The circuit formula $\mathcal{S}$ is added into the solver, which is then asked to minimize the conservatively approximated reliability term $\tilde{\mathscr{R}}$. Semantics $\mathcal{S}$ can be chosen among $\mathcal{S}^s$, $\mathcal{S}^t$, or $\mathcal{S}^q$. We have demonstrated that $\mathcal{S}^t$ is strictly more general than $\mathcal{S}^q$ ($\mathcal{S}^q \subseteq \mathcal{S}^t$, Theorem 4.7.6). For a given circuit, the greatest lower bound for reliability according to $\mathcal{S}^t$ must hence be at least as low as the one $\mathcal{S}^q$ results in. Intuitively, one might think $\mathcal{S}^q$ is best suited for reliability analysis, for it is more precise. It is however based on unsound approximations, unlike $\mathcal{S}^t$—which use conservative abstractions. A lower-bound of reliability given by $\mathcal{S}^t$ is hence more pessimistic but remains safe. One might also think $\mathcal{S}^q$ based encodings are harder to solve. That was indeed the case for using SMT to identify electrical errors (as we have seen in Chapter 5), but it is not exactly true when it comes to using OMT. This may seem counter-intuitive, but the precision-versus-performance tradeoffs drawn in Chapter 5 do not apply in the confines of this chapter. As the encoding of $\mathcal{S}^s$ is simpler (the constraints only consist of inequalities between semantics variables), it is tempting to say it would scale better than $\mathcal{S}^t$. We will also verify this through experiments.

Still, $\mathcal{S}^t$ has nice properties by simply taking into account the threshold voltage—which $\mathcal{S}^s$ does not.

The goal of this section is to check the validity of all these hypotheses (summarized in Table 6.2), empirically. We examine the results obtained from each of circuit semantics choices, and we compare them against the baseline approach and against SPICE simulations.

Observation	Hypothesis
$\mathcal{S}^s$ encoding is simpler than $\mathcal{S}^t$	H1: $\mathcal{S}^s$ is faster than $\mathcal{S}^t$
$\mathcal{S}^t$ encoding is simpler then $\mathcal{S}^q$	H2: $\mathcal{S}^t$ is faster than $\mathcal{S}^q$
$\mathcal{S}^s$ omits transistors threshold	H3: $\mathcal{S}^t$ is substantially more realistic than $\mathcal{S}^s$
$\mathcal{S}^q \subseteq \mathcal{S}^t$ (Theorem 4.7.6)	H4: $\mathcal{S}^q$ is substantially more realistic than $\mathcal{S}^t$
SPICE analyses are enumerative	H5: SPICE is slower than $\mathcal{S}^q$
SPICE uses actual device characteristics	H6: SPICE is substantially more realistic than $\mathcal{S}^q$

Table 6.2: Hypotheses regarding circuit semantics use for worst case reliability analysis.

6.4.1 Experimental Setup

This section describes the composition of the benchmarks, and presents parameter and encoding choices made. It discusses scaling observations upfront, and examines some implementation details.

Experiments are run on a machine with 32 GB system memory, using a single core clocked at 3.2 GHz. We use a 120 s solver timeout (after which the analysis is aborted) for each run analyzing a circuit.

6.4.1.1 The Circuits

We run our analysis on a set of 197 circuits, among which 156 ($\sim 79.2\,\%$) are multi-supply, with supply values ranging from 1 V to 3.42 V. They are used as part of a 10-bit analog-to-digital converter (ADC). Given the industrial context of this thesis, these benchmarks are not open to the public. Circuit sizes in our benchmark range from 2 to 1303 transistors. Table 6.3 presents an overview of the composition of circuits in the case study, grouped by intervals based on their size. Our analysis computes the minimal reliability of each circuit while complying with its semantics.

$\lvert\mathbf{D}_{\mathtt{MOS}}\rvert$	count
$[1, 10[$	80
$[10, 100[$	101
$[100, 1000[$	14
1000+	2

Table 6.3: Benchmark composition.

6.4.1.2 Reliability Parameters and Encoding Choices

The analysis is performed for an age reference $t_{\text{ref}} = 5\,\text{yr}$, and for temperature $T = 323.15\,\text{K}$. The apparent activation energy is set to $E_a = 0.5\,\text{eV}$, and the voltage acceleration parameter is set to $\gamma = 1$, which can be associated to an electrical oxide thickness of 10 nm [Vollertsen and Wu, 2004]. Shape parameter is set to $\beta = 1.1$, which reflects the fact that the failure rate increases as time goes by. We use a TTF scale factor $A_0 = 10$.

When it comes to circuit semantics encoding, we choose to relax constraints on inputs: we use $\mathcal{R}_{\mathbf{I}}$ relaxed and not $\mathcal{R}_{\mathbf{I}}$ strict.

As we deal with multi-supply designs, the approximation function being exact only applies under specific conditions, notably in the absence of voltage drops due to short-circuits and floating nets. Even for other parts of the circuit, since reliability decreases exponentially with voltage, the worst-case is usually reached when many transistors have their maximal voltage applied while few others

are applied a minimal voltage, rather than all transistors being applied a medium voltage. This means that worst-case states tend to use the points of the curve where the linearized curve matches the actual curve, which is a result of using the adaptive strategy for linearisation.

6.4.1.3 On Incremental Solving

We previously explained in the last paragraph of Section 6.3.3 that after running the analysis using the Z3 solver, when the lower-bound on reliability is not a reachable minimum (e.g., minimizing a value over an open interval), the value may be expressed as $\tilde{\mathscr{R}}^{\min} - \kappa \times \epsilon$ where $\tilde{\mathscr{R}}^{\min}$ is the lower bound, and κ a rational number. This gives a sound (yet not necessarily attainable) reliability value, but not the worst case state (i.e., voltage values for each net), since no model reaches reliability $\tilde{\mathscr{R}}^{\min}$. In such case, we add $\tilde{\mathscr{R}} \leq \tilde{\mathscr{R}}^{\min} - \kappa \times \epsilon$ with a very small ϵ (in our case, we take $\epsilon = 1 \times 10^{-10}$) as an additional constraint for the solver. We then ask a second time the solver (as a satisfiability query, without objective function) to provide a model. We use the incremental feature of Z3 (**`push/pop`** operations) to do so. This way, the solver does not restart all over again from scratch. It instead adds the additional constraint starting from a point where the circuit formula is already in the solver's context. Note that benchmarks show that using the incremental feature of Z3 is not necessarily faster than launching a second independent solver; the former is only faster than the latter in 57 % of the cases, for $\mathcal{S}^s$, when substitution of epsilon terms is involved. Figure 6.6 compares the two approaches with respect to Z3 solving time.

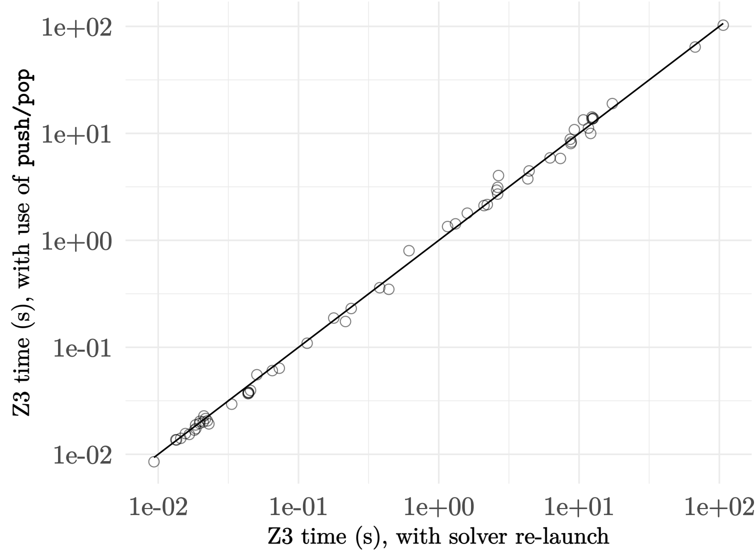


Figure 6.6: Comparison of Z3 solving time based on the use of the solver's incremental feature, with $\mathcal{S}^s$.

6.4.2 Experimental Evaluation

We run experiments using the experimental setup presented in the previous section, and we compare obtained results with all three circuit semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$. A total of 175 circuits (out of 197) are actually analyzed with $\mathcal{S}^q$ (the solver times-out for the remaining circuits), while only 160 circuits are analyzed with each semantics $\mathcal{S}^s$ and $\mathcal{S}^t$. Large circuits are, in general, less likely to be analyzed within the time limit. In fact, circuits that could not be analyzed within 120 s could also not be analyzed within a much larger time limit. We also observe that, for circuits ranging from 10 to 100 transistors, analysis might or might not terminate (Figure 6.7). For larger circuits, timeout is quasi-systematic.

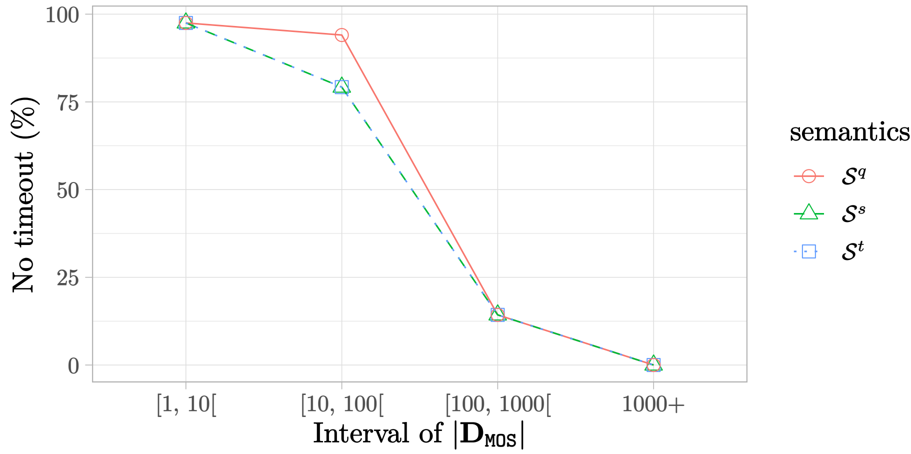
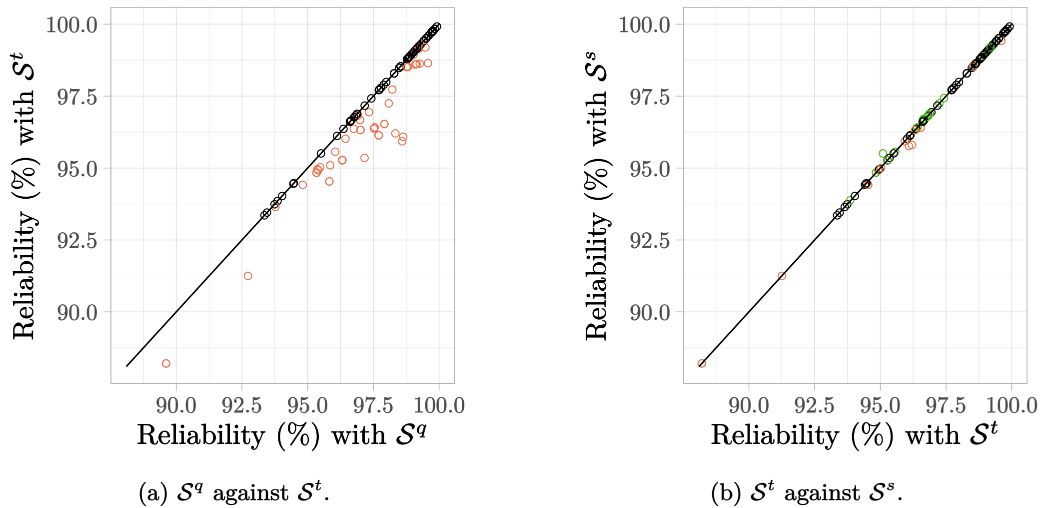


Figure 6.7: Termination of Z3 solving with respect to $|\mathbf{D}_{\texttt{MOS}}|$.

Figure 6.8 plots obtained results, comparing $\mathcal{S}^q$ against $\mathcal{S}^t$, and $\mathcal{S}^t$ against $\mathcal{S}^s$. Indeed, $\mathcal{S}^t$ always yields a reliability lower-bound that is at least as low as the one obtained with $\mathcal{S}^q$ (Figure 6.8a), which is coherent with $\mathcal{S}^q \subseteq \mathcal{S}^t$ (Theorem 4.7.6). In the majority of cases (61.6 %), values of the lower bound coincide among $\mathcal{S}^t$ and $\mathcal{S}^q$ (the cases which appear on the diagonal). In the remaining 38.4 % of the cases, the lower-bound obtained with $\mathcal{S}^t$ is 1 %-close to the one obtained with $\mathcal{S}^q$, i.e., in 62.3 % of the cases $\tilde{\mathscr{R}}^{\min}(\mathcal{S}^q) - 1\,\% \leq \tilde{\mathscr{R}}^{\min}(\mathcal{S}^t) < \tilde{\mathscr{R}}^{\min}(\mathcal{S}^q)$, where $\tilde{\mathscr{R}}^{\min}(\mathcal{S}^t)$ denotes the lower-bound obtained with $\mathcal{S}^t$, and $\tilde{\mathscr{R}}^{\min}(\mathcal{S}^q)$ denotes the lower-bound obtained with $\mathcal{S}^q$. The maximum difference observed between $\tilde{\mathscr{R}}^{\min}(\mathcal{S}^q)$ and $\tilde{\mathscr{R}}^{\min}(\mathcal{S}^t)$ is 2.67 %. On the basis of these results, Hypothesis H4 (Table 6.2) can neither be validated nor fully refuted.

As there is no inclusion relation between $\mathcal{S}^s$ and $\mathcal{S}^t$ (Theorems 4.7.1 and 4.7.2), no order is expected between their respective reliability lower-bound values yielded, and this is confirmed by Figure 6.8b. It shows that the values obtained with the two semantics remain very close to each other, the maximum difference $|\ \tilde{\mathscr{R}}^{\min}(\mathcal{S}^t) - \tilde{\mathscr{R}}^{\min}(\mathcal{S}^s)\ |$ being only 0.064 %. Hypothesis H3 (Table 6.2) is refuted. This is a useful observation in deciding which semantics to adopt: if $\mathcal{S}^s$ turns out to be much faster, it may justify using it over $\mathcal{S}^t$ while possibly sacrificing marginal precision.

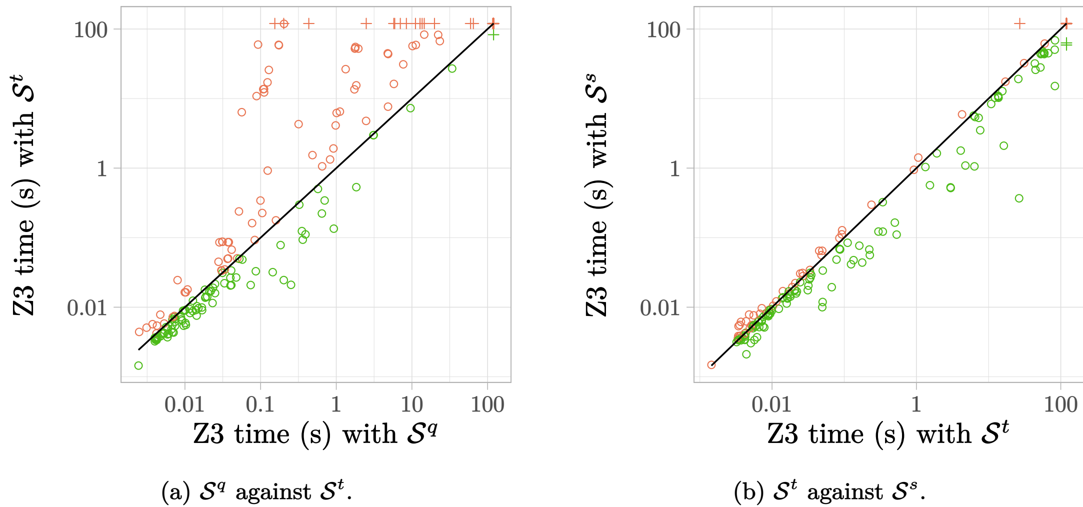


(a) $\mathcal{S}^q$ against $\mathcal{S}^t$.

(b) $\mathcal{S}^t$ against $\mathcal{S}^s$.

Figure 6.8: Comparison of reliability lower-bound at age $t_{\text{ref}} = 5\,\text{yr}$, obtained based on circuit semantics used.

Solving performance results are shown in Figure 6.9. $\mathcal{S}^q$ outperforms $\mathcal{S}^t$ in 42.77 % of the cases, when both semantics terminate within the 120 s time limit. For many circuits, time overhead related to $\mathcal{S}^t$ is more important the one related to $\mathcal{S}^q$ (Figure 6.9a). Also, $\mathcal{S}^t$ times-out much often compared with $\mathcal{S}^q$. There are 37 timeouts with $\mathcal{S}^t$, while $\mathcal{S}^q$ only times-out for 22 cases. These results refute our initial hypothesis that $\mathcal{S}^t$ is faster than $\mathcal{S}^q$ (Hypothesis H2, Table 6.2). When it comes to $\mathcal{S}^s$ against $\mathcal{S}^t$, $\mathcal{S}^s$ is indeed globally significantly faster, considering that the results are plotted on a log-scale (Figure 6.9b). This validates Hypothesis H1 (Table 6.2). Both semantics $\mathcal{S}^s$ and $\mathcal{S}^t$ time-out for 37 cases. Note that the + mark at the upper-right corner of Figure 6.9b corresponds to multiple circuits for which both semantics did not terminate within the 120 s time limit. These are, more or less, the same circuits for which $\mathcal{S}^t$ times-out in Figure 6.9a (the many horizontal + marks at the top of the figure).

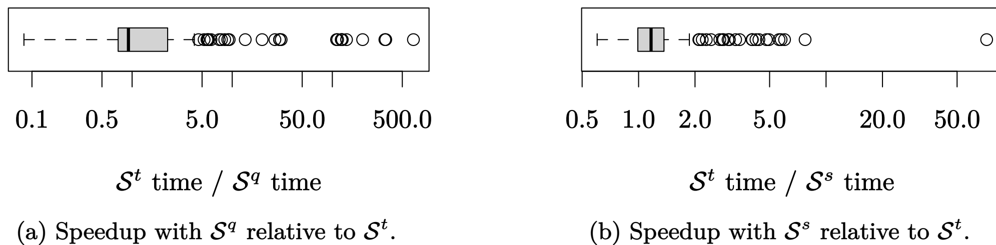


(a) $\mathcal{S}^q$ against $\mathcal{S}^t$.

(b) $\mathcal{S}^t$ against $\mathcal{S}^s$.

Figure 6.9: Comparison of the solving time for minimizing reliability based on the circuit semantics used. The + marks mean that the 120 s time limit is reached with at least one of the two semantics and the solver did not terminate.

Semantics relative speedups are summarized, partially, in Figure 6.10, by excluding timeouts. In average, $\mathcal{S}^q$ is 16.74 times faster than $\mathcal{S}^t$ (Figure 6.10a), with a median at 0.92. $\mathcal{S}^s$ is 1.95 times faster than $\mathcal{S}^t$ (Figure 6.10b), with a median at 1.17.



(a) Speedup with $\mathcal{S}^q$ relative to $\mathcal{S}^t$.

(b) Speedup with $\mathcal{S}^s$ relative to $\mathcal{S}^t$.

Figure 6.10: Comparing speedups among semantics $\mathcal{S}^q$ vs. $\mathcal{S}^t$, and $\mathcal{S}^s$ vs. $\mathcal{S}^t$.

6.4.3 Performance Versus Precision Tradeoffs

From Figures 6.8b and 6.9b, it is suggested that one should use $\mathcal{S}^s$ over than $\mathcal{S}^t$, for $\mathcal{S}^s$ tends to terminate faster while remaining very close to $\mathcal{S}^t$. Still, only $\mathcal{S}^t$ guarantees conservative modeling, and therefore guarantees the lower-bound of reliability is safe. Figure 6.8a suggests that $\mathcal{S}^q$ is globally faster than $\mathcal{S}^t$, in addition to exceeding the time limit less often. At this stage, comparing $\mathcal{S}^q$ against $\mathcal{S}^s$ makes more sense. We show these comparisons in Figure 6.11 for reliability, and Figure 6.12 for the solving time.

$\mathcal{S}^q$'s speedup relative to $\mathcal{S}^s$ is summarized in Figure 6.13, by excluding timeouts. In average, $\mathcal{S}^q$ is (surprisingly) 13.76 times faster than $\mathcal{S}^s$, with a speedup median at 0.82. Such speedup is very likely due to the fact that $\mathcal{S}^q$ is deterministic when it comes to nets' voltage valuations, which helps the solver converge faster.

We may tolerate using $\mathcal{S}^q$ over $\mathcal{S}^s$ (both being non-conservative) only if the approximations in $\mathcal{S}^q$ are not too unrealistic. Otherwise, using $\mathcal{S}^s$ is reasonable. We verify this, in the next section, with respect to simulation.

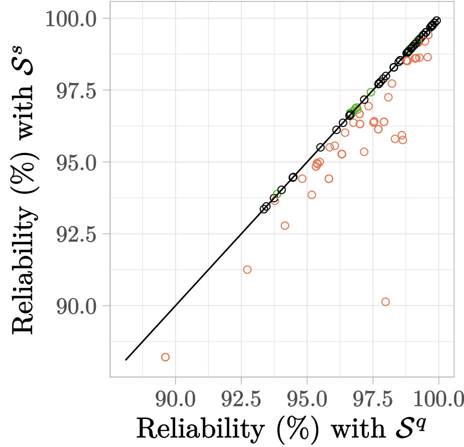


Figure 6.11: Comparison of reliability lower-bound at $t_{\text{ref}} = 5\,\text{yr}$, obtained with $\mathcal{S}^q$ against $\mathcal{S}^s$.

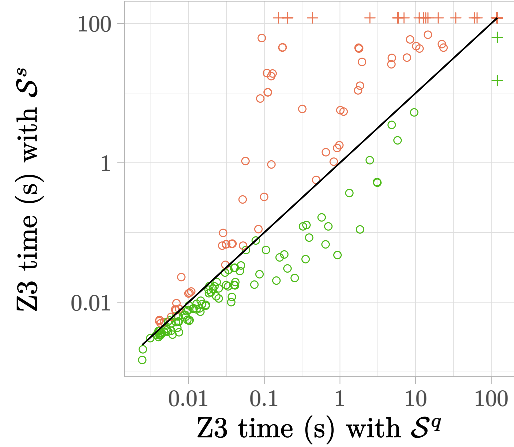


Figure 6.12: Comparison of the solving time using $\mathcal{S}^q$ against $\mathcal{S}^s$.

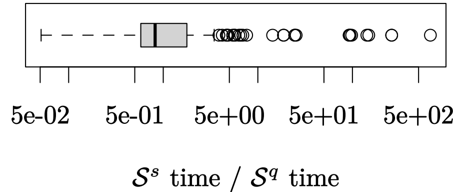


Figure 6.13: Speedup with $\mathcal{S}^q$ relative to $\mathcal{S}^s$.

6.4.4 Experimental Validation of Circuit Semantics Against SPICE

We conduct SPICE simulation campaigns on the same benchmarks of Section 6.4.1 to identify, for each circuit, the greatest lower bound for reliability and associated circuit state (in terms of nets voltages, determined by $\mathcal{V}$). We use the PyLTSpice library in Python [Brum, 2024] to automate simulations on top of the LTspice program [Analog Devices, 2024], using Level 1 SPICE model [Patel, 2014]. SPICE is a state-of-the-art tool, widely trusted to be sound, hence our using it as a reference. The goal of using simulation, in this context, is to identify circuit steady-states, yet SPICE provides much more advanced analyses. Using it as a standalone tool for identifying steady-states at production level would be exorbitant.

For simulations, the threshold voltage is set to $V_t = 0.2\,\text{V}$, so that we keep the same assumptions on transistor activation as in semantics $\mathcal{S}^t$ and $\mathcal{S}^q$. Inputs are constrained using an equivalent of $\mathcal{R}_\mathbf{I}$ strict, i.e., input voltage values are chosen among available supply voltages. Otherwise, it would be virtually impossible for simulation to consider continuous intervals of input vectors. The best thing that could be done is to consider sampled voltage values in a given interval, using some *acceptable* sampling step, but that comes with combinational explosion in terms of input vectors, and, thus, in terms of simulations to run. All of that would require additional engineering work

for marginal benefits, so we choose not to proceed in that direction. Constraining inputs as done in $\mathcal{R}_\mathbf{I}$ strict is enough for assessing the precision of our semantics-based analyses, and comparing their performances against SPICE's.

Our simulation-based analysis flow is depicted in Figure 6.14. From a circuit description and accompanying power information, power scenarios are enumerated. Each scenario defines constant supply and input voltages, used to run a single simulation of the circuit. Upon simulation completion, net voltages are retrieved from the simulation trace (which represents a circuit steady-state in our case). We compute the circuit's reliability for all traces, using the same piecewise affine approximation of the logarithm of devices reliabilities (Figure 6.4). This choice makes it possible to better evaluate the impact of using circuit semantics ($\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$) as opposed to the detailed model used by SPICE, without changing other parts of the analysis (for a fair comparison).

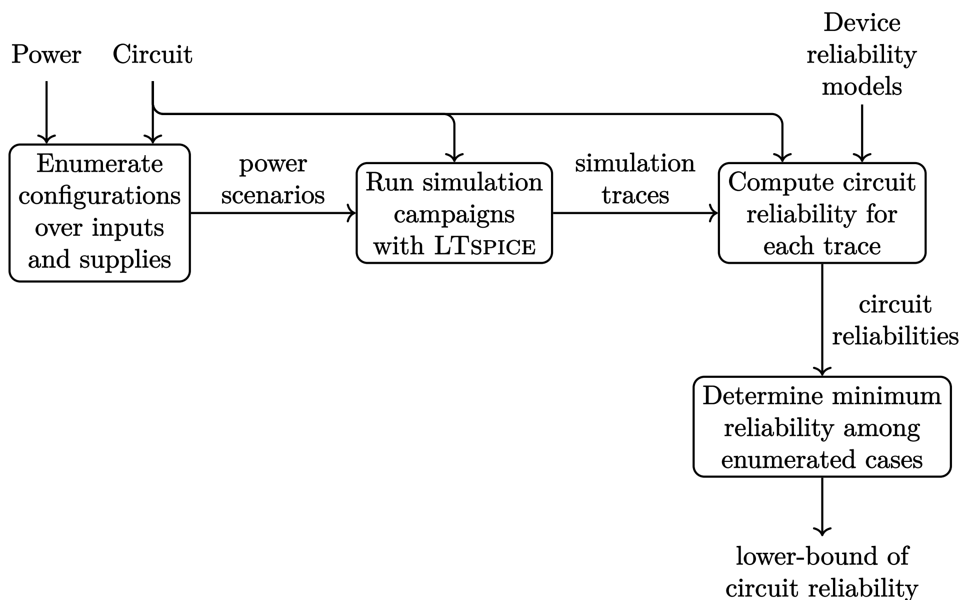


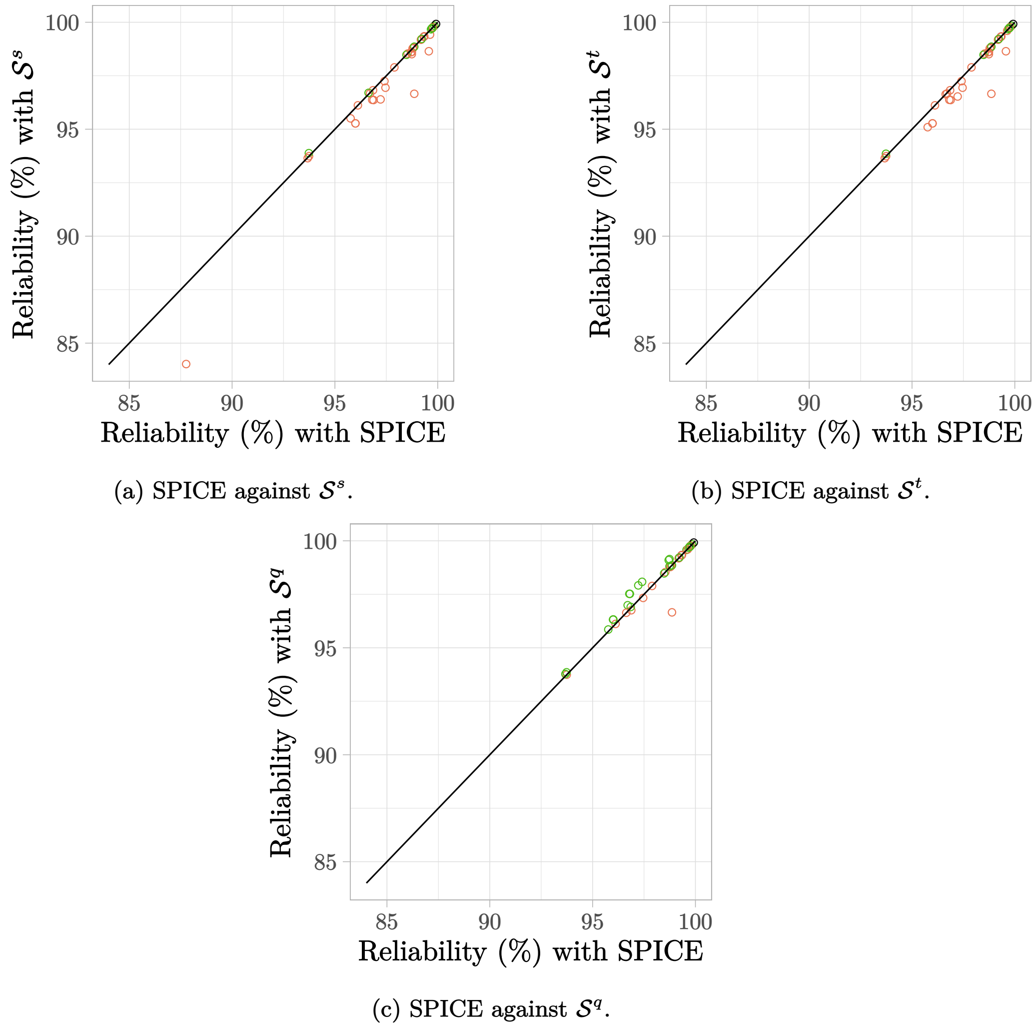
Figure 6.14: SPICE-based approach for identifying a lower-bound on circuit reliability and the corresponding circuit state.

Figure 6.15 compares obtained lower bound circuit reliabilities with SPICE simulations against circuit semantics. Dots on the diagonals correspond to cases where a semantics-based approach (with OMT solving) is either as precise as simulation, or is very close to simulation. Globally, semantics $\mathcal{S}^s$ and $\mathcal{S}^t$ result in reliability values that are at least as low as the ones obtained from simulation (Figures 6.15a and 6.15b). In some cases, the failure probability computed with SPICE is slightly lower (by $1 \times 10^{-5}\,\%$ at most) than the one obtained from our approach with $\mathcal{S}^s$ or $\mathcal{S}^t$. This is due to simulation precision, e.g., a supply voltage may be slightly above or below its ideal value in simulation, resulting thus in a lower reliability on some transistor, so a lower reliability on the whole circuit.

There is one case when such rounding errors result in overestimating the reliability lower bound with SPICE by more than 0.13 %. The concerned circuit contains two groups of large numbers of devices used in a parallel setting—i.e., having the same gate, source, and drain nets. They represent more than two thirds of the circuit. By performing a parallel reduction over these devices (i.e., only keeping one device for each set of devices sharing the same terminals), and re-running both SPICE simulations and $\mathcal{S}^t$ analyses, we observe that SPICE-based estimation of the lower bound is now 0.01 %-greater than $\mathcal{S}^t$. The exact reason for the discrepancy on the original circuit is still under investigation.

$\mathcal{S}^q$ on the other hand tends to result in relatively slightly higher reliability values (Figure 6.15c), with at most 2.4 % in 48.33 % of the cases. This refutes Hypothesis H6 (Table 6.2).

Knowing that (1) both $\mathcal{S}^q$ (from the way device I-V relations are abstracted, see Figure 4.29, page 68) and $\mathcal{S}^s$ (which does not take account of the threshold voltage) are not sound, and that (2) $\mathcal{S}^q$ is globally faster than $\mathcal{S}^s$, and (3) having the empirical results of Figure 6.15 (the results obtained with $\mathcal{S}^t$ being close to $\mathcal{S}^s$), we have enough reasons to discard the idea of using $\mathcal{S}^s$ in the context of reliability analysis.



(a) SPICE against $\mathcal{S}^s$.

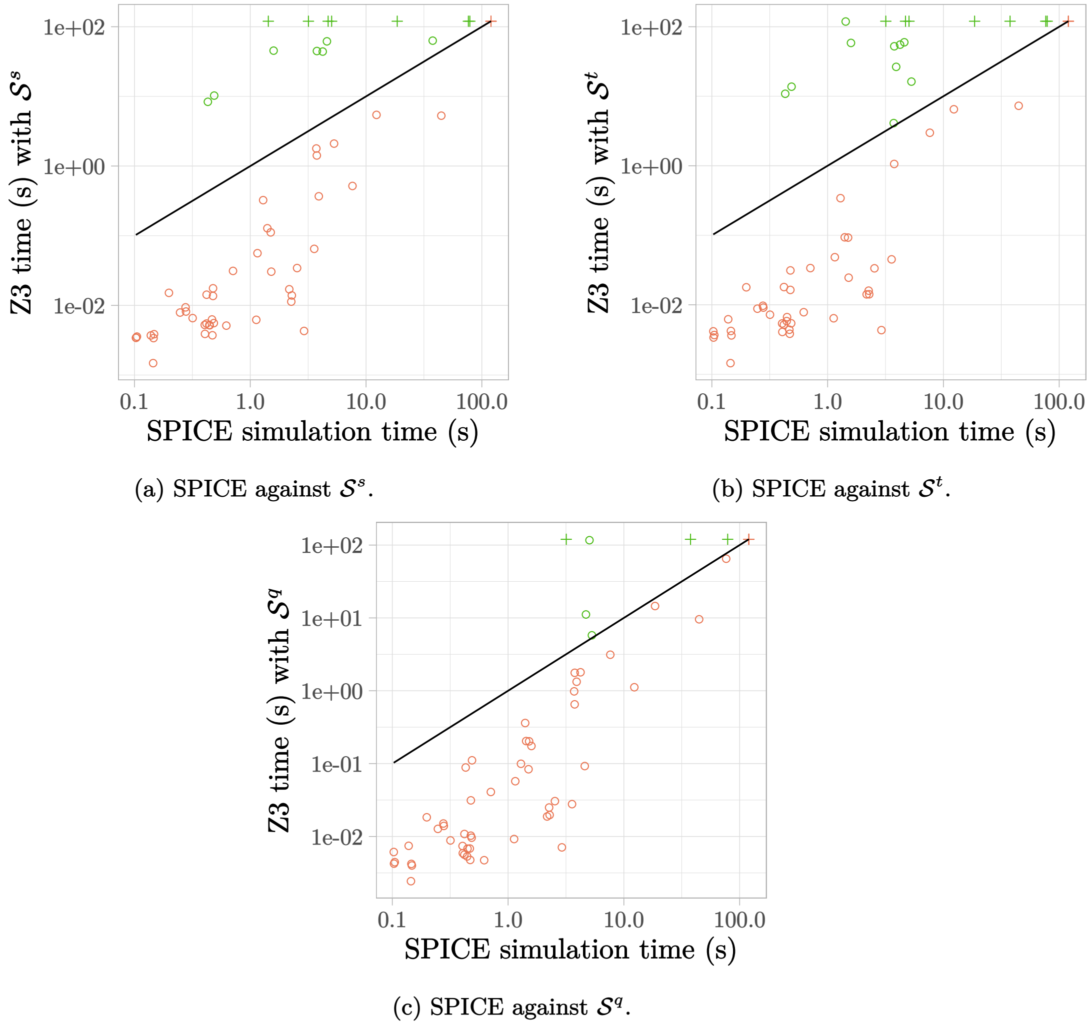
(b) SPICE against $\mathcal{S}^t$.

(c) SPICE against $\mathcal{S}^q$.

Figure 6.15: Comparison of reliability lower bound at age $t_{\text{ref}} = 5\,\text{yr}$, obtained from SPICE simulations against circuit semantics, excluding timeouts.

As previously mentioned, while simulation can be more accurate in terms of reliability estimation given an input vector, it becomes intractable if no input vector is provided. Conversely, as our OMT-based approach symbolically explores a continuous space of input values, we can consider every possible set of inputs for a given circuit (by using $\mathcal{R}_{\mathbf{I}}$ relaxed). This would be impossible with a simulation-based approach, as it can only enumerate discrete input values.

Total simulation time is compared with the solving time from semantics-based OMT analyses. Comparisons are shown in Figure 6.16. In the vast majority of cases, especially with $\mathcal{S}^q$ (Figure 6.16c), SPICE-based analysis is slower than our semantics-based approach (which validates Hypothesis H5, Table 6.2). It is yet worth mentioning that there are a few cases when SPICE is faster, namely when the number of circuit configurations to simulate is small. This is true for more cases with semantics $\mathcal{S}^s$ (Figure 6.16a) and $\mathcal{S}^t$ (Figure 6.16b).



(a) SPICE against $\mathcal{S}^s$.

(b) SPICE against $\mathcal{S}^t$.

(c) SPICE against $\mathcal{S}^q$.

Figure 6.16: Comparison of the simulation time for finding the lower bound for circuit reliability (at age $t_{\text{ref}} = 5\,\text{yr}$) against the solving time with circuit semantics. The + marks mean that the 120 s time limit is exceeded by the total simulation, or achieved with circuit semantics and the solver did not terminate.

From these experimental results, and from the ones of the previous section, it is suggested that $\mathcal{S}^q$ represents a good tradeoff between performance and precision for the specific use case considered in this chapter: finding the greatest lower bound for circuit reliability.

6.4.5 Impact of the Linearisation Strategy for the Reliability Function

Back in Section 6.3.1, we decided that the adaptive approach is the best strategy for linearising reliability functions. Here, we present the justification—based on experimental results—of such choice. We launch benchmarks to assess the effect of the choice of the piecewise affine approximation of reliability (Figure 6.4, page 102) on both the solving time and gain in terms of the worst-case reliability estimation. In this regard, we only consider semantics $\mathcal{S}^s$. Figure 6.17 shows a two-by-two comparison of the time overhead caused by doubling the number N of intervals I_k of the piecewise affine approximation, for non-negative voltage stress values. As a reminder, the reliability associated with any negative value of the voltage stress is approximated with the evaluation of (the logarithm of) reliability at 0 V.

Benchmarks are run for evenly-divided intervals with N = 1, 2, 4, and 8, as well as for the adaptive approach (which is based on adapting the choice of intervals based on the available supply values on each sub-circuit) and 'adaptive+' (which consists in dividing into two equal parts each of the intervals obtained with the adaptive version).

As experimental results show, increasing the number of intervals indeed slows down the solving time in the vast majority of cases. Also, they show that the adaptive approach is globally much faster than N=8. It is even significantly faster than N=4.

Figure 6.18 shows that the gain in terms of increasing the lower-bound value for reliability of the worst-case circuit state is mostly noticeable with N=2 compared to N=1. The gain using N=4 or N=8 is small, for a substantial additional performance cost. The adaptive version is both more precise than N=8 and faster that N=4, and hence appears to be a good compromise. The adaptive+ version is marginally better than the adaptive approach, showing that there is only little to gain, and the performance cost is too high compared to the benefit. Therefore, it justifies the choice we made to use the adaptive strategy throughout this chapter. By using knowledge about the circuit's supplies, we achieve good performance and precision, and do not have to use more general but heavier linearisation techniques (e.g., [Cimatti et al., 2018]).

6.4.6 Comparison of Our Approach Against the Baseline

In order to assess the contributions of this work, we compare these results against estimations of circuits' reliabilities obtained from running a baseline approach that uses no circuit semantics, as proposed in Section 6.2.5. The baseline approach which we consider consists in assuming that the transistors of a circuit are all simultaneously used to their full potential—i.e., to each transistor is applied the maximum stress voltage ($\mathcal{V}_{\text{GS}}$ or $\mathcal{V}_{\text{GD}}$) based on the supplies it can statically reach, while considering transistors as isolated components. Indeed, such assumption is overly *pessimistic*, as the corresponding nets' valuations highly unlikely match a feasible circuit state. Therefore, we consider that with such approach we obtain a lower bound of a circuit's reliability for our semantics-based approach. Accordingly, using our semantics-based approach we can increase such lower bound, which leads to getting a larger reliability estimation (which is a safe estimation when $\mathcal{S}^t$ is used). We define the gain, for each analyzed circuit, as a ratio: "the failure rate at t_{ref} obtained from the baseline method" over "the failure rate at t_{ref} obtained from a semantics-based approach". Such ratio is at least equal to 1 (the gain is equal to 1 when both approaches yield the same reliability).

Figure 6.19 shows the gains obtained with each semantics against the baseline. For about two thirds of the analyzed circuits, the gain is greater than 1, with all three semantics $\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$. The further (to the right) the dots are from the diagonal, the more pessimistic the reliability estimation using the baseline approach with respect to the considered circuit semantics is.

The performance of our tool using the adaptive approach for linearisation is illustrated in Figure 6.20, for $\mathcal{S}^s$, which shows that most circuits are analyzed in less than 10 seconds. Circuits containing less than 10 transistors are analyzed almost instantly (<0.1 s). In complement, Figure 6.21 shows, for $\mathcal{S}^s$, the gain over the baseline approach as a function of the solving time. Unsurprisingly, circuits for which no gain is obtained are simple, hence fast to analyze.

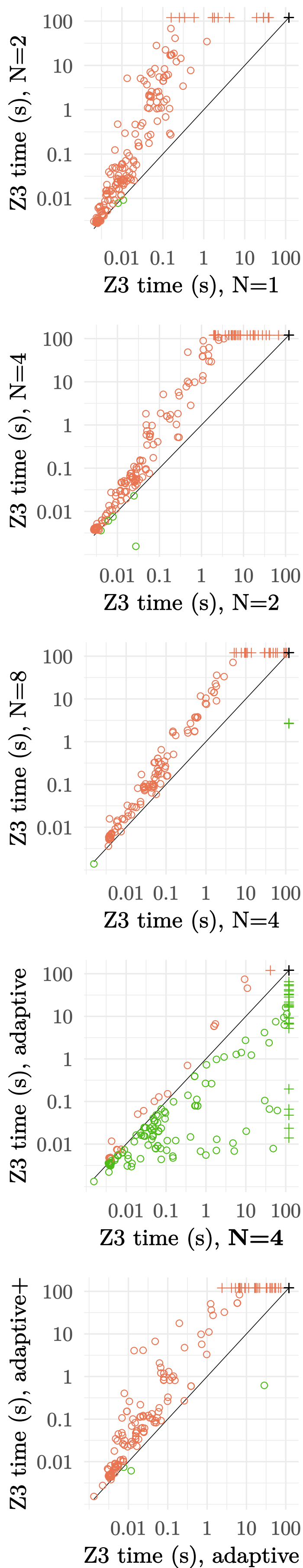


Figure 6.17: Two-by-two comparison of Z3 solving time, using $\mathcal{S}^s$.

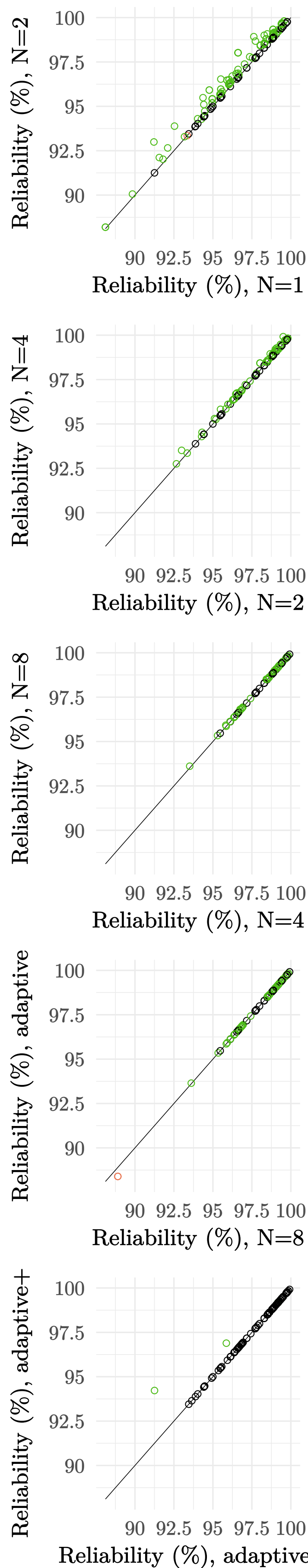
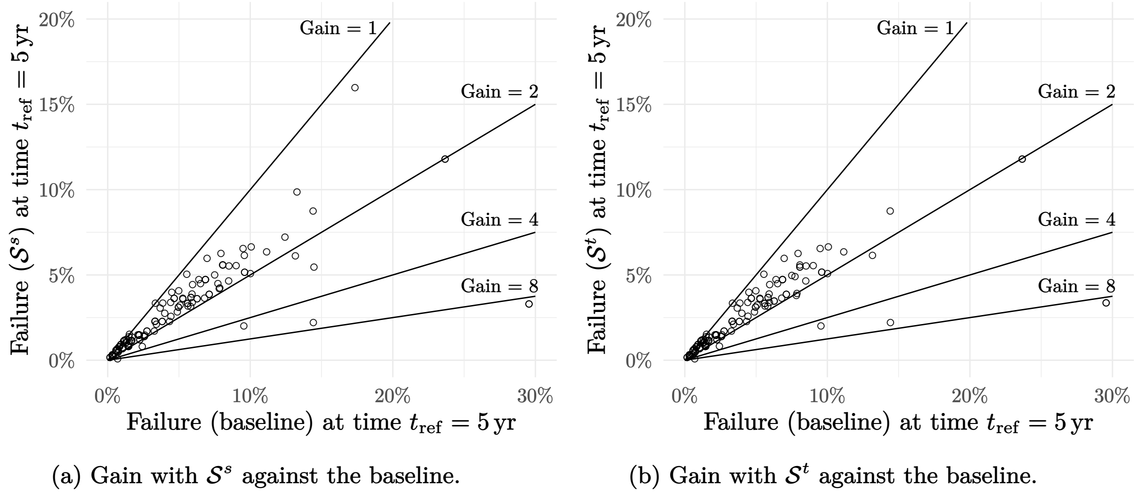
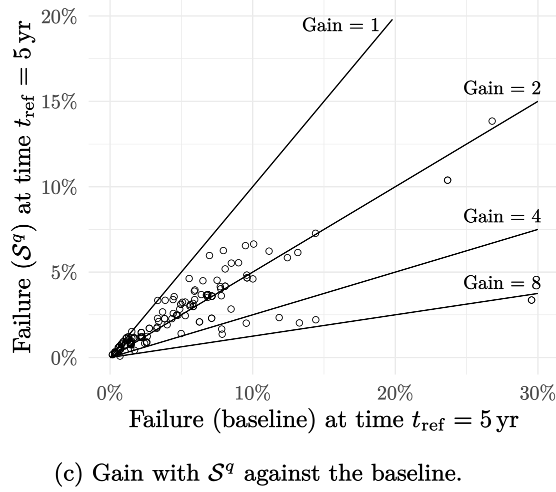


Figure 6.18: Two-by-two comparison of reliability, using $\mathcal{S}^s$.



(a) Gain with $\mathcal{S}^s$ against the baseline.

(b) Gain with $\mathcal{S}^t$ against the baseline.



(c) Gain with $\mathcal{S}^q$ against the baseline.

Figure 6.19: Gain obtained using semantics-based analysis over the baseline.

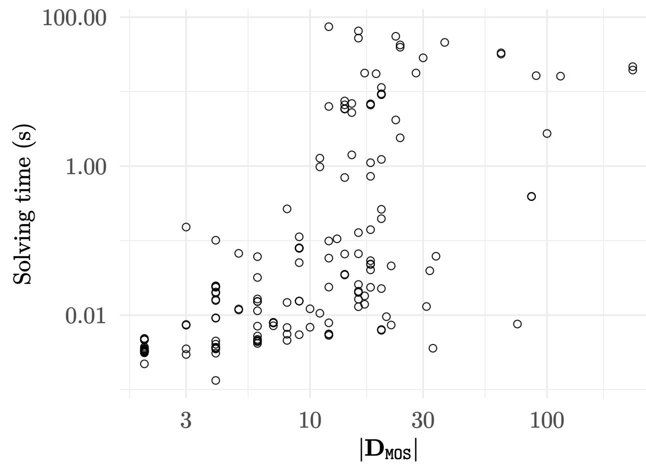


Figure 6.20: Solver performance for $\mathcal{S}^s$, with respect to circuits' size.

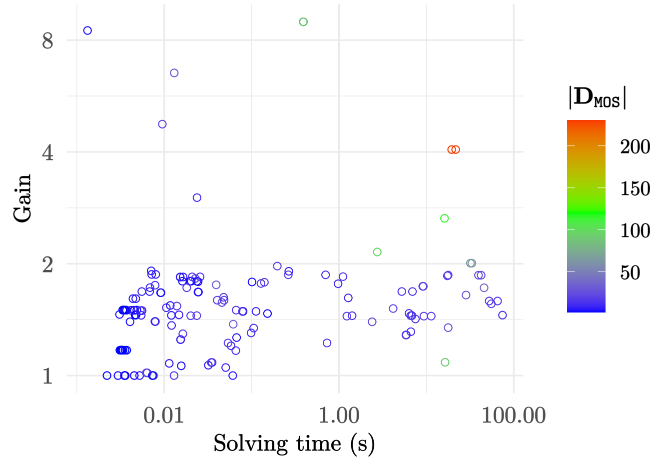


Figure 6.21: Gain using $\mathcal{S}^s$ over the baseline approach, with respect to the solving time.

6.4.7 Alternate Use Case: Minimizing Circuit Lifetime

We now consider the problem of the search for the age t_{ref} at which a circuit's reliability attains a specific value $\mathscr{R}_{\text{ref}}$ in the worst-case, considering some search precision η. We run experiments on the same benchmarks used in Section 6.4.1, with $\mathscr{R}_{\text{ref}} = 50\,\%$ and $\eta = 1\,\%$.

We use simple heuristics to choose the age at which we start the binary search, i.e., finding an upper-bound of the value of t_{ref}. Our heuristics are based on the fact that in most circuits, especially in CMOS-based ones, around half of the devices are fully activated in each steady-state. Accordingly, we consider that half of the devices are activated, and that the voltage applied on each device is maximal based on the reachable supplies. We then initialize our search by computing numerically $t_0 = \omega^{-1}(\mathscr{R}_{\text{ref}})$, and compute the worst-case reliability using our OMT-based method for $t = t_0$. If needed we multiply t_0 by 10 until reaching an upper-bound for t_{ref}, and then perform a binary search on the interval $[0, t_0]$. In total, 147 circuits are analyzed out of 197 within the set 120 s time-limit.

As each iteration of the binary search consists in minimizing reliability while satisfying the circuit's semantics, the solving time is almost always the same for each circuit, as shown in Figure 6.22. The total solving time is, therefore, linear with respect to the number of binary search steps needed for the analysis to conclude.

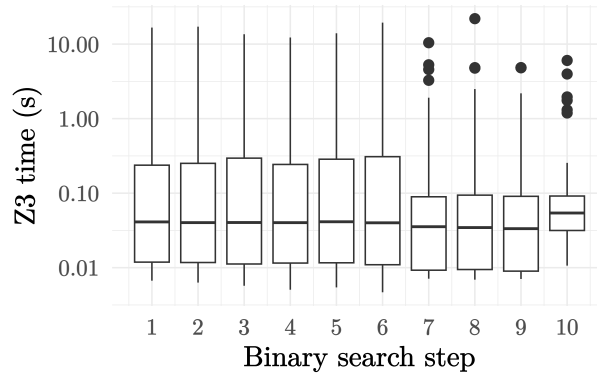


Figure 6.22: Average solving time for binary search with semantics $\mathcal{S}^s$.

6.5 Conclusion and Future Work

In this chapter, we presented a novel utilization of circuit semantics introduced in Chapter 4. We demonstrated that such modelings can be used not only to identify electrical errors in circuits, but also in the context of reliability analysis, considering the TDDB model. By enhancing the SMT-based approach with an OMT engine, it is possible to explore the space of steady-states of a circuit and exhibit the worst case steady-state with respect to reliability.

On an industrial use case composed of 197 sub-circuits, we assessed usability of the different semantics, and compared them against a simulation-based approach and against the baseline approach. Initially stated hypotheses are then answered on the basis of experimental evaluation. The conclusions drawn from these experimental results, regarding various hypotheses, are summarized in Table 6.4.

Initial hypothesis	Verdict	Evidence
H1: $\mathcal{S}^s$ is faster than $\mathcal{S}^t$	validated	see experimental results in Figure 6.9b
H2: $\mathcal{S}^t$ is faster than $\mathcal{S}^q$	refuted	$\mathcal{S}^t$ is globally slower (Figure 6.9a)
H3: $\mathcal{S}^t$ is substantially more realistic than $\mathcal{S}^s$	refuted	see experimental results in Figure 6.8b
H4: $\mathcal{S}^q$ is substantially more realistic than $\mathcal{S}^t$	sometimes	true in 38.4 % of the cases (Figure 6.8a)
H5: SPICE is slower than $\mathcal{S}^q$	validated	see experimental results in Figure 6.16c
H6: SPICE is substantially more realistic than $\mathcal{S}^q$	refuted	see experimental results in Figure 6.15c

Table 6.4: Summary conclusions on the hypotheses initially stated in Table 6.2.

The results on the runtime clearly discard our approach for monolithic circuit-wide analysis. However, scaling to arbitrary-size circuit can always be done with a complexity of $O(\text{circuit size})$, using two simple observations. First, once a worst-case reliability is computed for each sub-circuit, composing them pessimistically for an assembly of sub-circuits is easy. When reliability functions are exponential laws, this is the sum of failure rates (SOFR) model [Srinivasan et al., 2004], commonly used in the industry. Because we use the more accurate Weibull distribution (with $\beta \neq 1$), we cannot use SOFR, but we can instead compute the product of worst-case reliabilities of sub-circuits, because we consider only the first failure occurring among the sub-circuits. The second observation is that, when our analysis times-out, we can fall back to the baseline approach, which is always fast although more pessimistic, for the problematic sub-circuit. Static voltage propagation algorithms are very efficient, making it possible to handle very large designs.

Further experiments regarding the impact of the choice of intervals of linearisation of the reliability function were conducted. They have shown that the adaptive linearisation approach is the best strategy performance wise. We have demonstrated how our OMT-based approach can be used to improve the accuracy of reliability analysis, with respect to approaches that do not consider circuit semantics (i.e., the baseline approach). We also demonstrated that our OMT-based estimation is close to the actual reliability computation (using SPICE as a reference). Indeed, the first source of imprecision are the circuit semantics ($\mathcal{S}^s$, $\mathcal{S}^t$, and $\mathcal{S}^q$) that were shown to have a marginal impact—in Section 6.4.2. The second source of imprecision is due to the linearisation of the reliability function, which remains safe, and for which we showed that we are close to optimal—in Section 6.4.5. This shows that the abstractions used to encode circuit semantics into SMT/OMT formulas is, in practice, precise enough to identify the worst-case steady-state of a circuit. This can be used by designers to identify the states that impact reliability the most, or to find a formal lower bound for reliability.

7

Conclusion and Future Directions

In this thesis, we discussed circuit modeling and verification aspects. It is now time to conclude this work, by discussing the contributions presented, and laying down the ground for future extensions and research topics that may derive from it. We first summarize the contributions of this thesis in Section 7.1, and present the lessons learned—which can benefit future implementers. Section 7.2 then presents future research directions.

7.1 Contributions and Lessons Learned

This thesis addresses the problem of electrical rule checking (ERC) at transistor level. While state-of-the-art techniques tend to mostly use ad-hoc approaches [Ferres et al., 2025], relying on proprietary knowledge about the intellectual properties (IPs) being designed, we investigate the usability of formal approaches based on symbolic verification for more generic analysis. The main premise is that if, by some encoding process, circuit descriptions can be transformed to logic formulas, then it is possible to check (with the help of a solver) whether they satisfy/violate some of the design's electrical properties. The approach was first introduced in our paper [Ferres et al., 2023]. What we initially hoped for is that such approach improves correctness of the analysis, without necessarily sacrificing performance. We examined this on the basis of a combination of theoretical facts and experimental evaluation, and we recall our conclusions below.

7.1.1 Circuit Semantics

This work is about defining encoding processes by which transistor level circuit descriptions are transformed to logic formulas, i.e., circuit semantics. These modelings serve as the basis for reasoning about electrical properties in circuits' steady-states. Different kinds of circuit semantics are presented in Chapter 4. Relational switch-based circuit semantics $\mathcal{S}^s$ is the most basic one—published in [Oulkaid et al., 2024], whose variant $\mathcal{S}^t$ takes into account threshold voltages for device activation, hence improving correctness of the approach. Semantics $\mathcal{S}^q$, on the other hand, is designed radically differently: it is based on approximations of actual device equations, i.e., Intensity-Voltage (I-V) relations. While each of these semantics were first defined for transistors, we later presented their generalization to resistors and diodes.

7.1.2 Electrical Error Identification

A property $\mathcal{P}$ is verified on circuit semantics $\mathcal{S}$, if there is no model which satisfies the formula $\mathcal{S} \wedge \neg\mathcal{P}$. This can be checked with satisfiability modulo theories (SMT) solvers, like Z3 [De Moura and Bjørner, 2008]. In case the property does not hold on the circuit formula, a counterexample

is yielded, showing an erroneous circuit state. Two use cases have been studied in Chapter 5, demonstrating the applicability of the approach for error detection: (1) missing level-shifters (MLS), i.e., unprotected power domain interfaces, and (2) electrical overstress (EOS), i.e., application of voltages beyond device specifications. This work is part of our journal article [Oulkaid et al., 2025].

Missing Level-Shifter

When it comes to MLS, our experiments have shown that circuits with a few hundreds of transistors are analyzed almost instantaneously, which makes it convenient for analyzing small assemblies of basic circuit cells. Depending on the encoding (i.e., circuit semantics) chosen, circuits with thousands of transistors could be analyzed in a reasonable amount of time and memory usage. The presented technique have been published in [Oulkaid et al., 2024]. Experimental results have shown that semantics $\mathcal{S}^q$ results in a large time overhead, compared with $\mathcal{S}^s$ and $\mathcal{S}^t$. As $\mathcal{S}^t$ is more precise than $\mathcal{S}^s$ with being only marginally slower (globally), it is so far the best option for the specific case of MLS identification.

Electrical Overstress

Our experimental benchmarks regarding EOS have shown that semantics $\mathcal{S}^s$ and $\mathcal{S}^t$ are practically the same, both in terms of their results and in terms of their solving time. This observation however only applies to the specific circuits database used, and is not a general statement. Semantics $\mathcal{S}^q$ times-out more often, but allows reducing the number of false alarms (with respect to SPICE simulations). A judicious implementation choice would combine several approaches, typically running the fast $\mathcal{S}^t$ analysis and possibly falling-back to $\mathcal{S}^q$ only when $\mathcal{S}^t$ fails to prove the absence of a property.

7.1.3 Circuit Reliability Analysis

The case study we considered for reliability analysis is that of finding the greatest lower bound for circuit reliability. It is an optimization problem for which we use optimization modulo theories (OMT) solving. As seen via experimental results of Chapter 6, $\mathcal{S}^q$ is the one which performs better. It, however, remains unsound (as the linear approximations of device I-V characteristics are not conservative). $\mathcal{S}^t$ is sound, but times-out more often. Performances of $\mathcal{S}^s$ are marginally better than $\mathcal{S}^t$, yet the analysis is not sound. It is therefore not worth considering. Given that, on the benchmarks considered, $\mathcal{S}^q$ results are still close to conclusions obtained from SPICE simulations, tradeoffs are to be made on whether one must use $\mathcal{S}^t$ or $\mathcal{S}^q$. Ideally, some combination of both would be a good choice, either by using different encodings for devices in the same circuit, or deciding in advance which semantics is best suited for which kind of circuits. To attain such conclusions, further experiments are needed—ones which can help us define criteria (regarding circuit topology, size, etc.) for categorizing circuits.

7.1.4 Feedback on the Industrial Context of this Thesis

The modeling techniques studied in this thesis were theoretically compared to each other (Section 4.7), but also empirically evaluated on real-life industrial circuits. Such experiments were made possible thanks to a collaboration with the Aniah company [Aniah, 2025], where this thesis was partially conducted as part of the Convention Industrielle de Formation par la Recherche (CIFRE) program. Also, feedback from application engineers have been essential in better understanding verification challenges circuit designers have to deal with. This was for us an opportunity to a have a short development-validation circuit when defining new circuit semantics and/or applying them to new use cases.

Aniah have confirmed the usefulness of our formal approach, and thus have adopted the approach in their analysis framework. To date, the MLS detection technique presented in Section 5.1 have been implemented in their OneCheck tool.

7.2 Future Research Perspectives

The approach presented in this thesis for translating circuit descriptions to logic formulas constitutes a common foundation taking into account multiple aspects of formal circuit verification at transistor level. It could be extended in different ways. Apart from the possible short-term extensions previously exposed in Sections 4.8, 5.2.3 and 6.5, we consider two main research directions that would be valuable to investigate. These are presented in the following sections.

7.2.1 Modular Analysis Techniques

This work would greatly benefit from being supplemented with classic modular analysis techniques, especially since the transistor level circuits we analyse are commonly described hierarchically. In this regard, we identify two possible approaches—which could be combined. The first one is a top-down approach, in which each sub-circuit's inputs could be propagated from the top-level circuit. This would ensure more realistic hypotheses on input constraints ($\mathbf{I}_{\text{def}}$). In this respect, our framework is advantageous as it separates circuit semantics from environment constraints, i.e., $\mathbf{I}_{\text{def}}$ is fully customizable. As a second approach, one can use a bottom-up strategy, where the legitimacy of an error configuration on a sub-circuit is verified higher in the hierarchy. Whenever an error is yielded on a given sub-circuit, one could proceed with either (1) checking the satisfiability of the error on upper-level circuits, trying to find a counterexample disproving the error—using counterexample-guided abstraction refinement (CEGAR), or (2) building a predicate defining a pre-condition for the absence of error. The pre-condition could then be iteratively transmitted to upper levels, attempting to prove it.

Modular analysis can also be applied in the context of reliability analysis. It can improve the scaling of the approach by considering circuits' hierarchy in semantics encoding. Since our approach consists of identifying the greatest lower bound for reliability, we can use analysis results on subcircuits to compute a safe lower bound for reliability of full designs, by simply multiplying probabilities associated with the subcircuits they are made of. This technique has the advantage of being fast: once all subcircuits are analyzed independently, the full design's lower bound for reliability is computed in $O(N)$—N being the number of subcircuits. However, the usefulness of the results from such analysis can be discussed: the estimated lower bound can be unrealistically low, as it supposes all subcircuits are in their worst case. This may never occur if the semantics of the full design is considered.

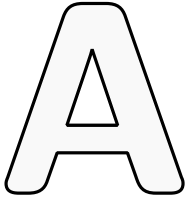
7.2.2 Error Diagnosis: Identifying the Minimal Error Model

When using a verification tool in a 'debugger mode', it is of great interest to have the errors reported in such a way that traces them back to their root cause. Sometimes, the error condition is indifferent to part of the circuit environment (i.e., a subset of pins). There may be a small subset of supplies and inputs which, when having specific voltages, trigger the error. The idea behind error diagnosis is that only relevant variables in the error model responsible for the error should actually be reported.

A first strategy would be to enumerate error models per device, then run a post-processing step which *refactors* analysis results. Such approach is undoubtedly costly, as we are faced with combinational explosion in terms of error models. As a complement, device dependencies can be taken into account in determining the order of priorities between different errors. For example, if the graph of cause-effect contains information that some device M_b depends on another device M_a, then it makes sense to first provide diagnosis about M_a. This is because an error on M_b may be a direct effect of the error on M_a, and, thus, fixing M_a (the root cause) results in the error on M_b to disappear.

Finally, through this thesis we studied a topic that was until now quite poorly covered: the use of formal methods for transistor level circuit verification. This thesis, in fact, lays out a foundation for future works in this research theme, highlighting the added value of a totally novel SMT/OMT based approach in transistor level circuit analysis. The contributions of this work have shown to be very useful for industrial use cases, and part of these contributions has been adopted in a production tool. Both short- and long-term perspectives are now open for study. Future contributions will

undoubtedly improve our understanding of which (combination of) analysis strategies are suited for which purpose. Any insights gained would be helpful in examining further hypotheses regarding performance versus precision tradeoffs. All of such outcomes would benefit engineers in building more robust circuit verification tools.



Résumé Substantiel

CETTE THÈSE a été réalisée dans le cadre de la Convention Industrielle de Formation par la Recherche (CIFRE), en collaboration avec l'entreprise ANIAH [Aniah, 2025]. Ce contexte industriel nous a permis de mieux cerner les problèmes de vérification considérés dans cette thèse. Il nous a également permis d'évaluer, en permanence, et de manière empirique, les approches développées sur des bases de données de circuits provenant essentiellement d'ANIAH.

Contexte historique. Vérifier un circuit intégré a longtemps été une tâche non triviale. Depuis au moins la fin des années 90, une tendance à passer 40–50 % du temps de conception des circuits intégrés dans le test et la validation est constatée dans les pratiques des entreprises de l'industrie des semiconducteurs [Burger and Goodman, 1997]. Ce pourcentage du temps passé dans la phase de vérification est passé à 50–60 % pour la grande majorité des projets réalisés en 2024 [Foster, 2024]. Aujourd'hui, il n'est pas rare que les puces électroniques contiennent plusieurs douzaines de milliards de transistors [Alam and Alam, 2020], contribuant ainsi à rendre la phase de vérification de plus en plus longue (et donc de plus en plus coûteuse). Dans les circuits ASIC (pour *application-specific integrated circuit*), au moins 60 % des fautes logiques et fonctionnelles ont été causées par des erreurs de conception sur la période 2016–2024 [Foster, 2024].

Niveaux d'abstraction et détection d'erreurs. Les erreurs fonctionnelles sont généralement détectables au niveau RTL (pour *register-transfer level*) [Gupta, 1992; Ashar et al., 1998; Irfan et al., 2016]. Les erreurs physiques, quant à elles, nécessitent d'intervenir à un niveau d'abstraction plus bas pour les détecter, notamment le niveau de disposition (i.e., *layout*)—où l'on a accès aux informations sur la géométrie et le placement des composants [Gibson et al., 2010; Kollu et al., 2012; Lescot et al., 2015]. Puis, entre le niveau RTL et le niveau de disposition, plusieurs types de propriétés peuvent être étudiés. C'est dans ces niveaux intermédiaires que les erreurs dites *électriques* peuvent être identifiées—c'est le cadre de cette thèse. Les propriétés électriques nécessitent une connaissance des schémas de circuits, ainsi que des scénarios d'alimentation qui leurs sont associés. Ces informations sont accessibles au niveau transistor et à des niveaux d'abstraction inférieurs, tels que le niveau de disposition, voire le niveau physique. Des règles de conception ont été établies, à partir d'une classification des bogues existants, afin d'éviter les erreurs de conception. Ces règles visent à protéger le circuit contre divers types de dommages et de dysfonctionnements. La violation des règles peut également entraîner des erreurs fonctionnelles. Nous considérons qu'il existe deux types de vérification associés : (1) la vérification des règles de conception (DRC, pour *design rule checking*), qui vise à vérifier les contraintes liées à la disposition physique du circuit, et (2) la vérification des règles électriques (ERC, pour *electrical rule checking*), qui vérifie le comportement électrique du circuit. C'est sur l'ERC que se focalise cette thèse.

Règles de conception électrique. Les approches ERC existantes sont principalement basées sur la simulation. Certaines d'entre elles combinent la simulation avec d'autres analyses *ad hoc*. La réalisation de simulations peut prendre beaucoup de temps, en particulier lorsque l'objectif est de vérifier l'absence d'une erreur spécifique sur un grand nombre d'états de circuit, sans parler de l'espace d'état de l'ensemble du circuit. Parfois, l'espace d'état peut même être infini si l'on considère des valeurs de tension continues, en particulier dans les circuits à alimentation multiple. Les techniques d'analyse classiques comprennent les algorithmes de propagation de tension et des analyses topologiques des circuits. Bien qu'elles puissent être rapides, elles sont connues pour générer de nombreuses fausses alarmes. À cet égard, les méthodes formelles ont le potentiel d'être plus performantes en termes de réduction des fausses alarmes. Pourtant, l'utilisation des approches formelles dans le contexte de l'ERC a rarement été étudiée. Le Chapitre 3 présente le contexte lié au domaine de la conception des circuits intégrés. Il se focalise sur le niveau transistor et la terminologie associée. Le chapitre présente les problèmes liés aux erreurs électriques, l'état de l'art dans la vérification des règles de conception électriques, ainsi que des perspectives d'amélioration. Le contenu de ce chapitre provient essentiellement de notre article de revue [Ferres et al., 2025]. L'article apporte une vue d'ensemble des techniques de vérification des properiétés éléctriques dans des circuits, propose une nouvelle taxonomie permettant de classifier les différentes techniques, et mets en évidence les défis liés à ce domaine de recherche.

Objectifs de la thèse. En 2022 (date de début de cette thèse) certains des travaux les plus remarquables incluaient (1) l'utilisation de solveurs de satisfiabilité (SAT) pour la détection des conditions des courts-circuits dans les circuits logiques [Afonso and Monteiro, 2017a], et (2) la vérification des paramètres des circuits analogiques en utilisant des solveurs SAT [Miller and Brewer, 2013]. Bien que ces travaux aient donné des résultats prometteurs quant à l'utilisation des méthodes formelles dans le contexte de l'ERC, aucune autre piste de recherche n'a été explorée depuis. Au début de cette thèse, nous manquions de techniques de vérification allant au-delà de la résolution SAT, et qui soient *à usage général* (dans le sens où elles sont utilisables pour vérifier différentes sortes d'erreurs, et non seulement dédiés au cas d'usage pour lequel elles étaient initialement conçues). Nous manquions également d'approches où l'analyse des circuits est indépendante de la topologie et où l'analysabilité ne dépend pas des hypothèses sur la configuration d'alimentation du circuit. Plusieurs approches de modélisation ont été explorées dans cette thèse en tenant compte de tous ces aspects. Leur utilité a été étudiée dans plusieurs cas d'usage, étayée à la fois par des démonstrations formelles et des évaluations empiriques. Dans cette thèse, nous ciblons non seulement les circuits numériques, mais aussi les conceptions à alimentation multiple. Pour ce faire, nous définissons la sémantique des circuits au niveau transistor, que nous utilisons pour générer des problèmes de satisfiabilité modulo théories (SMT). Les approches d'analyse que nous introduisons sont applicables à toutes les descriptions de circuits (notamment les conceptions à alimentation multiple, et avec des parties analogiques) et sont hautement personnalisables. Cela signifie que des hypothèses des utilisateurs peuvent être prises en compte. Il est en plus possible de formuler des propriétés électriques personnalisées et de les vérifier. Nous appliquons la sémantique des circuits développée dans cette thèse à deux problèmes distincts. Le premier est le problème de la vérification de la possibilité d'une configuration d'erreur sur un circuit. Les erreurs électriques que nous étudions sont la surtension électrique (EOS, pour *electrical overstress*), et les *décaleurs de niveau de tension* manquants (MLS, pour *missing level-shifter*). Le deuxième problème étudié est l'analyse de la fiabilité des circuits par rapport au mécanisme de défaillance lié au claquage d'oxide de la grille (TDDB, pour *time-dependent dielectric breakdown*). Plus précisément, nous formulons et résolvons le problème de l'identification de l'état du circuit le plus défavorable en termes de vieillissement. Pour ce deuxième cas d'utilisation, nous exploitons les capacités d'optimisation des solveurs SMT modernes, notamment l'extension νZ [Bjørner and Phan, 2014] du solveur Z3 [De Moura and Bjørner, 2008], qui nous permet de formuler des fonctions à maximiser ou minimiser (objectif) tout en satisfaisant les contraintes du problème SMT.

Méthodes utilisées dans cette thèse. Pour faire de la vérification formelle, plusieurs méthodes d'analyse sont possibles (e.g., vérification de modèles (*model-checking*), interprétation abstraite, assistant de preuve de théorèmes). L'apperçu des méthodes formelles du Chapitre 2 en dit un peu plus. La méthode retenue pour cette thèse est l'utilisation de procédures de décision, qui permettent

de déterminer la satisifiabilité d'une formule de logique. SAT (ou satisifiabilité) est le problème de déterminer s'il existe une affectation des variables d'une formule de logique qui fait que cette formule s'évalue à « vrai ». SMT (satisifiabilité modulo théories) considère des problèmes dont la résolution nécessite des théories au-delà de ce qu'utilise les algorithmes de SAT. Un exemple de ces théories est l'arithmétique (linéaire ou non linéaire) sur les rationnels. Des techniques de vérification électrique de circuits utilisant des solveurs SAT ont été proposées par le passé, notamment dans [Afonso and Monteiro, 2017a], pour la détection de court-circuits dans des circuit logiques. Leur approche n'est en revanche pas facilement généralisable à d'autres types d'erreurs au-delà de la détection des court-circuits, en raison du fait que les conditions d'erreur considérées sont intégrées dans les formules encodant le comportement du circuit. De plus, pour analyser des circuits à multiples niveaux d'alimentation, il est nécessaire de passer d'abord par une opération de *bit-blasting*, qui permet de traduire le problème vers SAT. Ce que nous proposons est simplement de traduire le circuit en formule SMT, en représentant les différentes tensions (et autres quantités numériques, e.g., l'intensité) par des rationnels. Nous faisons appel au solveur Z3 afin de vérifier une propritété $\mathcal{P}$ sur une formule (sémantique) de circuit $\mathcal{S}$, en déterminant la satisifiabilité de la formule $\mathcal{S} \wedge \neg\mathcal{P}$. Si le solveur répond « faux », on en déduit que le circuit vérifie la propriété. Sinon (si la réponse est « vrai »), on obtient un contre-exemple qui témoigne de la violation de la propriété. Dans un deuxième usage, nous utilisons de l'omptimisation modulo théories (OMT) pour analyser la fiabilité des circuits. Il s'agit d'une analyse pire-case, que nous détaillons plus loin.

Sémantiques de circuit. Le Chapitre 4 présente les sémantiques de circuit développées durant cette thèse. L'approche globale adoptée dans notre flot de vérification a été initialement introduite dans [Ferres et al., 2023]. Un premier essai, que nous appelons *sémantique orientée* $\mathcal{S}^o$, a été réalisé. $\mathcal{S}^o$ se base sur l'idée de modéliser le circuit de manière fonctionelle, c'est-à-dire que chaque sous-circuit calcule les valeurs des sorties en fonction des entrées, et que, quand les sous-circuits sont composés entre eux, il suffit de composer les fonctions les représentant. Cette hypothèse n'est valide que sur des topologies spéciales de circuit, notamment celles basées sur l'approche CMOS (pour *complementary metal-oxide-semiconductor*). La sémantique $\mathcal{S}^o$ présente donc de fortes limitations en termes de couverture de topologies, sans présenter d'intérêt du point de vue de la performance de l'analyse. Nous avons ensuite adopté une approche relationelle, comme alternative à l'approche fonctionelle de $\mathcal{S}^o$, tout en nous basant sur une abstraction *switch-based* pour modéliser les états des transistors. Nous appelons cette sémantique $\mathcal{S}^s$. Elle a été présentée dans un premier temps dans [Oulkaid et al., 2024]. Dans ce manuscrit, nous présentons, en plus, des extensions à cette sémantique pour (1) couvrir d'autres composants que les transistors (résistances et diodes), et (2) prendre en compte la tension de seuil V_t dans la modélisation. L'extension prenant en compte V_t est notée $\mathcal{S}^t$. Nous introduisons ensuite une sémantique quantitative qui, au lieu de modéliser les tensions des nœuds par des intervalles (notamment en cas de court-circuit), calcule des valeurs de tension des nœuds concrètes, sur la base d'une approximation des relations intensité-tension. Nous appelons cette sémantique $\mathcal{S}^q$. Elle fait partie de l'article de journal [Oulkaid et al., 2025]. De plus, le Chapitre 4 compare les sémantiques introduites ($\mathcal{S}^s$, $\mathcal{S}^t$, et $\mathcal{S}^q$), formellement. D'autres comparaisons basées sur des évaluations expérimentales sont effectuées ensuite, dans les Chapitres 5 et 6.

Détection des erreurs électriques. Le Chapitre 5 présente comment les sémantiques du Chapitre 4 peuvent être utiliées afin d'aborder des problèmes de vérification de circuits. Il est illustré par des cas d'usage industriels, pour lesquels des évaluations expérimentales ont été effectuées. Les applications considérées sont la détection des décaleurs de niveau de tension manquants (*missing level-shifters*, MLS), présentée dans [Oulkaid et al., 2024], ainsi que la détection des surtensions électriques (*electrical overstress*, EOS) présentée dans l'article de journal [Oulkaid et al., 2025].

Analyse de fiabilité des circuits. Le Chapitre 6 aborde le sous-problème de l'analyse de la fiabilité des circuits : celui d'identifier l'état correspondant au pire-cas du circuit. On formalise un tel problème (optimisation d'une fonction objectif), et on montre l'utilisation des sémantiques de circuit (introduites dans le Chapitre 4) pour le résoudre en utilisant un solveur OMT (i.e., solveur SMT avec des fonctionalités d'optimisation, notamment l'extension νZ [Bjørner and Phan,

2014] du solveur Z3). L'évaluation expérimentale est effectuée sur des cas d'étude industriels. Les contributions de ce chapitre font également partie d'un article soumis à une revue [Oulkaid et al., nd].

Perspectives de recherche. Ce travail gagnerait à être complété par des techniques classiques d'analyse modulaire, d'autant plus que les circuits au niveau transistor que nous analysons sont généralement décrits de manière hiérarchique. À cet égard, nous identifions deux approches possibles, qui pourraient être combinées. La première est une approche *top-down*, dans laquelle les entrées de chaque sous-circuit sont propagées à partir du circuit de niveau supérieur. Cela garantirait des hypothèses plus réalistes sur les contraintes sur les entrées. À cet égard, notre approche est avantageuse car elle sépare la sémantique des circuits des contraintes de l'environnement du circuit, ce qui rend ces dernières entièrement personnalisables. Comme deuxième approche, on peut utiliser une stratégie *bottom-up*, dans laquelle la légitimité d'une configuration d'erreur sur un sous-circuit est vérifiée à un niveau supérieur de la hiérarchie. Chaque fois qu'une erreur est trouvée dans un sous-circuit donné, on peut soit (1) vérifier la satisfiabilité de l'erreur sur les circuits de niveaux supérieurs, en essayant de trouver un contre-exemple réfutant l'erreur, à l'aide de techiques de raffinement d'abstraction guidée par des contre-exemples (CEGAR, pour *counterexample-guided abstraction refinement*), soit (2) construire un prédicat définissant une condition préalable à l'absence d'erreur. La condition préalable peut alors être transmise de manière itérative aux niveaux supérieurs, en essayant de la prouver. L'analyse modulaire peut également être appliquée dans le contexte de l'analyse de fiabilité. Elle peut améliorer la mise à l'échelle de l'approche en tenant compte de la hiérarchie des circuits dans l'encodage de la sémantique. Étant donné que notre approche consiste à identifier la borne inférieure pour la fiabilité, nous pouvons utiliser les résultats d'analyse sur les sous-circuits pour calculer une borne inférieure sûre pour la fiabilité des circuits complets, en multipliant simplement les probabilités associées aux sous-circuits qui les composent. Cette technique présente l'avantage d'être rapide : une fois que tous les sous-circuits sont analysés indépendamment, la borne inférieure du circuit complet est calculée en $O(N)$, N étant le nombre de sous-circuits. Cependant, l'utilité des résultats d'une telle analyse est discutable : la borne inférieure estimée peut être irréaliste, car elle suppose que tous les sous-circuits se trouvent, chacun, dans leurs pire cas local. Cela peut ne jamais se produire si la sémantique du circuit complet est été prise en compte.

Compléter un outil de vérification avec un mode « débogueur » serait très utile: cela permettrait de signaler les erreurs dès la détection de leur cause première (*route cause*). Parfois, la condition d'erreur est indépendante d'une partie de l'environnement du circuit (c'est-à-dire d'un sous-ensemble de broches). Il peut y avoir un petit sous-ensemble d'alimentations et d'entrées qui, lorsqu'elles ont des tensions spécifiques, déclenchent l'erreur. L'idée derrière le diagnostic d'erreur est que seules les variables pertinentes du modèle d'erreur responsable de l'erreur doivent être signalées. Une première stratégie consisterait à énumérer les modèles d'erreur par composant, puis à exécuter une étape de post-traitement qui refactorise les résultats de l'analyse. Une telle approche est sans aucun doute coûteuse, car nous sommes confrontés à une explosion combinatoire en termes de modèles d'erreur. En complément, les dépendances entre les composants peuvent être prises en compte pour déterminer l'ordre de priorité entre les différentes erreurs.

Conclusion. Cette thèse nous a permis d'étudier un sujet jusqu'à présent peu abordé : l'utilisation de méthodes formelles pour la vérification des circuits au niveau transistor. Cette thèse représente une base des futurs travaux dans ce domaine de recherche, en soulignant la valeur ajoutée d'une approche basée sur SMT et OMT. Les contributions de ce travail se sont révélées très utiles pour des cas d'utilisation industriels, et une partie de ces contributions ont été adoptées dans un outil de production. Des perspectives à court et à long terme s'ouvrent désormais à l'étude. Les contributions futures amélioreraient notre compréhension de quelle stratégie (ou combinaison de stratégies) d'analyse est adaptée pour quel objectif. Toutes les connaissances acquises seront utiles pour examiner d'autres hypothèses concernant le compromis entre la performance et la précision de l'analyse. Tous ces résultats aideront les ingénieurs à créer des outils de vérification des circuits plus robustes.

B

Oriented Signal Circuit Semantics

THE INITIALLY ATTEMPTED CIRCUIT SEMANTICS uses a reasoning about circuits in terms of functions rather than relations. The intuition behind it is presented in Section 4.4. This appendix details the semantics rules used to encode circuits into logic formulas based on such reasoning. As previously mentioned in Section 4.4.2, the semantics in this appendix presents some limitations, namely due to strong hypotheses about the circuits being modeled and analysed.

B.1 Virtual Nets

For each transistor, we create one additional variable, representing a drain voltage value computed locally (e.g., nets Z1 and Z2 of Figure 4.17 (page 55) as opposed to net Z). It serves as a computation intermediate. We call such variables 'virtual drains'. We denote $\widehat{\mathbf{N}}$ the set of all virtual drains of a circuit, whose cardinal is basically the size of the set of transistors. A transistor's virtual drain is given by function

$$\mathrm{D\widehat{R}N}\colon \mathbf{D}_{\mathtt{MOS}} \longrightarrow \widehat{\mathbf{N}}.$$

B.2 Voltage Abstraction

We use an abstract representation, in terms of integer numbers, of the *real* voltage values available in a circuit. We associate an even non-negative integer number for every supply in the circuit, so that it preserves the order relation. We keep odd positive numbers to abstractly represent all possible values between two successive supply values. This is illustrated with Figure B.1, which is valid for a circuit having two supplies (1.2 V and 3.3 V) and one ground (0 V). Function

$$\mathcal{V}^o\colon \mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{V}$$

is the corresponding abstraction of the voltage of a net, where $\mathbb{V} \triangleq \mathbb{N}$ is the set of non-negative integer numbers. Particularly, for supplies, we have

$$\mathcal{V}^o\colon \mathbf{S} \rightharpoonup 2\mathbb{V}.$$

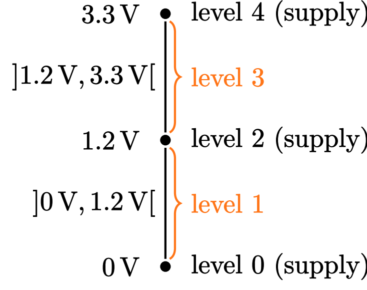


Figure B.1: Abstraction of voltage levels from actual supplies voltage values.

B.3 Net Status

The notion of net status is used to reason about the way a net is connected, in a given circuit state, to the supplies of the circuit. When a net is connected to a single supply, we say it has a *consistent* value. It is in a short-circuit state when connected to more than one supply. In case it is connected to no supply, it is floating. This is given by function

$$\mathcal{S}tatus\colon \mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \{\mathsf{Const}, \mathsf{Short}, \mathsf{Float}\}.$$

We assume all supplies are consistent, i.e.,

$$\forall s \in \mathbf{S},\ \mathcal{S}tatus(s) = \mathsf{Const}.$$

We introduce three predicates to encode net status:

$$\mathcal{C}onst\colon \mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B},$$

$$\mathcal{S}hort\colon \mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B},$$

$$\mathcal{F}loat\colon \mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B}.$$

To force mutual exclusion of net status, i.e., each net can only have one status, we have

$$\begin{aligned} &\forall n \in \mathbf{N} \cup \widehat{\mathbf{N}}, \\ \mathcal{C}onst(n) &\Leftrightarrow \mathcal{S}tatus(n) = \mathsf{Const} \\ \mathcal{S}hort(n) &\Leftrightarrow \mathcal{S}tatus(n) = \mathsf{Short} \\ \mathcal{F}loat(n) &\Leftrightarrow \mathcal{S}tatus(n) = \mathsf{Float}. \end{aligned}$$

B.4 Transistor States

We define an additional short notation for virtual transistor drain voltages as

$$\begin{aligned} \mathcal{V}^o_{\hat{\mathrm{D}}}\colon \mathbf{D}_{\mathtt{MOS}} &\longrightarrow \mathbb{N} \\ X &\longmapsto \mathcal{V}^o\left(\widehat{\mathrm{DRN}}(X)\right). \end{aligned}$$

The encoding of predicate $\mathcal{O}n\colon \mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{B}$, which represents a transistors on/off state, is done via semantics rules. Rule $\mathcal{R}^o_{\mathtt{NMOS}}$ gives the definition for NMOS devices, and $\mathcal{R}^o_{\mathtt{PMOS}}$ gives the definition for PMOS devices, with:

$$\forall M \in \mathbf{D}_{\mathtt{MOS}},\ \mathcal{V}^o_{\mathrm{G}}(M) \triangleq \mathcal{V}^o(\mathrm{GATE}(M)),$$

$$\forall M \in \mathbf{D}_{\mathtt{MOS}},\ \mathcal{V}^o_{\mathrm{S}}(M) \triangleq \mathcal{V}^o(\mathrm{SRC}(M)).$$

The encoding is based on the switch-based abstraction, as well as the assumption that information flows from source to drain. We ignore, for this semantics, the effect of threshold voltage. An NMOS device is turned on when its gate voltage exceeds its source voltage, leading to a connection between source and virtual drain. The voltage of the virtual drain gets the voltage of the source. The same applies to source and drain status information. When the $\mathcal{On}$ predicate is $\perp$ (i.e., the device is turned off), the only constraint which applies is that the virtual drain is floating—signifying the absence of a connection across the device. For a PMOS device, the activation condition is complementary to the NMOS device type, but the consequences are the same.

$$\bigwedge_{M \in \mathbf{D}_{\texttt{NMOS}}} \left(\begin{array}{ll} & \mathcal{On}(M) \triangleq \mathcal{V}^o_{\text{G}}(M) > \mathcal{V}^o_{\text{S}}(M) \\ \wedge & \mathcal{On}(M) \Rightarrow \left(\begin{array}{l} \mathcal{V}^o_{\hat{\text{D}}}(M) = \mathcal{V}^o_{\text{S}}(M) \\ \wedge\ \mathcal{Status}(\widehat{\text{DRN}}(M)) = \mathcal{Status}(\text{SRC}(M)) \end{array} \right) \\ \wedge & \neg\mathcal{On}(M) \Rightarrow \mathcal{Float}(\widehat{\text{DRN}}(M)) \end{array} \right) \qquad (\mathcal{R}^o_{\texttt{NMOS}})$$

$$\bigwedge_{M \in \mathbf{D}_{\texttt{PMOS}}} \left(\begin{array}{ll} & \mathcal{On}(M) \triangleq \mathcal{V}^o_{\text{G}}(M) < \mathcal{V}^o_{\text{S}}(M) \\ \wedge & \mathcal{On}(M) \Rightarrow \left(\begin{array}{l} \mathcal{V}^o_{\hat{\text{D}}}(M) = \mathcal{V}^o_{\text{S}}(M) \\ \wedge\ \mathcal{Status}(\widehat{\text{DRN}}(M)) = \mathcal{Status}(\text{SRC}(M)) \end{array} \right) \\ \wedge & \neg\mathcal{On}(M) \Rightarrow \mathcal{Float}(\widehat{\text{DRN}}(M)) \end{array} \right) \qquad (\mathcal{R}^o_{\texttt{PMOS}})$$

B.5 Net Drivers

For each net in the circuit, we define the set of drivers as shown in DRIV_{def}. Drivers of a net $n \in \mathbf{N}$ are the elements of the set of virtual drains of the transistors for which n is a drain. Net drivers are used to compute both the resulting voltage value and electrical status.

$$\begin{aligned} \text{DRIV}\colon \mathbf{N} &\longrightarrow \mathscr{P}(\widehat{\mathbf{N}}) \\ n &\longmapsto \bigcup_{M \in \mathbf{D}_{\texttt{MOS}}} \{\widehat{\text{DRN}}(M) \mid n = \text{DRN}(M)\} \end{aligned} \qquad (\text{DRIV}_{\text{def}})$$

We define the following short notations:

$$\forall n \in \mathbf{N},\ \text{DRIV}_{\text{const}}(n) \triangleq \{d \in \text{DRIV}(n) \mid \mathcal{Const}(d)\}$$

$$\forall n \in \mathbf{N},\ \text{DRIV}_{\text{short}}(n) \triangleq \{d \in \text{DRIV}(n) \mid \mathcal{Short}(d)\}$$

$$\forall n \in \mathbf{N},\ \text{DRIV}_{\text{float}}(n) \triangleq \{d \in \text{DRIV}(n) \mid \mathcal{Float}(d)\}$$

For each net $n \in \mathbf{N}$, driver configurations can be divided into four mutually exclusive cases. Each case corresponds to one (and only one) of the following four predicates to be $\top$:

$$
\begin{aligned}
\forall n \in \mathbf{N},\quad \textsc{EqShort}(n) \quad & \triangleq |\textsc{Driv}_{\text{short}}(n)| \geq 1 \\
& \wedge \left(\bigwedge_{\substack{d_i, d_j \in \textsc{Driv}(n) \smallsetminus \textsc{Driv}_{\text{float}}(n) \\ i \neq j}} \mathcal{V}^o(d_i) = \mathcal{V}^o(d_j) \right) \\
\forall n \in \mathbf{N},\quad \textsc{EqConst}(n) \quad & \triangleq |\textsc{Driv}_{\text{const}}(n)| \geq 1 \\
& \wedge |\textsc{Driv}_{\text{short}}(n)| = 0 \\
& \wedge \left(\bigwedge_{\substack{d_i, d_j \in \textsc{Driv}_{\text{const}}(n) \\ i \neq j}} \mathcal{V}^o(d_i) = \mathcal{V}^o(d_j) \right) \\
\forall n \in \mathbf{N},\quad \textsc{AllFloat}(n) \quad & \triangleq |\textsc{Driv}_{\text{float}}(n)| = |\textsc{Driv}(n)| \\
\forall n \in \mathbf{N},\quad \textsc{DiffShort}(n) \quad & \triangleq |\textsc{Driv}_{\text{const}}(n)| + |\textsc{Driv}_{\text{short}}(n)| \geq 1 \\
& \wedge \left(\bigvee_{\substack{d_i, d_j \in \textsc{Driv}(n) \smallsetminus \textsc{Driv}_{\text{float}}(n) \\ i \neq j}} \mathcal{V}^o(d_i) \neq \mathcal{V}^o(d_j) \right)
\end{aligned}
$$

We also define:

$$
\forall n \in \mathbf{N},\ \mathcal{V}^o_{\min}(n) \triangleq \begin{cases} \min\left(\bigcup_{d \in \textsc{Driv}(n)} \mathcal{V}^o(d) \right) & \text{if } \textsc{AllFloat}(n) \\ \min\left(\bigcup_{d \in \textsc{Driv}(n) \smallsetminus \textsc{Driv}_{\text{float}}(n)} \mathcal{V}^o(d) \right) & \text{otherwise} \end{cases}
$$

$$
\forall n \in \mathbf{N},\ \mathcal{V}^o_{\max}(n) \triangleq \begin{cases} \max\left(\bigcup_{d \in \textsc{Driv}(n)} \mathcal{V}^o(d) \right) & \text{if } \textsc{AllFloat}(n) \\ \max\left(\bigcup_{d \in \textsc{Driv}(n) \smallsetminus \textsc{Driv}_{\text{float}}(n)} \mathcal{V}^o(d) \right) & \text{otherwise} \end{cases}
$$

For each case of drivers configuration we define the merge rules ($\mathcal{R}^o_{\text{MERGE}}$) for a net $n \in \mathbf{N}$.

If the set of drivers of n contains at least one element that has a short-circuit status and that all of the short-circuited and consistent drivers happen to have the same voltage (i.e., $\text{EqShort}(n) = \top$), the status of n is necessarily short-circuit and the $\mathcal{V}^o(n)$ is the same as the voltage of consistent and short-circuited drivers.

If there is at least one consistent driver, with none being in short-circuit, and with all consistent drivers having the same voltage (i.e., $\text{EqConst}(n) = \top$), the merge operation results in a consistent net with the same voltage.

The result of the merge can be floating if and only if all drivers are floating (i.e., $\text{AllFloat}(n) = \top$). The net n will then have the voltage of one of its drivers.

Finally, if n is driven by different drivers, with non-floating drivers having different voltages (i.e., $\text{DiffShort}(n) = \top$), then net n is in short-circuit with its voltage belonging to an interval defined by its non-floating drivers.

$$\begin{aligned}
\forall n \in \mathbf{N},\quad \text{EqShort}(n) &\Rightarrow \mathcal{S}hort(n) \land \left(\bigwedge_{d \in \text{Driv}_{\text{const}}(n) \cup \text{Driv}_{\text{short}}(n)} \mathcal{V}^o(n) = \mathcal{V}^o(d) \right) \\
\forall n \in \mathbf{N},\quad \text{EqConst}(n) &\Rightarrow \mathcal{C}onst(n) \land \left(\bigwedge_{d \in \text{Driv}_{\text{const}}(n)} \mathcal{V}^o(n) = \mathcal{V}^o(d) \right) \\
\forall n \in \mathbf{N},\quad \text{AllFloat}(n) &\Rightarrow \mathcal{F}loat(n) \land \left(\mathcal{V}^o(n) \in \bigcup_{d \in \text{Driv}(n)} \mathcal{V}^o(d) \right) \\
\forall n \in \mathbf{N},\quad \text{DiffShort}(n) &\Rightarrow \mathcal{S}hort(n) \land \left(\mathcal{V}^o(n) \in \left]\mathcal{V}^o_{\min}(n), \mathcal{V}^o_{\max}(n)\right[\right)
\end{aligned} \qquad (\mathcal{R}^o_{\text{MERGE}})$$

B.6 Circuit Formula

Building the corresponding circuit formula, denoted $\mathcal{S}^o$, consists in applying the previously defined semantics rules on a circuit description ($\mathcal{S}^o{}_{\text{def}}$). Note that the definition of $\mathcal{S}^o$ includes constraints on supplies ($\mathcal{R}_{\mathbf{S}}$) and inputs ($\mathcal{R}_{\mathbf{I}}$), previously defined (page 52).

$$\mathcal{S}^o \triangleq \mathcal{R}^o_{\texttt{NMOS}} \land \mathcal{R}^o_{\texttt{PMOS}} \land \mathcal{R}^o_{\text{MERGE}} \land \mathcal{R}_{\mathbf{S}} \land \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}^o{}_{\text{def}})$$

B.7 Example

We consider a CMOS-based NAND gate (depicted in Figure B.2), where we restrict the supply voltages to 5 V for the primary supply (`VDD`) and to 0 V for the common ground (`GND`). From applying our voltage abstraction, we end up having three abstract values: 0, 1 and 2 (Figure B.3) to which we restrict input values. Table B.1 enumerates the obtained solutions from running the oriented signal semantics $\mathcal{S}^o$, with all combinations of input values over `A` and `B`, assuming inputs are chosen among supply levels 0, 1, and 2.

Note. The limitations of semantics $\mathcal{S}^o$ are discussed in Section 4.4.2.

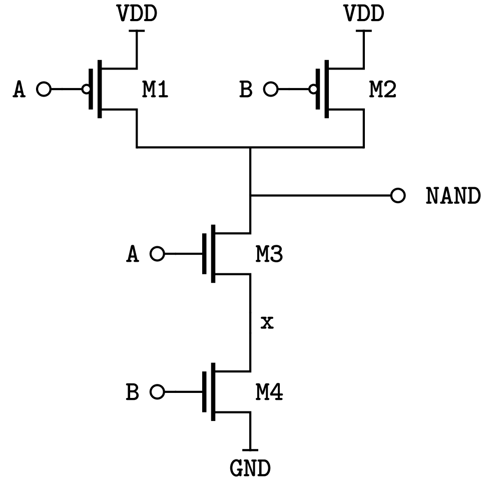


Figure B.2: (Reminder) CMOS-based NAND gate.

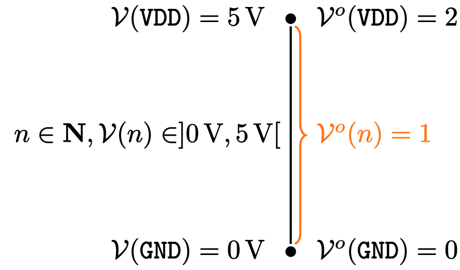


Figure B.3: NAND gate voltage abstraction.

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$\mathcal{V}^o(\texttt{A})$	0	0	0	2	2	2	1	1	1
$\mathcal{V}^o(\texttt{B})$	0	2	1	0	2	1	0	2	1
$\mathcal{V}^o(\texttt{NAND})$	2	2	2	2	0	1	2	1	1
$\mathcal{O}n(\texttt{M1})$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$
$\mathcal{O}n(\texttt{M2})$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$
$\mathcal{O}n(\texttt{M3})$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\mathcal{O}n(\texttt{M4})$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$

Table B.1: Enumeration of NAND gate solutions (each column represents a solution), numbered s_0–s_8.

C

Various Attempts of Quantitative Semantics

Quantitative circuit semantics ($\mathcal{S}^q$) introduced in Chapter 4 were preceded by multiple trials (Sections C.1 to C.4). This appendix presents each of these trials (labeled $\mathcal{S}^q_1$ to $\mathcal{S}^q_4$). It illustrates, with examples, what does not work about them. The one we finally adopt as quantitative semantics (labeled $\mathcal{S}^q_5$) is briefly introduced in Section C.5. Its semantics rules are thoroughly presented in the body of the thesis—in Section 4.6.

C.1 $r_{\mathtt{ON}}/r_{\mathtt{OFF}}$ without Threshold ($\mathcal{S}^q_1$)

To quantify the amount of voltage drop in switch-based semantics (namely, $\mathcal{S}^s$), one would, for example, think of associating different voltage drop quantities to different transistor states. For instance, a transistor that is switched-off can be associated a high resistance value $r_{\mathtt{OFF}}$ (ideally, infinite) between its source and drain nets, while a switched-on transistor can be associated a negligible resistance $r_{\mathtt{ON}}$ (ideally, null). An infinite resistance can be modeled with an open switch, while a simple wire ideally has zero resistance. We do not consider such ideal assumptions. Instead, we use pre-assigned values that are more realist. Specifically, $r_{\mathtt{OFF}} = 10^9\ \Omega$ and $r_{\mathtt{ON}} = 10^2\ \Omega$. This choice reflects the behavior of transistors: a switched-on transistor's source-to-drain connection behaves like a low resistive path, while such connection behaves like a high resistive path when the transistor is switched-off.

The idea is to substitute the $\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$ rule, in $\mathcal{S}^s$, with one or more new rules which reason about resistance values associated with transistor states, in order to determine the corresponding voltage drops (and by consequence, current) in a quantitative manner—with precise values—instead of intervals of possible values. To do so, we associate for each transistor, a current function $\mathcal{I}\colon \mathbf{N} \times \mathbf{D}_{\mathtt{MOS}} \times \mathbf{N} \longrightarrow \mathbb{I}$, such that, for a given transistor M, the quantity $\mathcal{I}(\textsc{Src}(M), M, \textsc{Drn}(M))$ represents the source-to-drain current. Function $\mathfrak{R}_{eq}\colon \mathbf{D}_{\mathtt{MOS}} \longrightarrow \{r_{\mathtt{ON}}, r_{\mathtt{OFF}}\}$ gives for each transistor a resistance value among $\{r_{\mathtt{ON}}, r_{\mathtt{OFF}}\}$. Intensity-Voltage (I-V) relations are then defined, with Ohm's law ($\mathcal{R}_{\text{Ohm}}$). In addition, Kirchhoff's current law is ensured on circuit nets with the previously defined $\mathcal{R}_{\text{Kirchhoff}}$ (page 66).

$$\bigwedge_{M \in \mathbf{D}_{\mathtt{MOS}}} \Big(\mathcal{V}_{\mathrm{S}}(M) - \mathcal{V}_{\mathrm{D}}(M) = \mathfrak{R}_{eq}(M) \times \mathcal{I}\big(\mathcal{V}_{\mathrm{S}}(M), M, \mathcal{V}_{\mathrm{D}}(M)\big)\Big) \qquad (\mathcal{R}_{\text{Ohm}})$$

Resistance values $\mathfrak{R}_{eq}(M)$, for each device $M \in \mathbf{D}_{\mathtt{MOS}}$, are constrained by transistor states ($\mathcal{R}_{\mathfrak{R}_{eq}}$).

$$\bigwedge_{M \in \mathbf{D}_{\mathtt{MOS}}} \left(\begin{array}{ll} & \mathcal{O}n(M) \Leftrightarrow \mathfrak{R}_{eq}(M) = r_{\mathtt{ON}} \\ \wedge & \neg\mathcal{O}n(M) \Leftrightarrow \mathfrak{R}_{eq}(M) = r_{\mathtt{OFF}} \end{array} \right) \qquad (\mathcal{R}_{\mathfrak{R}_{eq}})$$

Transistor states (defined with the $\mathcal{O}n$ predicate), on the other hand, use the same $\mathcal{O}n_{\text{def}}^{\texttt{PMOS}}$ and $\mathcal{O}n_{\text{def}}^{\texttt{NMOS}}$ rules as with $\mathcal{S}^s$.

We denote this semantics $\mathcal{S}_1^q$, and we define it as a conjunction of semantics rules, including supplies and inputs constraints ($\mathcal{R}_{\mathbf{S}}$ and $\mathcal{R}_{\mathbf{I}}$), as show in $\mathcal{S}_{1\,\text{def}}^q$.

Note. The $\mathcal{O}n$ predicates are *indirectly* constrained by $\mathcal{R}_{\mathfrak{R}_{eq}}$.

$$\mathcal{S}_1^q \triangleq \mathcal{R}_{\text{Ohm}} \wedge \mathcal{R}_{\text{Kirchhoff}} \wedge \mathcal{R}_{\mathfrak{R}_{eq}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}_{1\,\text{def}}^q)$$

Example and Limitation

We consider the Inverter circuit example, on which we apply $\mathcal{S}_1^q$. When setting the input (A) voltage to the ground, the PMOS device Mp is switched-on (with equivalent resistance $r_{\texttt{ON}}$), and the NMOS device Mn is switched-off (with equivalent resistance $r_{\texttt{OFF}}$). The output's voltage $\mathcal{V}(\texttt{Z})$ is close to (but is never exactly equal to) $\mathcal{V}(\texttt{VDD})$. This is illustrated in Figure C.1.

This modeling raises one main issue, which we can clearly see when net Z is used to drive some other circuitry. For example, if net Z is used as the gate of another inverter (or a chain of inverters in series), both NMOS and PMOS devices of the next inverter would be modeled as being switched-on (as the threshold voltage in this semantics variant is considered to be null)—which is incorrect. This suggests to consider the threshold voltage in the constraints which determine device states (the choice of $r_{\texttt{ON}}$ and $r_{\texttt{OFF}}$ values should be relevant with respect to the threshold voltage). We do this in the next semantics variant (the following section).

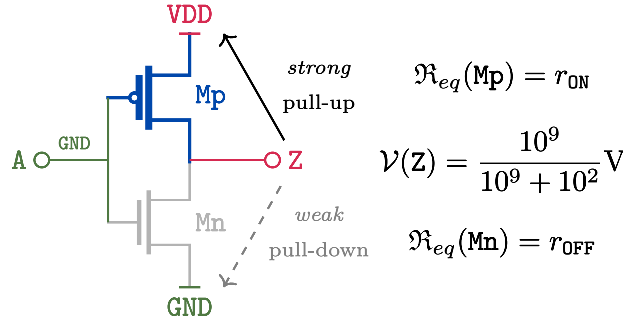


Figure C.1: Inverter's device states and output valuation with $\mathcal{S}_1^q$. We set $\mathcal{V}(\texttt{GND}) = 0\,\text{V}$ and $\mathcal{V}(\texttt{VDD}) = 1\,\text{V}$.

C.2 $r_{\texttt{ON}}/r_{\texttt{OFF}}$ with Threshold ($\mathcal{S}_2^q$)

By simply taking into account the threshold voltage V_t (typically, $V_t = 200\,\text{mV}$), i.e., replacing definitions $\mathcal{O}n_{\text{def}}^{\texttt{NMOS}}$ and $\mathcal{O}n_{\text{def}}^{\texttt{PMOS}}$, respectively, with $\mathcal{O}n^t{}_{\text{def}}^{\texttt{NMOS}}$ and $\mathcal{O}n^t{}_{\text{def}}^{\texttt{PMOS}}$. All other parts of semantics encoding remain the same, except for the device state predicates used in $\mathcal{R}_{\mathfrak{R}_{eq}}$—which now becomes $\mathcal{R}_{\mathfrak{R}_{eq}}^t$. We refer to this variation as $\mathcal{S}_2^q$ ($\mathcal{S}_{2\,\text{def}}^q$).

$$\bigwedge_{M \in \mathbf{D}_{\texttt{MOS}}} \left(\begin{array}{rl} & \mathcal{O}n^t(M) \Leftrightarrow \mathfrak{R}_{eq}(M) = r_{\texttt{ON}} \\ \wedge & \neg\mathcal{O}n^t(M) \Leftrightarrow \mathfrak{R}_{eq}(M) = r_{\texttt{OFF}} \end{array} \right) \qquad (\mathcal{R}_{\mathfrak{R}_{eq}}^t)$$

$$\mathcal{S}_2^q \triangleq \mathcal{R}_{\text{Ohm}} \wedge \mathcal{R}_{\text{Kirchhoff}} \wedge \mathcal{R}_{\mathfrak{R}_{eq}}^t \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}_{2\,\text{def}}^q)$$

Such modification solves the issue raised with $\mathcal{S}_1^q$, in the particular example of Section C.1): transistor states in a chain of inverters in series would be modelled more realistically. However, $\mathcal{S}_2^q$ raises a new issue, which we present in the following example.

Example and Limitation

Consider a classical CMOS-based NAND gate (Figure C.2). Applying $\mathcal{S}_2^q$ results in some physically-legitimate circuit models to become impossible (i.e., unsatisfiable with respect to circuit semantics). Table C.1 synthesizes the models obtained with $\mathcal{S}_2^q$, for the NAND circuit, with $\mathcal{V}(\mathtt{GND}) = 0\,\mathrm{V}$ and $\mathcal{V}(\mathtt{VDD}) = 1\,\mathrm{V}$. The semantics only captures 3 (out of 4) circuit states. The missed state, is the one with $\mathcal{V}(\mathtt{A}) = 1\,\mathrm{V}$ and $\mathcal{V}(\mathtt{B}) = 0\,\mathrm{V}$. It turns out some semantics rules contradict each other on this example. In particular, for transistor M3—whose gate is 1 V—the device is supposed to be switched-on. However, when such assumption is considered, the combination of Kirchhoff's current law and Ohm's law results in the voltages of nets Z and x to be very close (more precisely, less than V_t-close) to 1 V. This contradicts the fact that M3 is switched-on (see $\mathcal{O}n^{t\,\mathtt{NMOS}}_{\mathrm{def}}$). On the other hand, if M3 is hypothetically considered to be switched-off, this will lead to its source net (x) to be pulled-down further to the ground (0 V). Yet the conditions for M3 to be switched-on are met, which again contradicts the hypothesis. There are, therefore, no assignments of the variables of the formula, for these specific input (A and B) values, that satisfy the formula (the formula is unsatisfiable).

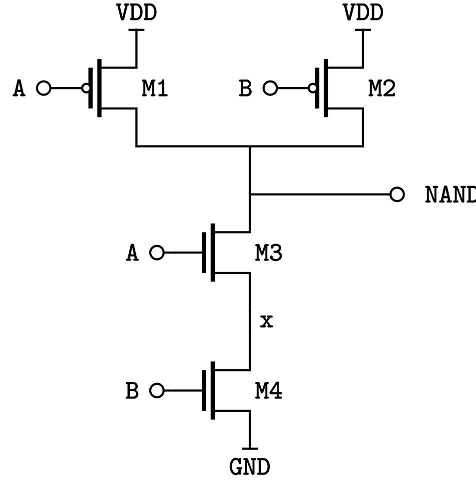


Figure C.2: (Reminder) CMOS-based NAND gate.

$\mathcal{V}(\mathtt{A})$	$\mathcal{V}(\mathtt{B})$	$\mathcal{V}(\mathtt{x})$	$\mathcal{V}(\mathtt{NAND})$	$\mathcal{O}n(\mathtt{M1})$	$\mathcal{O}n(\mathtt{M2})$	$\mathcal{O}n(\mathtt{M3})$	$\mathcal{O}n(\mathtt{M4})$	
1	1	0.000020	0.000040	$\perp$	$\perp$	$\top$	$\top$	
0	1	0.000010	0.999990	$\top$	$\perp$	$\perp$	$\top$	
0	0	0.499999	0.999998	$\top$	$\top$	$\perp$	$\perp$	
1	0	~ 1	~ 1	$\perp$	$\top$	$\top$	$\perp$	$\leftarrow$ missed with $\mathcal{S}_2^q$

Table C.1: NAND gate obtained models with $\mathcal{S}_2^q$.

To illustrate this, we consider a portion of the NAND gate, with only transistors M3 and M4 (Figure C.3). We replicate the conditions on inputs A and B for which the circuit is unsatisfiable with $\mathcal{S}_2^q$. We set assume $\mathcal{O}n^t(\mathtt{M3}) = \top$ and $\mathcal{O}n^t(\mathtt{M4}) = \perp$. We hence assign resistance $r_{\mathtt{ON}}$ to $\mathfrak{R}_{eq}(\mathtt{M3})$ and $r_{\mathtt{OFF}}$ to $\mathfrak{R}_{eq}(\mathtt{M4})$. The threshold voltage for both devices is $V_t = 200\,\mathrm{mV}$. For a simplified notation, the following shortcuts are used:

$$v_a = \mathcal{V}(\mathtt{A}) = 1\,\mathrm{V}$$
$$v_b = \mathcal{V}(\mathtt{B}) = 0\,\mathrm{V}$$
$$v_z = \mathcal{V}(\mathtt{Z})$$
$$v_x = \mathcal{V}(\mathtt{x})$$
$$v_g = \mathcal{V}(\mathtt{gnd}) = 0\,\mathrm{V}$$

$$r_3 = \mathfrak{R}_{eq}(\mathtt{M3}) = r_{\mathtt{ON}} \qquad i_3 = \mathcal{I}(\mathtt{Z}, \mathtt{M3}, \mathtt{x})$$
$$r_4 = \mathfrak{R}_{eq}(\mathtt{M4}) = r_{\mathtt{OFF}} \qquad i_4 = \mathcal{I}(\mathtt{x}, \mathtt{M4}, \mathtt{gnd})$$

Current i_3 is equal to i_4 (Kirchhoff's current law). By applying Ohm's law, we obtain the current-voltage relation on each device. We consider v_z to be set to some value in $]0\,\mathrm{V}, 1\,\mathrm{V}[$, and we

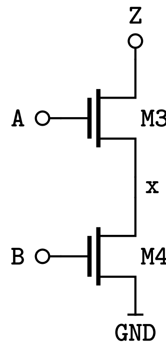


Figure C.3: Truncated pull-down network of the NAND gate (Figure C.2).

plot the relation between current and voltage of net `x` for both devices `M3` and `M4`. This is shown in Figure C.4. A similar reasoning can be done by considering `x` to be set and by varying `z`.

Note. The proportions of $r_{\text{ON}}/r_{\text{OFF}}$ of Figure C.4 have been changed, on purpose, for a better visualization. If the actual values were used, the green line (representing the current-voltage across `M3`) would visually look overlapped with the x-axis.

If we omit the constraints on device states (i.e., only considering a circuit made of resistors r_3 and r_4), there would be a solution to the system of equations (made of Ohm's rule and Kirchhoff's current law). The solution would be the intersection between the current-voltage relations of the two devices, i.e., the intersection between the blue and green lines in the figure. But since one of our assumptions is that `M3` is switched-on, i.e., $(v_a - v_z \geq V_t) \vee (v_a - v_x \geq V_t)$, an additional constraint must be taken into account: the solution must indeed correspond to a state where `M3` is switched-on. Graphically, the solution represented in Figure C.4 is not feasible, since the conditions for `M3` to be switched-on are not met (it shows $v_a - v_z < v_a - v_x < V_t$).

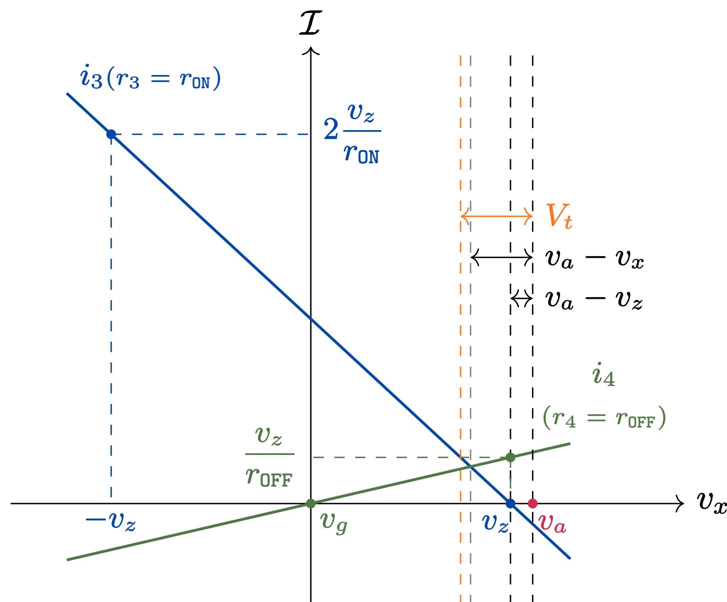


Figure C.4: Illustration of relations between current and voltage for devices `M3` and `M4` of Figure C.3, where resistance values r_{ON} and r_{OFF} are set to pre-defined values. No solution is possible.

C.3 Variable $r_{\mathtt{ON}}/r_{\mathtt{OFF}}$ ($\mathcal{S}_3^q$)

The observations from the previous example suggest that we should adjust our constraints on how equivalent $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ values are defined. The issue of the previous example, in fact, originates from having both $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ values set to pre-defined values. With lower values of $r_{\mathtt{OFF}}$ which would pull net `x` further to the ground, a solution may have been possible, if the condition for `M3` activation is met. Alternatively, using higher values for $r_{\mathtt{ON}}$, would result in a solution, if `M3` is switched-on. For example, it has been possible to find a solution with $r_{\mathtt{OFF}} = 800\,\Omega$ and $r_{\mathtt{ON}} = 100\,\Omega$, as well as with $r_{\mathtt{OFF}} = 1000\,\Omega$ and $r_{\mathtt{ON}} = 200\,\Omega$.

Figure C.5 shows how the fourth steady-state of the NAND example becomes satisfiable, by using $r_{\mathtt{OFF}}$ as interval of values between a minimum ($r_{\mathtt{OFF_{min}}}$) and maximum ($r_{\mathtt{OFF_{max}}}$) reference resistance values. The thick blue segment represents the space of solutions, where all constraints are satisfied.

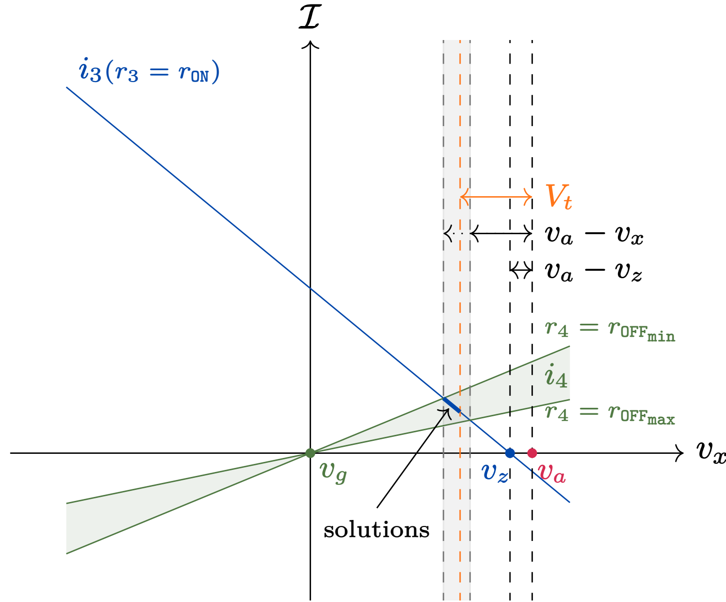


Figure C.5: Illustration of relations between current and voltage for devices `M3` and `M4` of Figure C.3, where only resistance value $r_{\mathtt{ON}}$ is set to pre-defined value, while $r_{\mathtt{OFF}}$ is variable. An interval of solutions, represented with the thick blue segment, is possible.

The space of solutions is increased when $r_{\mathtt{ON}}$ is also considered a variable (in addition to $r_{\mathtt{OFF}}$), with $r_{\mathtt{ON}} \in [r_{\mathtt{ON_{min}}}, r_{\mathtt{ON_{max}}}]$. Figure C.6 shows the solutions with the magenta-colored area.

Intuitively, the higher the degree of freedom of $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$, the less are the odds for the intersection between semantics constraints to be empty. We introduce a small variation upon the constraints illustrated in Figure C.6: we allow $r_{\mathtt{OFF}}$ to go beyond the maximum value $r_{\mathtt{OFF_{max}}}$, as well as $r_{\mathtt{ON}}$ to go below the minimum value $r_{\mathtt{ON_{min}}}$ (while remaining in $\mathbb{Q}_{>0}$). In terms of logic constraints, this is define with $\mathcal{R}^3_{r_{\mathtt{ON}}/r_{\mathtt{OFF}}}$.

$$(r_{\mathtt{OFF_{min}}} \leq r_{\mathtt{OFF}}) \wedge (0 < r_{\mathtt{ON}} \leq r_{\mathtt{ON_{max}}}) \qquad (\mathcal{R}^3_{r_{\mathtt{ON}}/r_{\mathtt{OFF}}})$$

We refer to this variation as $\mathcal{S}_3^q$ ($\mathcal{S}^q_{3\,\mathrm{def}}$), whose semantics rules (apart from $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ definitions) remain identical to $\mathcal{S}_2^q$. This choice solves the issue identified with $\mathcal{S}_2^q$ in the NAND gate example of Section C.2. However, it is not guaranteed to be correct on all circuits, because we still define $r_{\mathtt{ON_{max}}}$ and $r_{\mathtt{OFF_{min}}}$ arbitrarily and without considering resistance values in the interval $]r_{\mathtt{ON_{max}}}, r_{\mathtt{OFF_{min}}}[$. The transition from ‘on’ to ‘off’ must also be modeled for the modeling to be correct. The issue is, in fact, not solved by increasing the degree of freedom of semantics variables. Rather, the intersection between semantics rules must not be empty (i.e., it must be ensured that a solution exists). We investigate the possibility of such modeling in the next section.

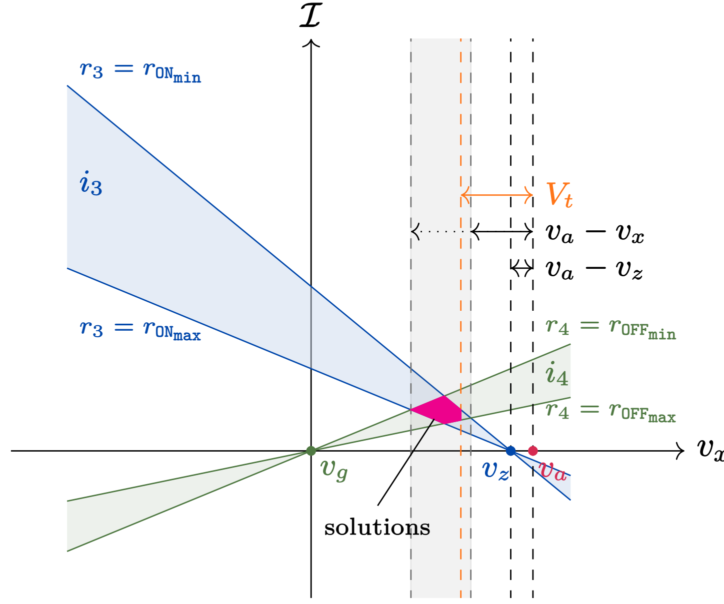


Figure C.6: Illustration of relations between current and voltage for devices `M3` and `M4` of Figure C.3, where both resistance values $r_{\mathtt{ON}}$ and $r_{\mathtt{OFF}}$ are variable. The magenta-colored area represents the possible solutions.

$$\mathcal{S}_3^q \triangleq \mathcal{R}_{\text{Ohm}} \wedge \mathcal{R}_{\text{Kirchhoff}} \wedge \mathcal{R}^t_{\mathfrak{R}_{eq}} \wedge \mathcal{R}^3_{r_{\mathtt{ON}}/r_{\mathtt{OFF}}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \qquad (\mathcal{S}^q_{3\,\text{def}})$$

C.4 A Three Domains Based Modeling ($\mathcal{S}_4^q$)

We now try to model transistors' equivalent resistance with relations. We consider a conservative approach where resistance may range over all non-negative positive rational numbers (theoretically, from 0-excluded, up to infinity). For this, we consider the same two reference resistance values $r_{\mathtt{ON_{max}}}$ and $r_{\mathtt{OFF_{min}}}$.

Device states (on/off) are defined with the same rules $\mathcal{O}n^{t\,\mathtt{NMOS}}_{\text{def}}$ and $\mathcal{O}n^{t\,\mathtt{PMOS}}_{\text{def}}$. In addition, we define predicates for transistor regions of operation (cut-off, linear and saturation [Ytterdal et al., 2003]). The regions of operation give more details on how current relates to voltage (I-V relations) than a simple on/off abstraction. The criterion for determining the region of operation of a device depends both on the voltage applied ($\mathcal{V}_{\text{GS}}$ or $\mathcal{V}_{\text{GD}}$) and the absolute value of the source-to-drain voltage ($\mathcal{V}_{\text{DS}}$). When the transistor is switched-on, and in addition it is applied a *high* (we define such condition in $E_{\mathcal{H}igh_{|\mathcal{V}_{\text{DS}}|}}$) voltage difference between its source and drain, it is said to be in the saturation mode ($E_{\mathcal{S}at}$). Otherwise, when switched-on without having such high source-to-drain voltage difference, it is in the linear mode ($E_{\mathcal{L}in}$). The cut-off region simply corresponds to the device being switched-off ($E_{\mathcal{C}ut}$).

$$\begin{aligned}
&\mathcal{H}igh_{|\mathcal{V}_{\text{DS}}|} : \mathbf{D} \longrightarrow \mathbb{B} \\
&M \longmapsto \begin{cases} \left(\begin{array}{ll} & |\mathcal{V}_{\text{DS}}(M)| > \mathcal{V}_{\text{GS}}(M) - |V_t| \\ \vee & |\mathcal{V}_{\text{DS}}(M)| > \mathcal{V}_{\text{GD}}(M) - |V_t| \end{array} \right), & \text{if } M \in \mathbf{D}_{\mathtt{NMOS}} \\[2ex] \left(\begin{array}{ll} & |\mathcal{V}_{\text{DS}}(M)| < \mathcal{V}_{\text{GS}}(M) + |V_t| \\ \vee & |\mathcal{V}_{\text{DS}}(M)| < \mathcal{V}_{\text{GD}}(M) + |V_t| \end{array} \right), & \text{if } M \in \mathbf{D}_{\mathtt{PMOS}} \end{cases}
\end{aligned} \qquad (E_{\mathcal{H}igh_{|\mathcal{V}_{\text{DS}}|}})$$

$$\begin{aligned} \mathcal{C}ut\colon \mathbf{D} &\longrightarrow \mathbb{B} \\ M &\longmapsto \neg\mathcal{O}n(M) \end{aligned} \tag{$E_{\mathcal{C}ut}$}$$

$$\begin{aligned} \mathcal{L}in\colon \mathbf{D} &\longrightarrow \mathbb{B} \\ M &\longmapsto \mathcal{O}n(M) \wedge \neg\mathcal{H}igh_{|\mathcal{V}_{\text{DS}}|}(M) \end{aligned} \tag{$E_{\mathcal{L}in}$}$$

$$\begin{aligned} \mathcal{S}at\colon \mathbf{D} &\longrightarrow \mathbb{B} \\ M &\longmapsto \mathcal{O}n(M) \wedge \mathcal{H}igh_{|\mathcal{V}_{\text{DS}}|}(M) \end{aligned} \tag{$E_{\mathcal{S}at}$}$$

Rule $\mathcal{R}_{regions}$ defines the relation between transistor regions predicates.

$$\bigwedge_{M \in \mathbf{D}} \mathcal{C}ut(M) \vee \mathcal{L}in(M) \vee \mathcal{S}at(M) \tag{$\mathcal{R}_{regions}$}$$

The equivalent resistance value is determined, for each transistor, based on whether it is switched-on or switched-off, taking into account an intermediate state where the transition from 'off' to 'on' happens. This is illustrated in Figure C.7. Formally, the corresponding constraints are shown in $\mathcal{R}_{\text{R}}$.

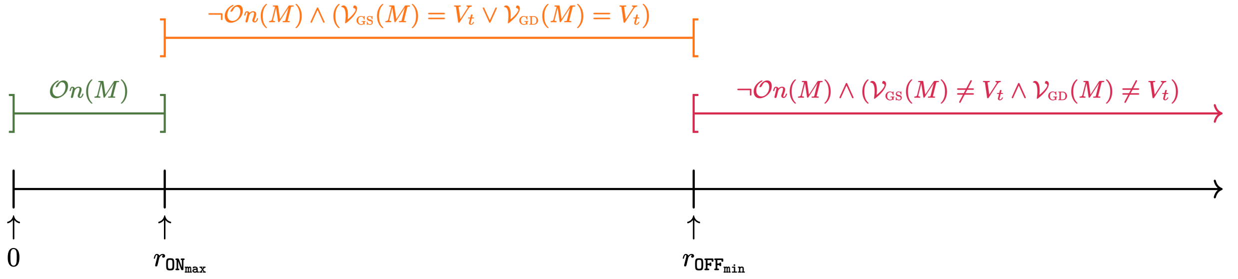


Figure C.7: Equivalent resistance $\mathfrak{R}_{eq}(M)$ of a device M with respect to its applied voltage ($\mathcal{V}_{\text{GS}}(M)$ and $\mathcal{V}_{\text{GD}}(M)$), in $\mathcal{S}_4^q$.

$$\bigwedge_{M \in \mathbf{D}} \left(\begin{array}{ll} & \mathcal{O}n(M) \Leftrightarrow 0 < \mathfrak{R}_{eq}(M) \leq r_{\texttt{ON}_{\texttt{max}}} \\ \wedge & \neg\mathcal{O}n(M) \wedge (\mathcal{V}_{\text{GS}}(M) \neq V_t \wedge \mathcal{V}_{\text{GD}}(M) \neq V_t) \Leftrightarrow \mathfrak{R}_{eq}(M) \geq r_{\texttt{OFF}_{\texttt{min}}} \\ \wedge & \neg\mathcal{O}n(M) \wedge (\mathcal{V}_{\text{GS}}(M) = V_t \vee \mathcal{V}_{\text{GD}}(M) = V_t) \Leftrightarrow r_{\texttt{ON}_{\texttt{max}}} < \mathfrak{R}_{eq}(M) < r_{\texttt{OFF}_{\texttt{min}}} \end{array} \right) \tag{$\mathcal{R}_{\text{R}}$}$$

At this stage, still Kirchhoff's current law and Ohm's law are needed to be added as constraints (the same rules $\mathcal{R}_{\text{Kirchhoff}}$ and $\mathcal{R}_{\text{Ohm}}$ previously defined). Circuit semantics $\mathcal{S}_4^q$ is hence defined with $\mathcal{S}_{4\,\text{def}}^q$.

$$\mathcal{S}_4^q \triangleq \mathcal{R}_{regions} \wedge \mathcal{R}_{\text{Ohm}} \wedge \mathcal{R}_{\text{Kirchhoff}} \wedge \mathcal{R}_{\text{R}} \wedge \mathcal{R}_{\mathbf{S}} \wedge \mathcal{R}_{\mathbf{I}} \tag{$\mathcal{S}_{4\,\text{def}}^q$}$$

This modeling solves the issues raised by previous modeling. However, the solving time quickly gets larger even on small circuits. Such scaling limitation is due to the fact that circuit semantics are now harder to solver, as they involve multiplications of variables (namely, resistance values and currents in Ohm's law, $\mathcal{R}_{\text{Ohm}}$). Such encoding requires enabling the theory of non-linear real arithmetic (NRA). We hence abandon this direction, and explore a new modeling strategy. Nevertheless, $\mathcal{S}_4^q$ was the intermediate step which inspired the final version of quantitative semantics $\mathcal{S}^q$—which we also denote $\mathcal{S}_5^q$ in the following section.

C.5 Hyperplanes Based Approximation of Device Characteristics ($\mathcal{S}_5^q \triangleq \mathcal{S}^q$)

The behavior of a transistor depends on its region of operation. To drop non-linear constraints from previous modelings, we opt for a definition of circuit behavior as the union of approximations of I-V relations. The constraints are defined as inequalities involving semantics variables (i.e., voltage and current), where no variables are multiplied together.

We use such variant of circuit modeling as our main quantitative semantics, and we extend it to resistors and diodes. The intuition behind it, as well as its related formalism, are detailed in the body of this thesis—Section 4.6 (page 63).

Publications

Conference Papers

Ferres, B., Oulkaid, O., Henrio, L., Khosravian Ghadikolaei, M., Moy, M., Radanne, G., and Raymond, P. (2023). Electrical Rule Checking of Integrated Circuits using Satisfiability Modulo Theory. In *https://ieeexplore.ieee.org/xpl/conhome/9774496/proceeding*, pages 1–2, Anvers (Antwerpen), Belgium. IEEE. (cited pages 3, 119, and 125).

Oulkaid, O., Ferres, B., Moy, M., Raymond, P., Khosravian, M., Henrio, L., and Radanne, G. (2024). A Transistor Level Relational Semantics for Electrical Rule Checking by SMT Solving. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE. (cited pages 3, 3, 119, 120, 125, and 125).

Journal Articles

Ferres, B., Oulkaid, O., Moy, M., Radanne, G., Henrio, L., Raymond, P., and G., M. K. (2025). A Survey on Transistor-Level Electrical Rule Checking of Integrated Circuits. *ACM Transactions on Design Automation of Electronic Systems.* (cited pages 3, 17, 119, and 124).

Oulkaid, O., Ferres, B., Moy, M., Raymond, P., and Khosravian, M. (2025). Modeling Techniques for the Formal Verification of Integrated Circuits at Transistor-Level: Performance vs. Precision Trade-offs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* (cited pages 3, 3, 120, 125, and 125).

Submitted Work

Oulkaid, O., Moy, M., Ferres, B., Khosravian, M., and Raymond, P. (n.d.). Time-Dependent Dielectric Breakdown Worst-Steady-State Analysis of Integrated Circuits using Optimization Modulo Theories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* [Submitted]. (cited pages 3 and 126).

Bibliography

Abrahams, M. and Barkley, J. (1998). RTL verification strategies. In *Wescon/98. Conference Proceedings (Cat. No. 98CH36265)*, pages 130–134. IEEE. (cited page 19).

Abu-Haeyeh, Y. and Hedrich, L. (2024). Formal Verification of Nonlinear Analog Circuits using State Space-Based Model Order Reduction. In *2024 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 1–4. IEEE. (cited page 19).

Abu-Khater, I. S., Bellaouar, A., and Elmasry, M. I. (2002). Circuit techniques for CMOS low-power high-performance multipliers. *IEEE Journal of solid-state circuits*, 31(10):1535–1546. (cited page 18).

Afonso, J. and Monteiro, J. (2017a). Analysis of short-circuit conditions in logic circuits. In *DATE*, pages 824–829, Lausanne, Switzerland. IEEE. (cited pages 2, 35, 36, 36, 36, 37, 38, 39, 124, and 125).

Afonso, J. and Monteiro, J. (2017b). Analysis of short-circuit conditions in logic circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 824–829. IEEE. (cited pages 16, 37, 86, and 86).

Akers (1978). Binary decision diagrams. *IEEE Transactions on computers*, 100(6):509–516. (cited page 8).

Alam, N. and Alam, M. (2020). The trend of different parameters for designing integrated circuits from 1973 to 2019 and linked to Moores law. *Aust. J. Eng. Innov. Technol*, 2(2):16–23. (cited pages 1 and 123).

Alpern, B. and Schneider, F. B. (1985). Defining liveness. *Information processing letters*, 21(4):181–185. (cited page 7).

Alwi, S. H. S. and Encrenaz, E. (2014). FSM-based properties and abstraction of components. In *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pages 37–43. IEEE. (cited page 8).

Analog Devices (2024). LTspice. `https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html`. [Accessed 15-07-2024]. (cited pages 59 and 109).

Aniah (2025). `https://aniah.fr/`. [Accessed 28-07-2025]. (cited pages 3, 120, and 123).

ANSI/ESDA/JEDEC (2012). User Guide of ANSI/ESDA/JEDEC JS-001 Human Body Model Testing of Integrated Circuits. *Electronic Discharge Association and JEDEC Solid State Technology Association, Rome NY and Arlington VA*. (cited page 28).

ANSI/ESDA/JEDEC (2022). ANSI/ESDA/JEDEC JS-002 For Electrostatic Discharge Sensitivity Testing - Charged Device Model (CDM) Device Level. *Electronic Discharge Association and JEDEC Solid State Technology Association, Rome NY and Arlington VA*. (cited page 28).

Art Schaldenbrand (Cadence) (2024). Analog Reliability Analysis for Mission-Critical Applications. https://www.cadence.com/en_US/home/resources/white-papers/analog-reliability-analysis-for-mission-critical-applications-wp.html. [Accessed 30-05-2024]. (cited page 100).

Ashar, P., Bhattacharya, S., Raghunathan, A., and Mukaiyama, A. (1998). Verification of RTL generated from scheduled behavior in a high-level synthesis flow. In *1998 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (IEEE Cat. No.98CB36287)*, pages 517–524. (cited pages 1 and 123).

Augustin, L., Gennart, B., Huh, Y., Luckham, D., and Stanculescu, A. (1988). Verification of VHDL designs using VAL. In *25th ACM/IEEE, Design Automation Conference. Proceedings 1988.*, pages 48–53. IEEE. (cited page 19).

Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J., and Asanović, K. (2012). Chisel: Constructing Hardware in a Scala Embedded Language. In *Proceedings of the 49th annual design automation conference*, pages 1216–1225. (cited page 23).

Backus, J. (1978). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641. (cited page 23).

Backus, J. (1981). The algebra of functional programs: function level reasoning, linear equations, and extended definitions. In *International Colloquium on the Formalization of Programming Concepts*, pages 1–43. Springer. (cited page 23).

Bank, R., Coughran, W., Fichtner, W., Grosse, E., Rose, D., and Smith, R. (1985). Transient Simulation of Silicon Devices and Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(4):436–451. (cited page 22).

Barrett, C., Stump, A., Tinelli, C., et al. (2010). The SMT-LIB Standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, page 14. (cited page 11).

Biere, A. (2021). Bounded model checking. In *Handbook of satisfiability*, pages 739–764. IOS press. (cited page 8).

Bjørner, N., Phan, A.-D., and Fleckenstein, L. (2015). νz-an optimizing smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 194–199. Springer. (cited page 14).

Bjørner, N. S. and Phan, A.-D. (2014). νZ-Maximal Satisfaction with Z3. *Scss*, 30:1–9. (cited pages 3, 95, 101, 124, and 125).

Blanchet, B. (2002). Introduction to abstract interpretation. *lecture script.* (cited page 10).

Blieck, S. and Janssens, E. (1996). Software Check for Power-down Mode of Analog Circuits. page 4. (cited pages 33, 34, 37, and 39).

Börger, E., Grädel, E., and Gurevich, Y. (2001). *The classical decision problem.* Springer Science & Business Media. (cited page 11).

Brum, N. (2024). PyLTSpice. https://pypi.org/project/PyLTSpice/. [Accessed 15-07-2024]. (cited page 109).

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691. (cited page 8).

Burger, D. and Goodman, J. R. (1997). Billion-transistor architectures. *Computer*, 30(09):46–49. (cited pages 1 and 123).

Burghardt, J. (2013). Abstract interpretation of integers by signs. `https://upload.wikimedia.org/wikipedia/commons/8/8b/Abstract_interpretation_of_integers_by_signs_svg.svg`. [Accessed 14-05-2025]. (cited pages xvii and 10).

Butler, R. W. (2001). Langley Formal Methods Program; What is Formal Methods — shemesh.larc.nasa.gov. `https://shemesh.larc.nasa.gov/fm/fm-what.html`. [Accessed 02-05-2025]. (cited page 5).

Cataldo, E., Di Lieto, A., Maccarrone, F., and Paffuti, G. (2016). Measurements and analysis of current-voltage characteristic of a pn diode for an undergraduate physics laboratory. *preprint arXiv:1608.05638*. (cited page 70).

Cimatti, A., Griggio, A., Irfan, A., Roveri, M., and Sebastiani, R. (2018). Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Transactions on Computational Logic (TOCL)*, 19(3):1–52. (cited page 113).

Clarke, E., Biere, A., Raimi, R., and Zhu, Y. (2001). Bounded model checking using satisfiability solving. *Formal methods in system design*, 19:7–34. (cited page 8).

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided Abstraction Refinement. In *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*, pages 154–169. Springer. (cited page 8 and 8).

Clarke, E. M., Emerson, E. A., and Sifakis, J. (2009). Model checking: algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84. (cited page 5).

Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542. (cited page 6).

Clarke, E. M., Klieber, W., Nováček, M., and Zuliani, P. (2011). Model checking and the state explosion problem. In *LASER Summer School on Software Engineering*, pages 1–30. Springer. (cited page 5).

Cousot, P. and Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. (cited page 9).

Cousot, P., Cousot, R., Feret, J., Miné, A., and Rival, X. (2001). The Astrée Static Analyzer. `https://www.astree.ens.fr`. [Accessed 28-05-2025]. (cited page 11).

Cuoq, P. and Prevosto, V. (2009). Frama-Cs value analysis plug-in. (cited page 11).

Dai, Y., Yang, Y., Jiang, N., Qi, P., Chen, Q., and Tong, J. (2022). A high performance and low power triple-node-upset self-recoverable latch design. *Electronics*, 11(21):3606. (cited page 23).

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397. (cited page 12).

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215. (cited page 12).

De, V. and Borkar, S. (1999). Technology and design challenges for low power and high performance. In *Proceedings of the 1999 international symposium on Low power electronics and design*, pages 163–168. (cited page 18).

De Moura, L. and Bjørner, N. (2008). Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer. (cited pages 3, 12, 13, 16, 62, 84, 119, and 124).

Dominik, C. and Drechsler, R. (2024). Polynomial formal verification of sequential circuits. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE. (cited page 19).

Drechsler, R. (2021). PolyAdd: Polynomial formal verification of adder circuits. In *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 99–104. IEEE. (cited page 19).

Drechsler, R., Mahzoon, A., and Goli, M. (2022a). Towards polynomial formal verification of complex arithmetic circuits. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 1–6. IEEE. (cited page 19).

Drechsler, R., Mahzoon, A., and Weingarten, L. (2022b). Polynomial formal verification of arithmetic circuits. In *Proceedings of International Conference on Computational Intelligence and Data Engineering: ICCIDE 2021*, pages 457–470. Springer. (cited page 19).

Fish, A., Milrud, V., and Yadid-Pecht, O. (2005). High-speed and high-precision current winner-take-all circuit. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(3):131–135. (cited page 28).

Foster, H. D. (2024). 2024 wilson research group ic/asic functional verification trend report. `https://static.sw.cdn.siemens.com/siemens-disw-assets/public/1NyfMZJ3nuPhMGQCGFDITG/en-US/ic-asic-functional-verification-trend-report-wp-86424-d3.pdf`. [Accessed 31-07-2025]. (cited pages 1, 1, 123, and 123).

Fourier, J. (1827). Histoire de lacadémie, partie mathématique (1824). *Mémoires de lAcadémie des sciences de lInstitut de France*, 7:38. (cited page 42).

Gao, S., Kong, S., and Clarke, E. M. (2013). dReal: An SMT solver for nonlinear theories over the reals. In *International conference on automated deduction*, pages 208–214. Springer. (cited page 101).

Gevinti, E., Cerati, L., Di Biccari, L., Ballarin, G., Andreini, A., Fragnoli, M., and Bogani, A. (2015). Schematic-Level and Layout-Level ESD EDA check methodology applied to smart power IC's - initialization and implementation. In *2015 37th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, pages 1–10, Reno, NV, USA. IEEE. (cited page 32).

Ghetti, A. (2004). Gate oxide reliability: Physical and computational models. In *Predictive simulation of semiconductor processing: status and challenges*, pages 201–258. Springer. (cited page 98).

Gibson, P., Ziyang Lu, Pikus, F., and Srinivasan, S. (2010). A Framework for Logic-Aware Layout Analysis. In *2010 11th International Symposium on Quality Electronic Design (ISQED)*, pages 171–175, San Jose, CA, USA. IEEE. (cited pages 1, 31, and 123).

Graf, S. and Loiseaux, C. (1993). A tool for symbolic program verification and abstraction. In *Computer Aided Verification: 5th International Conference, CAV'93 Elounda, Greece, June 28–July 1, 1993 Proceedings 5*, pages 71–84. Springer. (cited page 6).

Grimm, T., Lettnin, D. V., and Hübner, M. (2020). A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics*, 7(6, Article 81):81–1 – 81–27. (cited page 35).

Guggenbuhl, W., Di, J., and Goette, J. (1994). Switched-current memory circuits for high-precision applications. *IEEE Journal of Solid-State Circuits*, 29(9):1108–1116. (cited page 28).

Gundala, S., Basha, M. M., and Vijayakumar, S. (2022). Level-up/level-down voltage level shifter for nano-scale applications. *Journal of Engineering Science and Technology*, 17(1):745–759. (cited page 82).

Gupta, A. (1992). Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1(2):151–238. (cited pages 1 and 123).

Hadarean, L., Bansal, K., Jovanović, D., Barrett, C., and Tinelli, C. (2014). A tale of two solvers: Eager and lazy approaches to bit-vectors. In *International Conference on Computer Aided Verification*, pages 680–695. Springer. (cited page 13).

Hałgas, S. (2023). A spice-oriented method for finding multiple dc solutions in nonlinear circuits. *Applied Sciences*, 13(4):2369. (cited page 53).

Hany, S. and Hogan, M. (2022). Beyond geometry checks: Context-aware design verification. Technical report, Siemens EDA. (cited page 32).

Hasler, P. and Lande, T. S. (2001). Overview of floating-gate devices, circuits, and systems. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 48(1):1–3. (cited page 25).

Hasler, P., Minch, B. A., and Diorio, C. (1999). Floating-gate devices: they are not just for digital memories any more. In *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 388–391. IEEE. (cited page 25).

Herbert, S. and Marculescu, D. (2007). Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 international symposium on Low power electronics and design*, pages 38–43. (cited page 18).

Hilbert, D. and Ackermann, W. (1928). *Grundzüge der Theoretischen Logik*. Springer, Berlin, Germany. (cited page 11).

Hogan, M., Srinivasan, S., Medhat, D., Lu, Z., and Hofmann, M. (2013). Using static voltage analysis and voltage-aware DRC to identify EOS and oxide breakdown reliability issues. page 6. (cited pages 31, 33, 34, and 39).

Hsieh, Y.-W. and Levitan, S. P. (1998). Model abstraction for formal verification. In *Proceedings Design, Automation and Test in Europe*, pages 140–147. IEEE. (cited page 6).

Irfan, A., Cimatti, A., Griggio, A., Roveri, M., and Sebastiani, R. (2016). Verilog2SMV: A Tool for Word-Level Verification. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1156–1159. (cited pages 1 and 123).

JEDEC Publication JEP122H (2016). Failure Mechanisms and Models for Semiconductor Devices. (cited page 98).

Jia, F., Han, R., Huang, P., Liu, M., Ma, F., and Zhang, J. (2023). Improving bit-blasting for non-linear integer constraints. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 14–25. (cited page 12).

Jian, H. and Xubang, S. (2008). The design methodology and practice of low power SoC. In *2008 International Conference on Embedded Software and Systems Symposia*, pages 185–190. IEEE. (cited page 18).

Junttila, T. (2020). CS-E3220: SMT - Satisfiability Modulo Theories documentation. `https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-smt/index.html`. [Accessed 12-05-2025]. (cited page 13).

Kalel, D. (2024). *Advanced Structural and Semi-Formal Verification Flow for Clock Domain Crossing (CDC) in Asynchronous Multiclock Systems*. PhD thesis, Université Grenoble Alpes. (cited page 28).

Kaschani, KT (2015). What is Electrical Overstress? - Analysis and Conclusions. 55:853–862. (cited page 25).

Keim, M., Drechsler, R., Becker, B., Martin, M., and Molitor, P. (2003). Polynomial formal verification of multipliers. *Formal Methods in System Design*, 22:39–58. (cited page 19).

Khazhinsky, M. G., Cao, S., Gossner, H., Boselli, G., and Etherton, M. (2012). Electronic Design Automation (EDA) Solutions for ESD-Robust Design and Verification. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–8. IEEE. (cited page 28).

Khorasani, A. E., Griswold, M., and Alford, T. (2013). A Fast $I\{-\}V$ Screening Measurement for TDDB Assessment of Ultra-Thick Inter-Metal Dielectrics. *IEEE electron device letters*, 35(1):117–119. (cited page 42).

Ko, P. (1989). Circuit reliability simulator-oxide breakdown module. In *International Technical Digest on Electron Devices Meeting*, pages 331–334. IEEE. (cited page 100).

Ko, U., Balsara, T., and Lee, W. (1995). Low-power design techniques for high-performance CMOS adders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2):327–333. (cited page 18).

Kollu, K., Jackson, T., Kharas, F., and Adke, A. (2012). Unifying design data during verification: Implementing Logic-Driven Layout analysis and debug. In *2012 IEEE International Conference on IC Design & Technology*, pages 1–5, Austin, TX, USA. IEEE. (cited pages 1, 31, and 123).

Koo, K.-H., Seo, J.-H., Ko, M.-L., and Kim, J.-W. (2005). A New Level-Up Shifter for High Speed and Wide Range Interface in Ultra Deep Sub-Micron. In *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1063–1065. (cited page 82).

Kroening, D., Ouaknine, J., Strichman, O., Wahl, T., and Worrell, J. (2011). Linear completeness thresholds for bounded model checking. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 557–572. Springer. (cited page 9).

Kroening, D. and Strichman, O. (2008). *Decision procedures*, volume 5. Springer. (cited page 11).

Kundert, K. S. (1995). *DC Analysis*, pages 15–50. Springer, Boston, MA. (cited page 53).

Kunz, H., Boselli, G., Brodsky, J., Hambardzumyan, M., and Eatmon, R. (2010). An Automated ESD Verification Tool for Analog Design. In *Electrical Overstress/Electrostatic Discharge Symposium Proceedings 2010.* (cited pages 32 and 39).

Lamport, L. (1977). Proving the correctness of multiprocess programs. *IEEE transactions on software engineering*, (2):125–143. (cited page 7).

Lamport, L. (1994). The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923. (cited page 6).

Lancaster, D. and Berlin, H. M. (1988). *CMOS cookbook.* Elsevier. (cited page 21).

Lang, L. (2010). Case Study: Power-aware IP and Mixed-Signal Verification. *DVCON (San Jose, Calif).* (cited pages 22, 30, and 100).

Le, H. M., Herdt, V., Große, D., and Drechsler, R. (2016). Towards formal verification of real-world SystemC TLM peripheral models-a case study. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1160–1163. IEEE. (cited page 19).

Le Sueur, E. and Heiser, G. (2010). Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8. (cited page 18).

Lee, C.-Y. (1959). Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999. (cited page 8).

Lee, P., Kuo, M., Ko, P., and Hu, C. (1990). BERT-Circuit aging simulator (CAS). *UC Berkeley Memorandum UCB/ERL M90/2.* (cited page 100).

Leroy, X., Blazy, S., Kästner, D., Schommer, B., Pister, M., and Ferdinand, C. (2016). CompCert-a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems, 8th European Congress.* (cited page 11).

Lescot, J., Bligny, V., Medhat, D., Chollat-Namy, D., Lu, Z., Billy, S., and Hofmann, M. (2012). Static low power verification at transistor level for SoC design. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design - ISLPED '12*, page 129, Redondo Beach, California, USA. ACM Press. (cited pages 2, 30, 31, 31, 32, 32, 32, and 39).

Lescot, J., Dehan, P., Boujarra, W., Medhat, D., and Billy, S. (2015). A Comprehensive ESD Verification Flow at Transistor Level For large Soc Designs. In *2015 37th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, Reno, NV, USA. IEEE. (cited pages 1, 32, 37, 39, and 123).

Liu, C.-H., Liu, H.-Y., Lin, C.-W., Chou, S.-J., Chang, Y.-W., Kuo, S.-Y., Yuan, S.-Y., and Chen, Y.-W. (2008). An Efficient Graph-Based Algorithm for ESD Current Path Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, (8). (cited pages 28, 31, and 39).

LTwiki (2024). M. MOSFET — ltwiki.org. `https://ltwiki.org/LTspiceHelp/LTspiceHelp/M_MOSFET.htm`. [Accessed 09-10-2024]. (cited page 67).

Lu, Z. and Bell, D. A. (2010). Hierarchical verification of chip-level ESD design rules. page 6. (cited pages 28, 31, 32, and 39).

Marques-Silva, J. P. and Sakallah, K. A. (2002). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on computers*, 48(5):506–521. (cited page 12).

Mead, C. and Conway, L. (1980). Introduction to VLSI systems. (cited page 17).

Miller, M. and Brewer, F. (2013). Formal verification of analog circuit parameters across variation utilizing SAT. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1442–1447. IEEE. (cited pages 2, 37, 38, 38, 39, 65, 65, 93, and 124).

Mongelli, G., Faehn, E., Robins, D., Girard, P., and Virazel, A. (2024). A Fast and Efficient Graph-Based Methodology for Cell-Aware Model Generation. In *2024 IEEE International Test Conference (ITC)*, pages 270–279. IEEE. (cited page 19).

Mongelli, G., Faehn, E., Robins, D., Girard, P., and Virazel, A. (2025). Accelerating Cell-Aware Model Generation for Sequential Cells using Graph Theory. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE. (cited page 19).

Moore, G. E. (1965). Cramming more components onto integrated circuits. (cited page 1).

Nagel, L. and Pederson, D. O. (1973). SPICE (Simulation Program with Integrated Circuit Emphasis). (cited page 22).

Neuner, M. and Graeb, H. (2019). Power-Down Mode Verification for Hierarchical Analog Circuits. In *2019 16th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 125–128, Lausanne, Switzerland. IEEE. (cited pages 34, 35, 37, and 39).

Neuner, M. and Graeb, H. (2020a). Hierarchical Analog Power-Down Synthesis. In *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, Glasgow, UK. IEEE. (cited page 37).

Neuner, M. and Graeb, H. (2020b). Verification and revision of the power-down mode for hierarchical analog circuits. *Integration*, 73:1–9. (cited pages 32, 32, 34, 35, 35, 35, and 39).

Newcomb, J. C. (2012a). Circuit states. `https://patents.google.com/patent/US8225251B2`. (cited pages 36, 37, 38, and 39).

Newcomb, J. C. (2012b). Method of predicting electronic circuit floating gates. `https://patents.google.com/patent/US20120110528A1`. (cited pages 36, 38, and 39).

Nichols, K., Kazmierski, T., Zwolinski, M., and Brown, A. (1994). Overview of SPICE-like circuit simulation algorithms. *IEE Proceedings-Circuits, Devices and Systems*, 141(4):242–250. (cited pages 22, 30, and 100).

Ohlrich, M., Ebeling, C., Ginting, E., and Sather, L. (1993). Subgemini: Identifying Subcircuits using a Fast Subgraph Isomorphism Algorithm. In *Proceedings of the 30th International Design Automation Conference*, pages 31–37. (cited page 32).

Oliveira, M. F., Kuznik, C., Le, H. M., Große, D., Haedicke, F., Mueller, W., Drechsler, R., Ecker, W., and Esen, V. (2012). The system verification methodology for advanced TLM verification. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 313–322. (cited page 19).

Patel, T. (2014). Comparison of Level 1, 2 and 3 MOSFETs. *University of Texas at Arlington, December.* (cited pages 67 and 109).

Pelz, G. and Roettcher, U. (1994). Pattern Matching and Refinement Hybrid Approach to Circuit Comparison. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):264–276. (cited page 32).

Plassan, G., Peter, H.-J., Morin-Allory, K., Rahim, F., Sarwary, S., and Borrione, D. (2016). Conclusively verifying clock-domain crossings in very large hardware designs. In *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE. (cited page 28).

Plassan, G., Peter, H.-J., Morin-Allory, K., Sarwary, S., and Borrione, D. (2017). Improving the Efficiency of Formal Verification: The Case of Clock-Domain Crossings. In Hollstein, T., Raik, J., Kostin, S., Tertov, A., O'Connor, I., and Reis, R., editors, *VLSI-SoC: System-on-Chip in the Nanoscale Era Design, Verification and Reliability*, pages 108–129. Springer International Publishing. (cited pages 28 and 37).

Pnueli, A. (1977). The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pages 46–57. ieee. (cited page 6).

Quarles, T., Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. (1994). SPICE 3 Version 3F3 User's Manual. (cited page 41).

Randal E Bryant (1984). A Switch-Level Model and Simulator for MOS Digital Systems. *IEEE Transactions on Computers*, C-33(2):160–177. (cited pages 21, 23, and 58).

Ray, A., Devlin, B., Quah, F. Y., and Yesantharao, R. (2023). Hardcaml: An ocaml hardware domain-specific language for efficient and robust design. *arXiv preprint arXiv:2312.15035*. (cited page 23).

Saha, S., Celaya, J. R., Vashchenko, V., Mahiuddin, S., and Goebel, K. F. (2011). Accelerated aging with electrical overstress and prognostics for power MOSFETs. In *IEEE 2011 EnergyTech*, pages 1–6. IEEE. (cited page 25).

Sansen, W. M. (2007). *Analog design essentials*, volume 859. Springer Science & Business Media. (cited page 24).

Santos, R., Afonso, J., and Monteiro, J. (2020). Short-circuit Analysis using a Parallel QBF Solver. In *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, Segovia, Spain. IEEE. (cited pages 36, 36, 37, and 39).

Schneider, F. B. (1987). Decomposing properties into safety and liveness. Technical report, Cornell University. (cited page 8).

Sedra, A. S. and Smith, K. C. (2010). *5.11 The Junction Field-Effect Transistor (JFET).* Oxford University Press, Inc. [Accessed 18-09-2024]. (cited page 62).

Semeraro, G., Magklis, G., Balasubramonian, R., Albonesi, D. H., Dwarkadas, S., and Scott, M. L. (2002). Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pages 29–40. IEEE. (cited page 18).

Semiconductor, F. (1983). CMOS, the ideal logic family. *Application Note*, (77). (cited page 21).

Sheeran, M. (1984). muFP, a language for VLSI design. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, pages 104–112. (cited page 23).

Sheeran, M. (2005). Hardware Design and Functional Programming: a Perfect Match.(2005). *URL http://www. jucs. org/jucs_11_7/hardware_design_ and_functional/jucs_11_7_1135_1158_sheeran. pdf.* (cited page 23).

Shockley, W. (1949). The Theory of p-n Junctions in Semiconductors and p-n Junction Transistors. *Bell system technical journal*, 28(3):435–489. (cited page 70).

Siemens (2025). Basic Abstraction Techniques | Formal Verification. `https://verificationacademy.com/topics/formal-verification/the-formal-101-series_learn-formal-the-easy-way/basic-abstraction-techniques/`. [Accessed 28-05-2025]. (cited page 11).

Siemens EDA (2023). Calibre PERC. `https://eda.sw.siemens.com/en-US/ic/calibre-design/reliability-verification/perc/`. [ONLINE] Last accessed on 08 july, 2024. (cited pages 31 and 32).

Smith, P. (2010). Types of proof system. `http://www.logicmatters.net/resources/pdfs/ProofSystems.pdf`. (cited page 15).

Sommerville, I. (2011). *Software Engineering, 9/E.* Pearson Education India. (cited page 5).

Soos, M., Nohl, K., and Castelluccia, C. (2009). Extending SAT solvers to cryptographic problems. In Kullmann, O., editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, Lecture Notes in Computer Science, pages 244–257. Springer. (cited page 15).

Srinivasan, J., Adve, S. V., Bose, P., and Rivers, J. A. (2004). The case for lifetime reliability-aware microprocessors. *ACM SIGARCH Computer Architecture News*, 32(2):276. (cited page 117).

Srinivasan, S., Cohen, E., and Hofmann, M. (2014). A New Approach Using Symbolic Analysis To Compute Path-Dependent Effective Properties Preserving Hierarchy. In *2014 27th IEEE International System-on-Chip Conference (SOCC)*, pages 404–408. IEEE. (cited page 31).

Synopsys (2013). HSPICE Reference Manual: MOSFET Models. (cited pages 30 and 67).

Synopsys (2024). PrimeSim Reliability Analysis. `https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-reliability-analysis.html`. [Accessed 30-05-2024]. (cited page 100).

Tarraf, A. and Hedrich, L. (2019). Behavioral modeling of transistor-level circuits using automatic abstraction to hybrid automata. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1451–1456. IEEE. (cited page 19).

Toshiba Electronic Devices & Storage Corporation (2018). Power MOSFET Maximum Ratings. `https://toshiba.semicon-storage.com/info/application_note_en_20180726_AKX00062.pdf?did=13414`. (cited page 87).

Trivedi, N. and Alvarez, D. (2015). Essential - Integration of ESD Verification Methodologies. In *2015 37th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, Reno, NV, USA. IEEE. (cited page 32).

Verweij, J. and Klootwijk, J. (1996). Dielectric breakdown I: A review of oxide breakdown. *Microelectronics Journal*, 27(7):611–622. (cited page 97).

Viale, B. and Allard, B. (2020). Scalable and Versatile Design Guidance Tool for the ESD Robustness of Integrated Circuits - Part I. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):3067–3080. (cited pages 28, 31, 31, 32, 32, 32, 32, 33, 33, 37, and 39).

Viale, B., Fer, M., Courau, L., Galy, P., Jacquier, B., Lescot, J., and Allard, B. (2016). An automated tool for chip-scale ESD network exploration and verification. In *2016 38th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, pages 1–10, Garden Grove, CA, USA. IEEE. (cited pages 32, 32, 33, 37, and 39).

Visser, W., Havelund, K., Brat, G., Park, S., and Lerda, F. (2003). Model checking programs. *Automated software engineering*, 10:203–232. (cited page 9).

Vollertsen, R.-P. and Wu, E. (2004). Voltage acceleration and t63. 2 of 1.6–10 nm gate oxides. *Microelectronics Reliability*, 44(6):909–916. (cited page 105).

Wing, J. M. (1989). *What is a formal method?* Carnegie-Mellon University. Department of Computer Science. (cited page 5).

Yadav, A., Jindal, P., and Basappa, D. (2020). Study and analysis of RTL verification tool. In *2020 IEEE Students Conference on Engineering & Systems (SCES)*, pages 1–6. IEEE. (cited page 19).

Ytterdal, T., Cheng, Y., and Fjeldly, T. A. (2003). *Device Modeling for Analog and RF CMOS Circuit Design*. Wiley, 1 edition. (cited pages 67 and 138).

Zwerger, M. and Graeb, H. (2012). Short-Circuit-Path and Floating-Node Verification of Analog Circuits in Power-Down Mode. In *2012 International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pages 241–244, Seville, Spain. IEEE. (cited pages 2, 21, 23, 33, 34, 34, 34, 37, 39, and 58).

Zwerger, M. and Graeb, H. (2014). Verification of the power-down mode of analog circuits by structural voltage propagation. *Analog Integrated Circuits and Signal Processing*, 78(1):177–189. (cited pages 2 and 30).

Zwerger, M. and Graeb, H. (2015). Detection of Asymmetric Aging-Critical Voltage Conditions in Analog Power-Down Mode. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1269–1272, Grenoble, France. IEEE Conference Publications. (cited pages 32, 33, 34, 34, 34, and 39).

Zwerger, M., Neuner, M., and Graeb, H. (2017). Analog Power-Down Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(12):1954–1967. (cited pages 35 and 37).

Glossary of Notations

Boolean Logic

$\mathbb{B}$	set of Boolean values	with $\mathbb{B} = \{\top, \bot\}$
$\top$	truth value for *true*	
$\bot$	truth value for *false*	
$\neg$	logical negation	
$\wedge$	logical conjunction	
$\vee$	logical disjunction	
$\Rightarrow$	logical implication	
$\Leftrightarrow$	logical equivalence	
$\models$	'satisfies'	e.g., $M \models F$ if M is a *model* of Boolean formula F

Mathematical Notations

$\emptyset$	empty set
$\lvert \cdot \rvert$	cardinal of a set
$\cup$	union among sets
$\cap$	intersection of sets
$\subseteq$	subset of set
$\mathscr{P}$	function which gives the powerset of a set
$\mathbb{1} \colon \mathbb{B} \longrightarrow \{0, 1\}$	Indicator function

Circuit Semantics

$\mathcal{S}$	circuit semantics (generic)
$\mathcal{S}^o$	oriented signal circuit semantics
$\mathcal{S}^s$	switch-based circuit semantics, without threshold
$\mathcal{S}^t$	switch-based circuit semantics, with threshold
$\mathcal{S}^q$	quantitative circuit semantics

Sets

$\mathbf{N}$	set of nets	
$\mathbf{P}$	set of pins	$\mathbf{P} \subset \mathbf{N}$
$\mathbf{S}$	set of supplies	$\mathbf{S} \subset \mathbf{P}$
$\mathbf{I}$	set of inputs	$\mathbf{I} \subset \mathbf{P}$
$\mathbf{O}$	set of outputs	$\mathbf{O} \subset \mathbf{P}$
$\mathbf{G}$	set of transistor gate	$\mathbf{G} \subset \mathbf{N}$
$\mathbf{D}$	set of devices	$\mathbf{D} = \mathbf{D}_{\texttt{MOS}} \cup \mathbf{D}_{\texttt{DIO}} \cup \mathbf{D}_{\texttt{RES}}$
$\mathbf{D}_{\texttt{MOS}}$	set of transistors	$\mathbf{D}_{\texttt{MOS}} = \mathbf{D}_{\texttt{NMOS}} \cup \mathbf{D}_{\texttt{PMOS}}$
$\mathbf{D}_{\texttt{NMOS}}$	set of NMOS transistors	$\mathbf{D}_{\texttt{NMOS}} \subset \mathbf{D}_{\texttt{MOS}}$
$\mathbf{D}_{\texttt{PMOS}}$	set of PMOS transistors	$\mathbf{D}_{\texttt{PMOS}} \subset \mathbf{D}_{\texttt{MOS}}$
$\mathbf{D}_{\texttt{DIO}}$	set of diodes	$\mathbf{D}_{\texttt{DIO}} \subset \mathbf{D}$
$\mathbf{D}_{\texttt{RES}}$	set of resistors	$\mathbf{D}_{\texttt{RES}} \subset \mathbf{D}$
$\widehat{\mathbf{N}}$	set of virtual drain nets in $\mathcal{S}^o$	
$\mathbb{V}$	set of voltage values	
$\mathbb{I}$	set of current values	
$\mathbb{P}$	set of probabilities	
$\mathbb{T}$	set of age (t) valuations	
$\mathbb{Q}$	set of rational values	

Static functions

$\textsc{Gate}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N}$	gate of a transistor
$\textsc{Src}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N}$	source of a transistor
$\textsc{Drn}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbf{N}$	drain of a transistor
$\widehat{\textsc{Drn}}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \widehat{\mathbf{N}}$	virtual drain of a transistor in $\mathcal{S}^o$
A	:	$\mathbf{D}_{\texttt{RES}} \longrightarrow \mathbf{N}$	first resistor terminal in lexicographical order
B	:	$\mathbf{D}_{\texttt{RES}} \longrightarrow \mathbf{N}$	second resistor terminal in lexicographical order
R	:	$\mathbf{D}_{\texttt{RES}} \longrightarrow \mathbb{Q}_{\geq 0}$	resistor's resistance value
$\textsc{Anode}$	:	$\mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbf{N}$	anode of a diode
$\textsc{Cathode}$	:	$\mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbf{N}$	cathode of a diode
$\textsc{Neighbors}_{\texttt{MOS}}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D}_{\texttt{MOS}} \times \mathbf{N})$	net's transistor neighbors
$\textsc{Neighbors}_{\texttt{RES}}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D}_{\texttt{RES}} \times \mathbf{N})$	net's resistor neighbors
$\textsc{Neighbors}_{\texttt{DIO}}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D}_{\texttt{DIO}} \times \mathbf{N})$	net's diode neighbors
$\textsc{Neighbors}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\mathbf{N} \times \mathbf{D} \times \mathbf{N})$	neighbors of a net
$\textsc{ReachSuppl}^{\natural}$	:	$\mathbf{N} \times \mathscr{P}(\mathbf{N}) \longrightarrow \mathscr{P}(\mathbf{S})$	helper function for net reachable supplies
$\textsc{ReachSuppl}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\mathbf{S})$	a net's reachable supplies
$\textsc{ReachGates}^{\natural}$	:	$(\mathbf{P} \setminus \mathbf{S}) \times \mathscr{P}(\mathbf{N}) \longrightarrow \mathscr{P}(\mathbf{G})$	helper function for net reachable transistor gates
$\textsc{ReachGates}$	:	$(\mathbf{P} \setminus \mathbf{S}) \longrightarrow \mathscr{P}(\mathbf{G})$	a net's reachable transistor gates
$\textsc{Driv}$	:	$\mathbf{N} \longrightarrow \mathscr{P}(\widehat{\mathbf{N}})$	net's drivers in $\mathcal{S}^o$
$\textsc{Driv}_{\text{const}}$	:	$\mathbf{N} \longrightarrow \widehat{\mathbf{N}}$	net's consistent drivers in $\mathcal{S}^o$
$\textsc{Driv}_{\text{short}}$	:	$\mathbf{N} \longrightarrow \widehat{\mathbf{N}}$	net's short-circuited drivers in $\mathcal{S}^o$
$\textsc{Driv}_{\text{float}}$	:	$\mathbf{N} \longrightarrow \widehat{\mathbf{N}}$	net's floating drivers in $\mathcal{S}^o$
$\textsc{AvailSuppl}$	:	$\mathbf{S} \longrightarrow \mathscr{P}(\mathbb{V})$	set of available supplies values of a supply pin
$\textsc{allowedInterval}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \{[a,b], (a,b) \in \mathbb{V}^2\}$	transistor's allowed range of the voltage applied

Dynamic functions

Semantics Common Rules

$\mathcal{R}_{\substack{\text{static}\\\text{domain}}}$	nets' static domain (all semantics)
$\mathcal{R}_{\mathbf{S}}$	supply constraints
$\mathcal{R}_{\mathbf{I}}$	input constraints
$\mathcal{R}_{\mathbf{I}}$ strict	strict input constraints
$\mathcal{R}_{\mathbf{I}}$ relaxed	relaxed input constraints

Semantics $\mathcal{S}^o$-specific Rules

$\mathcal{R}^o_{\texttt{NMOS}}$	NMOS device behavior
$\mathcal{R}^o_{\texttt{PMOS}}$	PMOS device behaviorǎ
$\mathcal{R}^o_{\text{MERGE}}$	rule for merging net's drivers

Semantics $\mathcal{S}^s$-specific Rules

$\mathcal{R}_{\substack{\text{local}\\\text{voltage}}}$	nets' local voltage

Semantics $\mathcal{S}^t$-specific Rules

$\mathcal{R}^t_{\substack{\text{local}\\\text{voltage}}}$	nets' local voltage

Semantics $\mathcal{S}^q$-specific Rules

$\mathcal{R}^{\texttt{MOS}}_{\text{regions}}$	transistor regions constraints
$\mathcal{R}^{\texttt{DIO}}_{\text{regions}}$	diode regions constraints
$\mathcal{R}^{\texttt{RES}}_{\text{current}}$	resistor current constraints
$\mathcal{R}_{\text{regions}}$	device regions of operation
$\mathcal{R}_{\text{Kirchhoff}}$	Kirchhoff's current law
$\mathcal{R}_{\substack{\text{current}\\\text{sign}}}$	current sign integrity

Boolean Predicates

$\mathcal{O}n$	:	$\mathbf{D}_{\texttt{MOS}} \cup \mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbb{B}$	tells whether a transistor is turned on in semantics $\mathcal{S}^s$
$\mathcal{O}n^t$	:	$\mathbf{D}_{\texttt{MOS}} \cup \mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbb{B}$	tells whether a transistor is turned on in semantics $\mathcal{S}^t$
$\mathcal{C}ut$	:	$\mathbf{D}_{\texttt{MOS}} \cup \mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbb{B}$	tells whether a transistor or diode in the cut-off mode
$\mathcal{L}in$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$	tells whether a transistor in the linear mode
$\mathcal{S}at$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$	tells whether a transistor in the saturation mode
$\mathcal{F}wd$	:	$\mathbf{D}_{\texttt{DIO}} \longrightarrow \mathbb{B}$	tells whether a diode in the forward mode
$\mathcal{S}tatus$	:	$\mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \{\mathsf{Const}, \mathsf{Short}, \mathsf{Float}\}$	net 'status' in $\mathcal{S}^o$
$\mathcal{C}onst$	:	$\mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B}$	tells whether a net has a consistent voltage in semantics $\mathcal{S}^o$
$\mathcal{S}hort$	:	$\mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B}$	tells whether a net is in short-circuit in semantics $\mathcal{S}^o$
$\mathcal{F}loat$	:	$\mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{B}$	tells whether a net is floating in semantics $\mathcal{S}^o$
EqShort	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether all drivers of a net that are in short-circuit are equal in $\mathcal{S}^o$
EqConst	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether all consistent drivers of a net are equal in $\mathcal{S}^o$
AllFloat	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether all drivers of a net are floating in $\mathcal{S}^o$
DiffShort	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether not all non-floating drivers of a net are equal in $\mathcal{S}^o$
currentFlows	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether current flows through a net
noCurrent	:	$\mathbf{N} \longrightarrow \mathbb{B}$	tells whether current does not flow through a net
$\mathcal{E}_{\texttt{MLS}}$	:	$\mathbf{D}_{\texttt{PMOS}} \longrightarrow \mathbb{B}$	missing level-shifter (MLS) error
$\mathcal{A}_{\texttt{EOS}}$	:	$\mathbf{D}_{\texttt{MOS}} \longrightarrow \mathbb{B}$	absence of electrical overstress (EOS) error

Rationals

$\mathcal{V}$: $\mathbf{N} \longrightarrow \mathbb{V}$ voltage of a net
$\mathcal{V}^o$: $\mathbf{N} \cup \widehat{\mathbf{N}} \longrightarrow \mathbb{N}$ voltage abstraction of a net in $\mathcal{S}^o$
$\mathcal{V}^o_{\hat{\mathrm{D}}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{N}$ voltage abstraction of transistor virtual drain in $\mathcal{S}^o$
$\mathcal{V}^o_{\mathrm{G}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{N}$ voltage abstraction of transistor gate in $\mathcal{S}^o$
$\mathcal{V}^o_{\mathrm{S}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{N}$ voltage abstraction of transistor source in $\mathcal{S}^o$
$\mathcal{V}^o_{\min}$: $\mathbf{N} \longrightarrow \mathbb{N}$ minimum voltage among the drivers of a net in $\mathcal{S}^o$
$\mathcal{V}^o_{\max}$: $\mathbf{N} \longrightarrow \mathbb{N}$ maximum voltage among the drivers of a net in $\mathcal{S}^o$
$\mathcal{V}_{\mathrm{G}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage of transistor's gate
$\mathcal{V}_{\mathrm{S}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage of transistor's source
$\mathcal{V}_{\mathrm{D}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage of transistor's drain
$\mathcal{V}_{\mathrm{GS}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage difference between the gate and source of a transistor
$\mathcal{V}_{\mathrm{GD}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage difference between the gate and drain of a transistor
$\mathcal{V}_{\mathrm{DS}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage difference between the drain and source of a transistor
$\mathcal{V}_{\mathrm{SD}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{V}$ voltage difference between the source and drain of a transistor
$\mathcal{V}_{\mathrm{A}}$: $\mathbf{D}_{\mathtt{RES}} \rightharpoonup \mathbb{V}$ voltage of resistor's first (A) terminal
$\mathcal{V}_{\mathrm{B}}$: $\mathbf{D}_{\mathtt{RES}} \longrightarrow \mathbb{V}$ voltage of resistor's second (B) terminal
$\mathcal{V}_{\mathrm{AB}}$: $\mathbf{D}_{\mathtt{RES}} \longrightarrow \mathbb{V}$ voltage difference across a resistor
$\mathcal{V}_{\mathrm{A}}$: $\mathbf{D}_{\mathtt{DIO}} \rightharpoonup \mathbb{V}$ voltage of diode's anode
$\mathcal{V}_{\mathrm{K}}$: $\mathbf{D}_{\mathtt{DIO}} \longrightarrow \mathbb{V}$ voltage of diode's cathode
$\mathcal{V}_{\mathrm{AK}}$: $\mathbf{D}_{\mathtt{DIO}} \longrightarrow \mathbb{V}$ voltage difference between the anode and cathode of a diode
$\mathcal{I}$: $\mathbf{N} \times \mathbf{D} \times \mathbf{N} \longrightarrow \mathbb{I}$ current from a net to another through a device
$\mathcal{I}_{\mathrm{SD}}$: $\mathbf{D}_{\mathtt{MOS}} \longrightarrow \mathbb{I}$ source-to-drain current of a transistor
$\mathcal{I}_{\mathrm{AB}}$: $\mathbf{D}_{\mathtt{RES}} \longrightarrow \mathbb{I}$ current through a resistor
$\mathcal{I}_{\mathrm{AK}}$: $\mathbf{D}_{\mathtt{DIO}} \longrightarrow \mathbb{I}$ anode-to-cathode current of a diode

Reliability

δ voltage stress applied on a transistor for a voltage valuation
$\delta_{\mathcal{V}}$ voltage stress applied on a transistor for a given voltage valuation $\mathcal{V}$
$\mathscr{R}_{\mathrm{MOS}}$ reliability of a transistor
$\mathscr{R}_{\mathrm{CIRC}}$ reliability of a subset of transistors composing a circuit
$\mathscr{R}$ reliability of a circuit
$\tilde{\mathscr{R}}$ reliability of a circuit computed from piecewise approximation of device reliabilities
I_k intervals of the piecewise affine approximation of device reliabilityă
$V_{\min}$ minimum voltage considered for device reliability linearisation
$V_{\max}$ maximum voltage considered for device reliability linearisation
ω logarithm of device reliability function
Ω linearised logarithm of device reliability function
`minimize` objective minimization function
`maximize` objective maximization function
$\tilde{\mathscr{R}}^{\min}$ minimum linearised circuit reliability
$\mathbb{1}$ Indicator function
t_{ref} reference value for age (in years)
α time to failure
β Weibull factor
$\mathscr{R}_{\mathrm{ref}}$ reference value for circuit reliability

Constants

V_t	threshold voltage
$\frac{W}{L}$	channel's width-to-length ratio
K_p	the transconductance parameter)
λ	charge-carrier effective mobility parameter
E_a	device activation energy
γ	voltage acceleration parameter
A_0	scale factor
T	device temperature

Index

A

Abstract interpretation (AI), 9–11
Analog-to-digital converter (ADC), 83, 105
Application-specific integrated circuit (ASIC), 1

B

Binary decision diagram (BDD), 8, 9
Bounded model checking (BMC), 8, 9

C

Charged-device model (CDM), 28
Circuit description language (CDL), 19, 20, 26, 41, 43, 44, 50
Circuit design flow, 17
Clock domain crossing (CDC), 26, 28
Complementary metal-oxide-semiconductor (CMOS), 21, 23, 25, 36, 51, 54–57, 83, 86, 101, 116, 131
Computer-aided design (CAD), 19
Conflict-driven clause learning (CDCL), 12, 13
Conjunctive normal form (CNF), 12
Convention Industrielle de Formation par la Recherche (CIFRE), iv, 3, 120, 123
Counterexample-guided abstraction refinement (CEGAR), 8, 28, 36, 37, 121

D

Davis-Putnam-Logemann-Loveland algorithm (DPLL), 12, 13
Design rule checking (DRC), 1, 19
Dynamic voltage and frequency scaling (DVFS), 18

E

Electrical overstress (EOS), 3, 25, 31, 34, 39, 42, 65, 67, 81, 84, 87–95, 120
Electrical rule checking (ERC), 1, 3, 16, 19, 28, 32, 34, 36–38, 41, 86, 119
Electrostatic discharge (ESD), 19, 28, 31, 32, 34, 37, 39

F

Floating net, 21, 23, 24, 31, 33, 35, 36, 39, 53, 60, 61, 63, 91, 101, 105

H

Human-body model (HBM), 28, 39

I

Input/output (I/O), 44, 50
Integrated circuit (IC), ix, 1, 17, 19, 41, 95
Intellectual property (IP), 33, 44, 119
Intensity-Voltage (I-V) characteristics, 29, 37, 64, 65, 78, 79, 111, 119, 120, 133, 138, 140

J

Joint Electron Device Engineering Council (JEDEC), 97–99

L

Layout level, 17
Layout-versus-schematic (LVS), 19
Linear real arithmetic (LRA), 16, 42, 58, 64, 67, 70, 96, 97
Linear temporal logic (LTL), 6

M

Metal-oxide-semiconductor field-effect transistor (MOSFET), 19, 20, 34, 54, 68, 69
Missing level-shifter (MLS), 3, 25, 36, 39, 42, 81–84, 90, 120
Model checking (MC), 5, 6, 8, 9, 11
Multiple clock domain (MCD), 18

N

N-channel MOSFET (NMOS), 20, 21, 45, 54, 56, 59, 64, 79, 82, 93, 99, 128, 129, 134
Non-linear real arithmetic (NRA), 14, 101, 139

O

Optimization modulo theories (OMT), ix, xi, 14, 16, 38, 42, 43, 94–96, 100–104, 110–112, 116, 117, 120, 121

P

P-channel MOSFET (PMOS), 20, 21, 45, 54, 56, 59, 61, 82, 83, 93, 97, 99, 128, 129, 134

Power-down mode, 18, 33–36, 44, 52, 56

Q

Quantifier-free (QF), 12, 14, 16

R

Register-transfer level (RTL), 1, 17–19, 23, 35–37

S

Satisfiability (SAT), 1, 2, 8, 11, 16, 37, 38, 86, 93

Satisfiability modulo theories (SMT), ix, xi, 2, 3, 12, 14–16, 37, 38, 41–43, 49, 64, 67, 70, 78, 79, 81, 84, 92, 93, 95, 96, 101, 103, 104, 117, 119, 121

Schichman-Hodges (SPICE Level 1) model, 30, 67, 109

Short-circuit, 19, 21, 23–26, 28, 31, 33, 35–39, 45, 51, 53, 55, 56, 60, 61, 63, 65, 66, 79, 86, 101, 104, 105

Simulation Program with Integrated Circuit Emphasis (SPICE), 21, 22, 28, 65–67, 78, 90–92, 100, 105, 109, 110, 117, 120

SMT-LIB, 11, 12, 14, 15

Sum of failure rates (SOFR), 117

System on a chip (SoC), 17–19, 26, 31, 32, 44

T

Theorem proving, 11

Time-dependent dielectric breakdown (TDDB), 3, 39, 42, 95, 97, 100, 117

Time-dependent oxide breakdown (TDOB), 97

Time-to-failure (TTF), 97, 98, 105

Transaction level modeling (TLM), 19

Transistor level, 1, 17, 41, 119

U

Uninterpreted functions (UF), 12

User-aided abstraction refinement (UsAAR), 28

V

Very-large-scale integration (VLSI), 17, 23

W

Weibull distribution, 98, 99, 117